Ph.D. Thesis

2015

Solving Differential Equations with Evolutionary Algorithms

José María Chaquet Ulldemolins

Master in Advanced Artificial Intelligence by

Universidad Nacional de Educación a Distancia

Doctoral Program in Intelligence Systems

Advisor

Enrique J. Carmona Suárez

Universidad Nacional de Educación a Distancia

Departamento de Inteligencia Artificial

Escuela Técnica Superior de Ingeniería Informática

Solving Differential Equations with Evolutionary Algorithms

José María Chaquet Ulldemolins

Industrial Engineer by Universidad Politécnica de Madrid

Master in Advanced Artificial Intelligence by UNED

Advisor

Enrique J. Carmona Suárez

Associate Professor in Artificial Intelligence Department

at Universidad Nacional de Educación a Distancia

This document has been generated
using *Lyx*, LaTeX, *Veusz* and
other open-source tools.

"If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is."

-John von Neumann.

For María, Javier and Alicia

Acknowledgments

I would like to express my gratitude to my thesis director, Enrique, for all the time dedicated to me. Our relationship started during my Master degree studies in *Artificial Intelligence* and continue until today. I remember several times when the security guard of the school kindly *invited* us to leave the facilities because it was so late in the night. And thanks to Enrique for allowing me to use several computers installed in his office. This allow me to complete the calculations needed for this work in time.

And of course, I would like to acknowledge my wife María for her generosity and support. Thanks to her I could allocate the time needed to complete this thesis. To be honest, I think the more challenging issue was sharing my time between my duties at work in ITP, at home with my little kids and with my research activities. Without the support of Maria all these would have been impossible.

Summary

Many fundamental laws of Physics and Chemistry can be formulated as differential equations (DEs) and they are useful to model different problems in fields as Biology, Economics or Engineering. Some DEs admit solutions given by explicit formulae. However, in the general case, only approximated solutions can be found. Several methods to find approximated solutions to DEs are available in the literature. The most extended ones are numerical methods. However, in the present dissertation we focus the attention in non-standard methods, called by some authors *heuristic* methods.

In this thesis, first a survey of heuristic algorithms to solve DEs is presented. In this type of methods, the original problem, solving a DE, is transformed into an optimization problem. The new problem involves seeking a function, solution of the DE, which minimizes a cost function defined with the boundary conditions and with the DE itself. To solve the new problem, candidate solutions are expressed using functions basis, parametric kernels or generic mathematical expressions. The search of the optimum solution is performed by means of Evolutionary Algorithms.

The main contribution of the present thesis is twofold. In the one hand, several studies about the optimum way of representing the candidate solutions are presented. On the other hand, the election of the most efficient evolutionary algorithm for tuning the parameters which express the solution is provided. We have also focused in developing a general framework to solve different types of DEs, such as ordinary (liner and non-linear), partial and systems of DEs.

To show the performance of the proposed algorithms, they are applied on a set of 32 DEs

extracted from the literature and a comparison with other heuristic and numerical methods are commented. Good behavior is observed, DEs are correctly solved achieving competitive performance. A lower computational effort compared with other heuristic methods is also observed.

Numerical methods are very efficient, well developed and can cope with the majority of real problems. However, from the moment in which the problem of solving a differential equation is transformed in an optimization problem, the proposed methods, in particular, and evolutionary algorithms, in general, have interesting properties (mesh-free, mathematical function as solution, same algorithm for different families of equations, etc) that can be useful in this application domain. On the other hand and from a general view, the main drawbacks of evolutionary methods is that there is not guarantee of which accuracy can be obtained and the low convergence speed because a population of candidate solutions has to be handled.

Resumen (Summary in Spanish)

Una gran cantidad de leyes fundamentales de Física y Química se formulan mediante *ecuaciones diferenciales* (EDs), las cuales son además útiles a la hora de modelizar diferentes problemas en disciplinas tales como Biología, Economía o Ingeniería. Algunas EDs admiten soluciones cerradas o exactas. Sin embargo, la gran mayoría sólo pueden resolverse de forma aproximada. Existen en la literatura gran cantidad de métodos para encontrar dichas aproximaciones. Los más extendidos son los denominados métodos numéricos. En la presente tesis nos centraremos en otro tipo de métodos no estándar, llamados por algunos autores métodos *heurísticos*.

En este trabajo se expone un estudio de las algoritmos heurísticos presentes en la literatura para resolver EDs. En este tipo de métodos, el problema original (resolver una ED) se transforma en un nuevo problema de optimización. El nuevo problema conlleva encontrar una función, solución de la ED, que minimiza una función de coste construida con las condiciones de contorno y con la propia ED. Para resolver el nuevo problema, las soluciones candidatas se pueden expresar mediante una base funcional, *kernels* paramétricos o mediante expresiones matemáticas genéricas. La búsqueda de la solución óptima al problema se realiza mediante algoritmos evolutivos.

Las principales contribuciones de la presente tesis se pueden agrupar en dos ideas básicas. Por un lado, se realizan varios estudios para determinar el modo óptimo de representar las soluciones candidatas. Por otro lado, se expone cuál es el algoritmo evolutivo más eficiente para ajustar los parámetros que expresan la solución. Los algoritmos expuestos tratan de ser además genéricos: son capaces de resolver, manteniendo los parámetros de control, una

gran variedad de EDs, tales como ecuaciones ordinarias (lineales y no lineales), ecuaciones en derivadas parciales y sistemas de ecuaciones.

Para estudiar el comportamiento de los algoritmos propuestos, se han aplicado a un conjunto de 32 EDs extraídas de la literatura. También se presenta una comparación con otros métodos heurísticos y numéricos. Se observa un buen comportamiento de los algoritmos propuestos, las EDs se han resuelto correctamente alcanzando resultados competitivos con menor coste computacional comparado con otros métodos heurísticos.

Los métodos numéricos son eficientes, maduros y capaces de resolver la mayoría de los problemas reales. Sin embargo, en cuanto el problema de resolver una ecuación diferencial se transforma en un problema de optimización, los métodos propuestos cuentan con propiedades interesantes (métodos no basados en malla, solución simbólica, mismos algoritmos para diferentes familias de ecuaciones, etc) que pueden resultar útiles en este campo de conocimiento. Por otro lado y desde un punto de vista general, las principales desventajas de los métodos evolutivos son que el nivel de convergencia no está garantizado y el mayor coste computacional por usar una estrategia poblacional.

Contents

List of Figures

List of Tables

List of Acronyms

This is a list of acronyms used in this thesis:

ANN Artificial Neural Network

ANFIS Adaptive Network-based Fuzzy Inference System

ASA Active-set Algorithm

BFGS Broyden-Fletcher-Goldfarb-Shanno Algorithm

BNF Backus-Naur Form

CA Cellular Automata

CAS Computer Algebra System

CGP Cartesian Genetic Programming

CNN Cellular Neural Network

CMA-ES Covariance Matrix Adaptation Evolution Strategies

DAE Differential Algebraic Equation

DDE Delay Differential Equation

DE Differential Equation

DEM Diffusive Element Method

DESCMA-ES Differential Equation Solver based on CMA-ES

DESES Differential Equation Solver based on Evolution Strategies

DESGE Differential Equation Solver based on Grammatical Evolution

DEv Differential Evolution

DS Downhill Simplex

EC Evolutionary Computation, a. k. a. Evolutionary Computing

ED Ecuación Diferencial (Differential Equation in Spanish)

ES Evolution Strategies

DE Differential Equation

FEM Finite Element Method

FDE Fractional Differential Equation

FDM Finite Difference Method

FENN Finite-Element Neural Network

FNN Fuzzy Neural Network

FVM Finite Volume Method

GA Genetic Algorithm

GDC Greatest Common Divisor

GE Grammatical Evolution

GEP Gene Expression Programming

GP Genetic Programming

GSD Gradient Steepest Descent

LGP Linear Genetic Programming

LHS Left Hand Side

LHSE LHS Enhanced

LS-SVM Least Squares Support Vector Machine

MEP Multi-Expression Programming

ODE Ordinary Differential Equation

PDE Partial Differential Equation

PIM Point Interpolation Method

PRC Persistent Random Constants

PS Pattern Search

RES Robust Evolution Strategies

RK Runge Kutta method

RMSE Root of the Mean Squared Error

SA Simulated Annealing

SDE Stochastic Differential Equation

SFDE System of Fuzzy Differential Equations

SPH Smoothed Particle Hydrodynamics

SVM Support Vector Machine

Chapter 1

Introduction

In this first chapter, the motivation of the thesis will be exposed in Section 1.1. Then, in Section 1.2 the scope of this work will be commented. The problem statement and research goals of this dissertation are summarized in Section 1.3. The research methodology is described in 1.4. Finally, the structure of the thesis is described in 1.5.

1.1 Motivation

Part of the extraordinary progress of science in the modern era can be explained by the discovery of *Calculus* in the 17th century by Isaac Newton and Gottfried Wilhelm Leibniz, although elements of it have appeared in ancient Greece, China, medieval Europe, India, and the Middle East. Calculus is the mathematical study of change and makes use of the fundamental notions of convergence of infinite sequences and infinite series to a well-defined limit.

One important concept managed in Calculus is the *Derivative*, which measures the sensitivity to change of a quantity which is determined by another quantity. The process of finding a derivative is called differentiation. A *differential equation* (DE) is just an algebraic relation between functions and their derivatives. These mathematical entities allow scientists to understand a wide range of complex phenomena. Many fundamental laws of Physics and Chemistry can be formulated as DEs. And they are useful to model different problems in a

lot of scientific fields, such as Biology, Economics or Engineering.

DEs are mathematically studied from several different perspectives, mostly concerned with their solution, i. e., the set of functions that satisfy the equation. Only the simplest DEs are solvable by explicit formulas. But normally only approximated solutions can be obtained. The most extended methods to solve DEs make use of numerical analysis. For that, the equation is discretized bringing it into a finite-dimensional subspace. This can be done by a finite element, a finite difference, or a finite volume methods reducing the problem to the solution of an algebraic equation.

In the late years of 20th century new approaches to solve DEs were reported. They try to solve DEs with *non-standard* algorithms. By non-standard methods we mean those not inspired in numerical methods and therefore, less extended in the literature. Other authors call these type of algorithms *heuristic* methods. Although normally they are less efficient than numerical methods, heuristics have some advantages regarding set-up of the problem, storing requirements, interpolation properties, etc.

Although a great progress has been produced in the context of methods to solve DEs, several issues are still challenging. Mesh-based methods like finite element analysis have become a traditional engineering tool. However, they still have some drawbacks, such as difficulties in mesh generation and mesh distortion in large deformation analysis of structures. This causes researchers and engineers to search for alternative numerical approaches and the so-called meshless methods were originated. That method uses only distribution of nodes on a differential equation domain to achieve its approximate solution. Over the last few decades, there have been numerous meshless methods proposed to solve a number of engineering applications. The meshless methods, despite having some advantages, have some weak points such as difficulties in dealing with boundary conditions. Other limitations of numerical methods (messless or mesh-based) are that the solution is given by numerical values at the collocation points used to solve the equation. Therefore, the solution is only valid in those points, needing interpolation if the solution is needed in different collocation points. Although several very efficient methods are reported, normally based on numerical analysis, they only are valid to a very specific equation, or at least, a family of equations. If the equation to

solve is changed, the method must be adjusted, or even it must be completely modified.

A great number of contributions reported in recent literature try to cover the limitations of current methods. Thus, regarding numerical analysis, several lines can be distinguished, such as improvements in the mesh generation, high order methods, preconditioning, parallelization or implicit methods. Heuristic methods focus their efforts in two different issues. In the one hand, how coding the candidate solutions. For instance, cellular automata, polynomials or neural networks have been reported. On the other hand, several optimization methods have been employed to tune the unknowns parameters which code the solution: backpropagation, Genetic Algorithm, Swarm Optimization or Genetic Programming among others.

Focusing in the contribution of heuristic methods, it is difficult to make a quantitative comparison because only fitness values are provided, which are high dependent on the solver parameters and on how differential equations is transformed into an optimization problem. Other limitation which can be commented is that normally heuristic methods are applied on a few problems, or to very simple equations. So a lack of generality in the benchmarking tests is observed. A common drawback of the works reported in the literature is that a lot of computational effort is needed to solve relative simple equations, making their use in realistic applications very difficult. It is observed as well in the literature a lack of information about the convergence process. Due to the stochastic properties of heuristic algorithms, several runs must be performed and mean and dispersion values should be provided to actually study the behavior of the different approaches.

1.2 Scope of the Thesis

The main goal of the present thesis is to develop new algorithms based on Evolutionary Computing to solve a wide variety of DEs. In the following lines, these concepts will be extended.

Although some DEs admit exact solutions, the majority of equations only can be approximately solved. There are in the literature several approaches to find approximated solution of DEs. The present work can be classified inside what some authors call *heuristic* methods,

opposite to numerical methods which reduce the original problem into a set of algebraic equations. Inside heuristic methods, we are interested in *Evolutionary Computation* (EC), a subfield of artificial intelligence that deals with optimization problems. Its algorithms can be considered global optimization methods with a metaheuristic or stochastic optimization character and are mostly applied for black box problems (no derivatives known), often in the context of expensive optimization. For that, our original problem (solving a DE) must be transformed into an optimization one. This is done defining a proper fitness function based on the DE itself and its boundary conditions. Then, a population of candidate solutions is evolved trying to obtain the best solution in terms of the fitness function.

Evolutionary algorithms are inspired in biological process and have a set of common characteristics. Thus, first an initial population of individuals is randomly generated, giving the first generation. Then, the fitness function of each individual in that population is evaluated. On each generation, several operations are repeated until some stop condition is fulfilled. Therefore, in each generation the best-fit individuals are selected for reproduction. An offspring population is generated using mutation and crossover operators. Finally, least-fit individuals are replaced with new individuals, although several survival mechanism can be adopted.

DEs is a wide concept and several types can be enumerated. Generally speaking, we can distinguish between classic DEs, that is, deterministic DEs, and not classic ones. In the second group we can find stochastic and fuzzy DEs. In this work we only deal with classic equations. Among these deterministic equations exist several types such as ordinary and partial equations, linear and non-linear or system of DEs. We are interested in all of these types.

On the contrary than numerical methods, we will build the candidate solutions using symbolic expressions. These expressions can be generated using functions basis, parametric kernels or generic algebraic expressions. It is important to highlight that even in those cases where no exact solution exist, the symbolic expression obtained will be used to approximate the solution. Having symbolic expressions are very attractive regarding storing requirements or for obtaining numerical values at points different from those used to find the solution.

There are several paradigms in Evolutionary algorithms. According to the characteristic of the problem to solve, in the present work we have employed Grammatical Evolution (GE), a recent variation of genetic programming, and Evolution Strategies (ES). The first algorithm seeks the best combination of symbolic functions to expressed the candidate solutions, whereas the ES focus on the the best parameterization of the solution given a function basis or function kernels, due to its good behavior dealing with optimization problems of real-valued functions.

1.3 Problem Statement and Research Goals

From a mathematical point of view, the problem to be solved can be stated as following: we consider the general equation

$$\mathbf{L}\mathbf{y}\left(\mathbf{x}\right) = \mathbf{f}\left(\mathbf{x}\right) \ \ in\ \Omega \subset \mathbb{R}^d \tag{1.1}$$

subject to the boundary conditions

$$\mathbf{B}\mathbf{y}\left(\mathbf{x}\right) = \mathbf{g}\left(\mathbf{x}\right) \ \ on\ \partial\Omega, \tag{1.2}$$

where $\mathbf{L}$ and $\mathbf{B}$ are differential operators in the space $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y}\left(\mathbf{x}\right)$ denotes the unknown solution vector. Functions $\mathbf{f}\left(\mathbf{x}\right)$ and $\mathbf{g}\left(\mathbf{x}\right)$ denote source terms, so only depend on $\mathbf{x}$, but not on $\mathbf{y}$ or its derivatives. From a general point of view, $\mathbf{y}\left(\mathbf{x}\right)$, $\mathbf{f}\left(\mathbf{x}\right)$ and $\mathbf{g}\left(\mathbf{x}\right)$ belong to the set of vector-valued functions $\mathbb{R}^d \rightarrow \mathbb{R}^m$. On the other hand, $\Omega \subset \mathbb{R}^d$ is a bounded domain and $\partial\Omega$ denotes its boundary. Strictly speaking, this notation corresponds to elliptic equations, where the boundary conditions must be imposed in all the boundary of the domain. In other problems, such as initial value differential equations, the boundary conditions are given in a subset of $\partial\Omega$. Note that if $d = 1$ and $m = 1$, we have an ordinary differential equation (ODE) problem, which can be linear (LODE) or non linear (NLODE). If $d = 1$ and $m > 1$, a system of differential equations (SODE) problem is managed. Finally, if $d > 1$ and $m = 1$, a

partial differential equation (PDE) problem is established. The solution satisfying (1.1) and (1.2) can be computed solving the following *Constrained Optimization Problem*:

$$\begin{aligned} &Minimize: && \int_{\Omega} \|\mathbf{L}\mathbf{y}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\|^2 \, d\mathbf{x} \\ &Subject\ to: && \int_{\partial\Omega} \|\mathbf{B}\mathbf{y}(\mathbf{x}) - \mathbf{g}(\mathbf{x})\|^2 \, d\mathbf{x} = 0 \end{aligned} \tag{1.3}$$

where $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^d$ space.

The problem is then discretized using a set of n_C collocation points

$$C = \{(\mathbf{x}_i) \mid_{i=1,\cdots,n_C} \subset \Omega\}, \tag{1.4}$$

situated within the domain and as well n_B points on the boundary

$$B = \{(\mathbf{x}_j) \mid_{j=1,\cdots,n_B} \subset \partial\Omega\}. \tag{1.5}$$

Finally the original problem is transformed into a *Free Constrained Optimization Problem* defining a cost function as follows

$$F(\mathbf{y}) = \sum_{i=1}^{n_C} \|\mathbf{L}\mathbf{y}(\mathbf{x}_i) - \mathbf{f}(\mathbf{x}_i)\|^2 + \sum_{j=1}^{n_B} \|\mathbf{B}\mathbf{y}(\mathbf{x}_j) - \mathbf{g}(\mathbf{x}_j)\|^2. \tag{1.6}$$

Therefore, the original problem of solving the DE given by Eq. (1.1) is transformed into a new problem consisting in seeking a function $\mathbf{y}(\mathbf{x})$ which minimizes the cost function given by Eq. (1.6).

In the context of the problem presented above, the main goals treated in this thesis are:

1. **Stablish the more efficient way of expressing the solution function $\mathbf{y}(\mathbf{x})$.**
2. **Create, adapt or select efficient evolutionary methods to search the function $\mathbf{y}(\mathbf{x})$.**
3. **Create a generic framework to solve different types of deterministic DEs:**

linear and non linear, ordinary and partial equations, and systems of DEs.

Regarding the first goal, several alternatives can be studied. Thus, it is possible to build the solution using functions basis, parametric kernels or generic algebraic expressions.

To solve the optimization problem of seeking the best function $\mathbf{y}(\mathbf{x})$, heuristic methods based in population of candidate solution, as Evolutionary Algorithms, will be used. This approach reduces the probability of being trapped in local minima. In this context, it will be important to determine if the algorithms need to solve non-separable problems. These type of problems are very challenging because they can not be split into smaller ones, so the original problem must be solved simultaneously in all the coordinates.

As a result of these work, a *generic* tool to solve DEs should be obtained. Generic not only means that the same algorithm can managed a wide variety of DEs, but as well it should be robust, in the sense that good results should be obtained without investing a lot of effort in adjusting the control parameters. Another important aspect to highlight is that, although evolutionary algorithms are stochastic, low dispersion should be observed in the results when the algorithm runs several times solving the same DE. An friendly interface is desirable in order to facilitate the use by non-expert users in high advanced mathematics.

Finally, to finish this section, a set of *research question* will be enumerated. These statements suggest some open question or hypothesis, which they will be treated along the thesis. In the conclusion and based on the research performed in the thesis, they will be answered.

1. **Is it more efficient to express candidate solutions using function basis, parametric kernels or generic mathematical expressions?**

2. **Does the heuristic algorithms need to deal with non-separable problems in the context of solving DEs?**

3. **Although numerical methods are by far the most efficient approach to solve DEs, could heuristics outperform them in some aspects? If any, which ones?**

4. **Because derivatives must be obtained from each solution proposed by the evolutionary algorithm, it is possible to implement efficient solvers without**

using complex symbolic engine libraries?

5. **Although evolutionary algorithms are stochastic methods, it is possible to guarantee a good convergence to the solution?**

1.4 Research Methodology

To solve the proposed problem, several EC paradigms can be employed. A first option could be Genetic Programming (GP), or some modern variations as Grammatical Evolution (GE). These two options build candidate solutions by means of algebraic expressions based on a set of functions or terminals defined a priori. On the other hand, the original problem could be transformed into an optimization one where several real unknown values must be sought. In this context, other paradigms such as Genetic Algorithms (GA) or Evolution Strategies (ES) could be used.

Evolutionary Algorithms often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape and can cope with high dimensionality spaces making good exploration of the solution space. Furthermore, Evolutionary Algorithms can be easily combined with other methods to perform local search to improve the exploitation of the best solutions.

If the DE is solved as a real optimization problem, in order to simplify the search process and to ensure that the solution can be expressed accurately by the algorithm, a basis function could be employed. Then, the problem will turn into seeking the best lineal combination of the basis function, so the domain space size is considerably reduced compared with the symbolic regression approach. Several basis functions could be used. In our case, trigonometric functions (Fourier decomposition) and Gaussian kernels have been employed.

Following the complete action schedule is given:

1. Bibliography study of the current state of the art (EC techniques to solve DEs): 4 months.

2. Analysis and election of a symbolic math engine library: 1 month.

3. Definition of a problem test suite extracted from the literature to check the performances of the solvers: 4 months.
4. Implementation, analysis and test of a DE solver based on GE (DESGE): 6 months.
5. Implementation, analysis and test of a DE solver based on ES and Fourier series (DE-SES): 7 months.
6. Publish DESGE algorithm (generate a paper and journal submission): 1 month
7. Implementation, analysis and test of a DE solver based on ES and Gaussian kernels (DESCMA-ES): 7 months.
8. Publish DESCMA-ES algorithm (generate a paper and journal submission): 2 month
9. Write the Ph.D. report: 4 months.

1.5 Structure of the Thesis

This thesis consists of 7 chapters. Below we provide a brief overview summarizing the contents of each of these chapters:

- Chapter 1 **"Introduction"**: We present the motivation and scope of this thesis. The problem statement is mathematically formalized. Finally, the research methodology is provided.
- Chapter 2 **"State of the Art"**: First the concept of derivative and differentiation is provided. Then, a general introduction of differential equations and their importance in fields such as engineering, biology or economics is commented. A taxonomy of the existing methods to solve differential equations is provided. Here a survey of existing works for solving differential equations with heuristic methods is commented. The study is carried out according to the algorithms used (neural networks, genetic programming, swarm optimization, etc) and how the candidate solutions are expressed (Fourier series, general mathematical expressions, linear combination of kernels, etc).

- Chapter 3 **"Background"**: In this chapter all the evolutionary algorithms used in the present thesis for solving differential equations are described. Concretely, Grammatical Evolution (GE), Evolution Strategies (ES) and Covariance Matrix Adaptation Evolution Strategies (CMA-ES). Additionally, one heuristic optimization method used as a local search is presented: Downhill Simplex method.

- Chapter 4 **"Novel Methods for solving DEs"**: This chapter is the main contribution of the present work. Here, several methods to solve differential equations are described, all of them using evolutionary paradigms: GE, ES and CMA-ES.

- Chapter 5 **"Results and Discussion"**: In this chapter first an extensive set of DE problems extracted from the literature are provided, including the domain range, boundary conditions and exact solution (if it exists). Then, the proposed algorithms are applied on these problems to measure the performance.

- Chapter 6 **"Future Research"**: An outlook on future directions of the work is provided. The best algorithm presented in the thesis is applied with some modifications (changing the kernel basis functions) on some real problems involving stiff equations.

- Chapter 7 **"Conclusions"**: We discuss and summarize the main conclusions and contributions of the work.

Additionally, the thesis contains the following appendices at the end, with complementary information:

- Appendix A **"Release Control Version"**: Release notes of the implementation programs developed in C++ to test all the algorithms described in the present work.

- Appendix B **"A C++ Program to Test GiNaC Library"**: Several checks are performed to validate a symbolic mathematical engine.

- Appendix C **"Configuration File Examples"**: Two examples of the configuration file for solving one ordinary differential equation and one partial differential equation are listed.

- Appendix D **"Conclusiones (Conclusions in Spanish)"**: We present the conclusions of this thesis in Spanish language.

- Appendix E **"Publications"**: We list our main publications related with parts of the work presented in this thesis.

Chapter 2

State of the Art

In this chapter a state of the art of the existing methods to solve differential equations (DEs) will be exposed. First, in Section 2.1, a brief review of *derivative* concept is provided and a definition of DEs will be given. It will be highlighted the importance of these entities in almost all the scientific fields. DEs can provide not only qualitative knowledge about a certain phenomenon, but quantitative information can be obtained if we can solve them. In Section 2.2 a possible taxonomy of the most popular methods to solve DEs is briefly described. Following, in Section 2.3 we pay attention to heuristic methods for solving DEs giving a survey of the most important works reported in the literature. Finally, Section 2.4 gives some conclusions about the state of the art in the field of solving DEs with heuristic methods.

2.1 Introduction

The Derivative

Calculus is the mathematical study of the *rate of change*. It has two major branches, *differential* calculus (concerning rates of change and slopes of curves), and *integral* calculus (concerning accumulation of quantities and the areas under and between curves). These

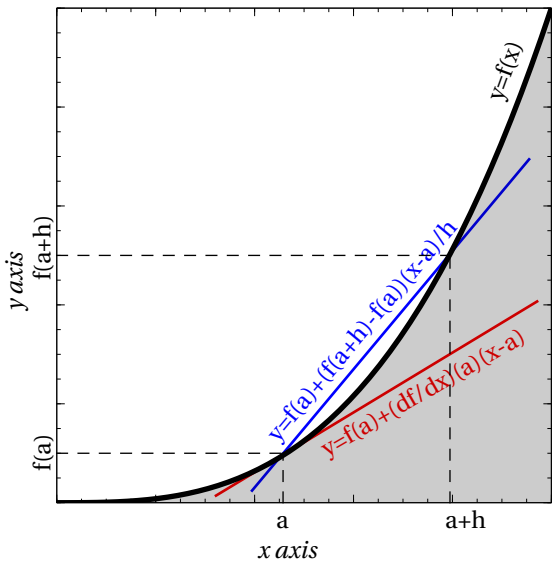

Figure 2.1: Geometrical interpretation of derivative of function $f(x)$ at point a as the limit of secant line when h goes to 0.

two branches are related to each other by the fundamental theorem of Calculus [6]. The modern development of calculus is usually credited to Isaac Newton (1643-1727) and Gottfried Leibniz (1646-1716), who provided independent and unified approaches to differentiation and derivatives.

The process of finding a derivative is called *differentiation*. The reverse process is called *antidifferentiation*. The fundamental theorem of Calculus states that antidifferentiation is the same as integration. Differentiation and integration constitute the two fundamental operations in single-variable calculus.

Differentiation is the action of computing a derivative. The derivative of a function $f(x)$ of a variable x is a measure of the rate at which the value of the function changes with respect to the change of the variable. It is called the derivative of f with respect to x. The geometrical interpretation of the derivative is given in Fig. 2.1. If x and y are real numbers, and if the graph of f is plotted against x, the derivative is the slope of this graph at each point. The mathematical rigorous definition of derivative of function f at point a is expressed using the concept of limit:

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}. \tag{2.1}$$

	First Derivative	Second Derivative
Lagrange (prime)	f'	f''
Newton (dot)	$\dot{f}$	$\ddot{f}$
Euler	Df	$D^2 f$
Leibniz	df/dx	$d^2 f/dx^2$
Subscript	f_x	f_{xx}

Table 2.1: Common derivative notations.

When the limit exists, f is said to be *differentiable* at a. Here $f'(a)$ is the Lagrange's notation to express the derivative. Other notations are also common in the literature, as Table 2.1 shows.

Let f be a function that has a derivative at every point a in the definition domain of f. Because every point has a derivative, there is a function that maps the point a to the derivative of f at a. This function is written f' and is called the derivative function or the derivative of f. Sometimes f has a derivative at the majority of points of its domain, but not all of them. The function whose value at a maps $f'(a)$ whenever $f'(a)$ is defined and elsewhere is undefined is also called the derivative of f. It is still a function, but its domain is strictly smaller than the domain of f.

Let f be a differentiable function, and let $f'(x)$ be its derivative. The derivative of $f'(x)$ (if it has one) is written $f''(x)$ and is called the second derivative of f. Similarly, the derivative of a second derivative, if it exists, is written $f'''(x)$ and is called the third derivative of f. Continuing this process, one can define, if it exists, the nth derivative as the derivative of the $(n-1)$th derivative. These repeated derivatives are called higher-order derivatives. The nth derivative is also called the derivative of order n.

If function f depends on several variables, instead of derivatives, partial derivatives are used. In general, the partial derivative of a function $f(x_1, \cdots, x_n)$ in the direction x_i at the

point $(a_1, \cdots, a_n)$ is defined as:

$$\frac{\partial f}{\partial x_i}(a_1, \cdots, a_n) = \lim_{h \to 0} \frac{f(a_1, \cdots, a_i + h, \cdots, a_n) - f(a_1, \cdots, a_n)}{h}. \tag{2.2}$$

Differential Equations (DEs)

Once the concept of derivative has been exposed in the previous section, here DEs will be introduced.

A DE is just an algebraic relation between functions and their derivatives. Many fundamental laws of Physics and Chemistry can be formulated as DEs. And they are useful to model different problems in a lot of scientific fields, such as Biology, Economics or Engineering. Moreover, when the same DE describes different phenomena, it has been used as unifying principle. As an example, the propagation of sound in the atmosphere, or the waves on the surface of a pond may be described by the same second-order partial differential equation, the *Wave* equation. In the same way, heat conduction in a solid is governed by another second-order partial differential equation, the *Heat* equation. The *Black-Scholes* equation in finance is, for instance, related to the Heat equation.

DEs are mathematically studied from several different perspectives, mostly concerned with their solution, i. e., the set of functions that satisfy the equation. Only the simplest differential equations are solvable by explicit formulas. However, some properties of solutions of a given DE may be determined without finding their exact form. If a closed-form expression for the solution is not available, the solution may be *numerically approximated*. The theory of dynamical systems puts emphasis on qualitative analysis of systems described by differential equations, while many numerical methods have been developed to determine solutions with a given degree of accuracy.

The study of DEs is a vast field in pure and applied Mathematics, Physics, and Engineering. All of these disciplines are concerned with the properties of DEs of various types. Pure Mathematics focuses on the existence and uniqueness of solutions, while applied Mathematics emphasizes the rigorous justification of the methods for approximating solutions. The study

of the stability of solutions of differential equations is known as *stability theory*.

Following, a taxonomy of DEs will be described. A first classification can be done according to the number of independent variables involved. Thus, an *ordinary differential equation* (ODE) is a differential equation in which the unknown function (also called dependent variable) is a function of a single independent variable. Ordinary DEs are further classified according to the order of the highest derivative of the dependent variable with respect to the independent variable appearing in the equation. The most important cases for applications are first-order and second-order differential equations.

On the other hand, when the unknown function is a function of multiple independent variables and the equation involves its partial derivatives, the DE is called *partial differential equation* (PDE). The order is defined similarly to the case of ordinary differential equations, but further classification into *elliptic*, *hyperbolic*, and *parabolic* equations, especially for second-order linear equations. Some PDEs do not fall into any of these categories over the whole domain of the independent variables and they are said to be of *mixed* type.

Both ordinary and partial DEs are broadly classified as *linear* and *nonlinear*. To determine if a DE is linear or not, we express the equation as a function F of the independent variables, dependant variables and their derivatives. To fix ideas, in the case of the following DE

$$y'' + x \cdot y' + y = \sin(x),$$

we can express the original DE by means of the function F in this way:

$$F(x, y, y', y'') = 0.$$

In this case, the function will be

$$F(x_1, y_1, y_2, y_3) = y_3 + x_1 \cdot y_2 + y_1 - \sin(x_1).$$

Note that x_1 denote the independent variable, meanwhile y_i are the dependent variables and their derivatives. Then, the original DE will be linear if and only if the function F

is linear respect to y_i variables. The characteristic property of linear equations is that their solutions form an affine subspace of an appropriate function space, which results in much more developed theory of linear DEs. Homogeneous linear DEs are a further subclass for which the space of solutions is a linear subspace, i.e., the sum of any set of solutions or multiples of solutions is also a solution. Nonlinear DEs can exhibit very complicated behavior over extended time intervals, characteristic of chaos. Even the fundamental questions of existence, uniqueness, and extendibility of solutions for nonlinear DEs, and well-posedness of initial and boundary value problems for nonlinear PDEs are hard problems and their resolution in special cases is considered to be a significant advance in the mathematical theory.

When several dependent variables are involved simultaneously, a *system of differential equation* must be solved. Depending of the derivatives, the system can be of ordinary or of partial differential equations.

Other equations related with DEs are delay differential equations (DDEs), stochastic differential equations (SDEs), differential algebraic equations (DAEs) and fractional differential equations (FDEs). A DDE is an equation for a function of a single variable, usually called time, in which the derivative of the function at a certain time is given in terms of the values of the function at earlier times. A simple example could be $x'(t) = -x(t-1)$. A SDE is an equation in which the unknown quantity is a stochastic process and the equation involves some known stochastic processes. A DAE is a DE comprising differential and algebraic terms, given in implicit form. FDEs (also known as extraordinary differential equations) are a generalization of differential equations through the application of *fractional calculus*. Fractional calculus studies the possibility of taking real or complex number powers of the differentiation operator. Thus, extending nth derivative of a monomial x^k

$$\frac{d^n}{dx^n}x^k = \frac{k!}{(k-n)!}x^{k-n}$$

to not integral number v, we obtained the fractional derivative using the gamma function:

$$\frac{d^v}{dx^v}x^k = \frac{\Gamma(v+1)}{\Gamma(k-v+1)}x^{k-v}.$$

In general, the fractional derivative of function $f(t)$ of order v is given by

$$\frac{d^v}{dx^v} f(t) = \frac{1}{\Gamma(v)} \int_0^t (t-\xi)^{-v-1} f(\xi)\, d\xi,$$

for $t > 0$ and $\nu \in \mathbb{R}^+$.

Examples of DEs

To conclude this section, some famous DEs are enumerated in Tables 2.2, 2.3 and 2.4. For the sake of conciseness, only some equations will be briefly commented. Equations are sorted in three groups according if they are related to Physics and Engineering, Biology or Economics. Inside each group, the equations are presented in chronological order.

The Newton's *second law* states that the net force on an object is equal to the derivative of its linear momentum in an inertial reference frame. The second law can also be stated in terms of an object's acceleration. Since Newton's second law is only valid for constant-mass systems, mass can be taken outside the differentiation operator by the constant factor rule in differentiation. The three laws of motion were first compiled by Isaac Newton in his *Philosophiæ Naturalis Principia Mathematica*, first published in 1687.

The *Wave* equation is an important second-order linear partial differential equation for the description of waves such as sound, light and water waves. It arises in fields like acoustics, electromagnetics, and fluid dynamics. Historically, the problem of a vibrating string such as that of a musical instrument was studied by Jean le Rond d'Alembert, Leonhard Euler, Daniel Bernoulli, and Joseph-Louis Lagrange. In 1746, d'Alembert discovered the one-dimensional Wave equation, and within ten years Euler discovered the three-dimensional Wave equation.

The *Heat* equation is a parabolic partial differential equation that describes the distribution of heat (or variation in temperature) in a given region over time. The Heat equation is a consequence of Fourier's law of conduction. Solutions of the Heat equation are characterized by a gradual smoothing of the initial temperature distribution by the flow of heat from warmer to colder areas of an object. Generally, many different states and starting conditions

Equation Name	Date	Expressions
Newton's second law	1687	$\mathbf{F} = \frac{d(m\mathbf{v})}{dt}$
Wave	1746	$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$
Euler-Lagrange	1750	$L_x\left(t, q\left(t\right), q'\left(t\right)\right) - \frac{d}{dt} L_v\left(t, q\left(t\right), q'\left(t\right)\right) = 0$
Cauchy-Riemann	1752	$\begin{cases} \frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x} \end{cases}$
Laplace	1783	$\nabla^2 \varphi = 0$
Heat	1822	$\frac{\partial u}{\partial t} = \alpha \nabla^2 u$
Navier-Stokes	1822	$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}$
Poisson	1813	$\nabla^2 \varphi = f$
Geodesic	1854	$\frac{d^2 x^\lambda}{dt^2} + \Gamma^\lambda_{\mu\nu} \frac{dx^\mu}{dt} \frac{dx^\nu}{dt} = 0$
Hamilton	1855	$\begin{cases} \frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}} \\ \frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}} \end{cases}$
Diffusion	1855	$\frac{\partial \phi(\mathbf{r},t)}{\partial t} = \nabla \cdot \left[D\left(\phi, \mathbf{r}\right) \nabla \phi\left(\mathbf{r}, t\right)\right]$
Maxwell	1861	$\begin{cases} \nabla \cdot \mathbf{E} = \rho/\varepsilon_0 \\ \nabla \cdot \mathbf{B} = 0 \\ \nabla \times \mathbf{E} = -\partial \mathbf{B}/\partial t \\ \nabla \times \mathbf{B} = \mu_0\left(\mathbf{J} + \varepsilon_0 \partial \mathbf{E}/\partial t\right) \end{cases}$
Radioactive decay	1896	$-\frac{dN}{N} = \lambda dt$
Einstein field	1915	$R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R + g_{\mu\nu} \Lambda = \frac{8\pi G}{c^4} T_{\mu\nu}$
Schrödinger	1925	$i\hbar \frac{\partial \Psi}{\partial t} = \hat{H} \Psi$
Lorenz	1963	$\begin{cases} \frac{dx}{dt} = \sigma\left(y - x\right) \\ \frac{dy}{dt} = x\left(\rho - z\right) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases}$

Table 2.2: Some famous DEs in Physics and Engineering sorted in chronological order.

will tend toward the same stable equilibrium. As a consequence, to reverse the solution and conclude something about earlier times or initial conditions from the present heat distribution is very inaccurate except for short time periods.

Navier-Stokes equations, named after Claude-Louis Navier and George Gabriel Stokes, describe the motion of fluid substances. These equations arise from applying Newton's second law to fluid motion, together with the assumption that the stress in the fluid is the sum of a diffusing viscous term (proportional to the gradient of velocity) and a pressure term.

Maxwell's equations are a set of partial differential equations that, together with the Lorentz force law, form the foundation of classical electrodynamics, classical optics, and electric circuits. These fields in turn underlie modern electrical and communications technologies. Maxwell's equations describe how electric and magnetic fields are generated and altered by each other and by charges and currents. They are named after the Scottish physicist and mathematician James Clerk Maxwell, who published an early form of those equations between 1861 and 1862.

In quantum mechanics, the *Schrödinger* equation is a partial differential equation that describes how the quantum state of a physical system changes with time. It was formulated in late 1925, and published in 1926, by the Austrian physicist Erwin Schrödinger.

The *Lorenz* system is a system of ordinary differential equations first studied by Edward Lorenz. It is notable for having chaotic solutions for certain parameter values and initial conditions. In particular, the Lorenz attractor is a set of chaotic solutions of the Lorenz system which, when plotted, resemble a butterfly or figure eight. Edward Lorenz developed this equation when he was studying a simplified mathematical model for atmospheric convection. Besides, it also arise in simplified models for lasers, dynamos, thermosyphons, brushless DC motors, electric circuits, chemical reactions and forward osmosis

A *logistic* function or logistic curve is a common "S" shape (sigmoid curve), with equation $f(x) = 1/(1 + e^{-x})$. The function was named in 1844-1845 by Pierre François Verhulst, who studied it in relation to population growth. The initial stage of growth is approximately exponential. Then, as saturation begins, the growth slows, and at maturity, growth stops.

Equation Name	Date	Expressions
Verhust	1884	$\frac{d}{dx}f(x) = f(x) \cdot (1 - f(x))$
Lotka-Volterra	1910	$\begin{cases} \frac{dx}{dt} = x(\alpha - \beta y) \\ \frac{dy}{dt} = -y(\gamma - \delta x) \end{cases}$
Von Bertalanffy model	1934	$L'(t) = r_B(L_\infty - L(t))$
Hodgkin-Huxley model	1952	$\begin{cases} I = C_m \frac{dV_m}{dt} + g_k(V_m - V_k) + \\ g_{Na}(V_m - V_{Na}) + g_l(V_m - V_l) \end{cases}$
Replicator equation	1973	$\dot{x}_i = x_i \left[f_i(x) - \sum_{j=1}^{n} x_j f_j(x) \right]$

Table 2.3: Some famous DEs in Biology sorted in chronological order.

Equation Name	Date	Expressions
Malthusian growth model	1798	$\frac{dx}{dt} = kx$
Solow-Swan	1956	$\dot{k}(t) = sk(t)^\alpha - (n + g + \delta)k(t)$
Black-Scholes	1973	$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$
Shethi model	1983	$dX_t = \left(rU_t\sqrt{1 - X_t} - \delta X_t\right)dt + \sigma(X_t)\,dz_t$

Table 2.4: Some famous DEs in Economics sorted in chronological order.

A *Malthusian growth* model, sometimes called a simple exponential growth model, is essentially exponential growth based on a constant rate. This model is widely regarded in the field of population ecology as the first principle of population dynamics.

The *Black-Scholes* model is a mathematical model of a financial market containing certain derivative investment instruments. From the model, one can deduce the Black-Scholes formula, which gives a theoretical estimate of the price of European-style options. The formula led to a boom in options trading and legitimized scientifically the activities of the Chicago Board Options Exchange and other options markets around the world. It is widely used, although often with adjustments and corrections, by options market participants.

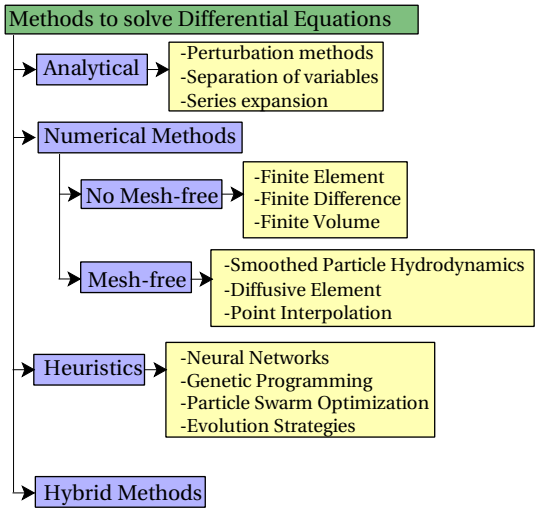

Figure 2.2: A possible taxonomy of methods to solve differential equations.

2.2 Taxonomy of Methods for solving DEs

Some simple differential equations admit solutions given by explicit formulas. But in the general case, only approximated solutions can be found. Several paradigms exist in the literature to solve the equations. Fig. 2.2 shows a possible taxonomy of existing methods to solve differential equations. A first group can be formed with analytical methods, which try to find exact solutions. One of these method is *separation of variables*, also known as *Fourier* method. Basically, algebra allows one to rewrite the equation so that each of two variables occurs on a different side of the equation. For example, being $u \equiv u(x,t)$ we consider the Heat equation

$$u_t = \alpha u_{xx}$$

with homogeneous boundary condition $u(0,t) = u(L,t) = 0$. Note that subscript notation described in Table 2.1 is employed. Now, if we assume that a possible solution is in the form of $u(x,t) = X(x)T(t)$ and substituting u back into equation and using the product rule, we obtain

$$\frac{T'(t)}{\alpha T(t)} = \frac{X''(x)}{X(x)}.$$

Since the right hand side depends only on x and the left hand side only on t, both sides are equal to some constant value $-\lambda$. Making some algebra, we finally obtain the general solution of the equation in the form of

$$u(x,t) = \sum_{n=1}^{\infty} D_n \sin\frac{n\pi x}{L} \exp\left(-\frac{n^2\pi^2\alpha t}{L^2}\right),$$

where D_n coefficients are determined by the initial condition $u(x,0) = f(x)$. Thus,

$$f(x) = \sum_{n=1}^{\infty} D_n \sin\frac{n\pi x}{L}.$$

However, as it has been commented, very few problems admit an analytical solution. The second group of methods transforms the original differential equations into a system of algebraic equations and solve them by numerical methods. Inside this category, two subfamilies can be distinguished according to whether or not a connectivity or mesh is needed. Mesh schemes can be divided in *finite element* method (FEM) [7], *finite difference* method (FDM) [8], or *finite volume* method (FVM) [9]. For the sake of clarity, let see an example of finite difference method to solve the Heat equation. For that, we approximate all the derivatives by finite differences. We discretize the space domain using a mesh $x_0, \cdots, x_J$ and the time by means of several instants $t_0, \cdots, t_N$. We assume a uniform partition both in space and in time, so the difference between two consecutive space points will be h and between two consecutive time points will be k. The points $u(x_j, t_n) = u_j^n$ will represent the numerical approximation of $u(x_j, t_n)$ with $j = 0, \cdots, J$ and $n = 0, \cdots, N$. Using a forward difference at time t_n and a second-order central difference for the space derivative at position x_j we get the recurrence equation:

$$\frac{u_j^{n+1} - u_j^n}{k} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2}$$

This is an explicit method for solving the one-dimensional heat equation. We can obtain u_j^{n+1} from the other values in this way:

$$u_j^{n+1} = (1-2r)\, u_j^n + r u_{j-1}^n + r u_{j+1}^n$$

where $r = k/h^2$. So, with this recurrence relation, and knowing the values at time n, one can obtain the corresponding values at time $n + 1$. Note that u_0^n and u_j^n must be replaced by the boundary conditions, where in this example, they are both 0. This explicit method is known to be numerically stable and convergent whenever $r \leq 1/2$. Other schemes have different behavior, such as *implicit* methods or *Crank-Nicolson* method.

On the other hand, mesh-free algorithms, such as *smoothed particle hydrodynamics* (SPH), *diffusive element* method (DEM) and *point interpolation* method (PIM), do not require a mesh connectivity [10]. However, the final algebraic equations obtained are solved using numerical methods. For the sake of completeness, following a brief introduction to SPH is given. This method was applied traditionally to solve fluids, although it is possible to be applied to solids. The method works by dividing the fluid into a set of discrete elements, referred to as particles. These particles have a spatial distance (known as the "smoothing length", typically represented in equations by h), over which their properties are "smoothed" by a kernel function. This means that the physical quantity of any particle can be obtained by summing the relevant properties of all the particles which lie within the range of the kernel. The contributions of each particle to a property are weighted according to their distance from the particle of interest, and their density. Mathematically, this is governed by the kernel function. Kernel functions commonly used include the *Gaussian* function and the *cubic spline*. The latter function is exactly zero for particles further away than two smoothing lengths.

A radical different approach (gather in *Heuristic* group at Fig. 2.2) for solving differential equations consists on transforming the problem into an optimization one where a candidate solution is tested according to a fitness or cost function which measures how the differential equations are fulfilled. Generally speaking, these methods are as well mesh-free, but are more flexible because can cope with different type of equations. Several paradigms have been reported in this field, such as cellular automata [11], artificial neural networks [12, 13, 14, 15, 16, 17, 2, 18, 19, 20, 21, 22], genetic algorithms (GAs) [23], genetic programming (GP) [24, 25, 26, 3, 27, 28], particle swarm optimization (PSO) [29, 4], differential evolution [5] or support vector machines (SVM) [30]. A survey about solving differential equations with

a special type of neural networks (multilayer perceptron) and radial basis functions can be consulted in [19].

Finally, a fourth approach can be adopted, where numerical and heuristics methods are combined [31, 32, 33]. Thus, He et al. [31] propose three hybrid algorithms for solving linear and partial differential equations. The hybrid algorithms integrate the classical SOR method with evolutionary computation techniques. The recombination operator in the hybrid algorithms mixes two parents by a kind of averaging, which is similar to the intermediate recombination often used in evolution strategies. The mutation operator is equivalent to one iteration in the SOR method. However, the mutation is stochastic as a result of stochastic self-adaptation of the relaxation parameter .

2.3 Heuristic Methods for DEs

In this section a review of the existing works which try to solve differential equations with "non standard" algorithms is presented. By non standard methods we mean those algorithms not inspired in numerical methods and therefore, less extended in the literature. Other authors call these type of algorithms *heuristic* methods. Although normally they are less efficient than numerical methods, heuristics have some advantages regarding set-up of the problem, storing requirements, interpolation properties, etc. Several classifications could be done in this area according to the optimization algorithm used or how the candidate solutions are expressed. Table 2.6 presents a summary of the present survey. It shows the year, the main equations solved, how the candidate solutions are built or expressed, the optimization algorithm and whether if a local search is used or not.

2.3.1 Cellular Automata (CA)

A cellular automaton is a discrete model which consist of a regular grid of cells, each in one of a finite number of states. The grid can be in any finite number of dimensions. For each cell, a set of cells called its neighborhood is defined relative to the specified cell. An initial state is selected by assigning a state for each cell. A new generation is created, according to

Year	Paper	Equation	Solution expression	Optimization Method	Local Search
1995	[11]	Burger and Korteweg-de-Vries	CNN	Downhill Simplex	No
1998	[12]	Ordinary and Partial DEs	ANN	BFGS	No
2000	[13]	First order PDEs for a controller	ANN	Backpropagation	No
2001	[24]	1D Convection-Diffusion	Polynomials	GP	No
2001	[34]	1D Unsteady Groundwater flow	ANN	Evolutionary Algorithm	No
2003	[14]	Catalytic solid-gas reactor	ANN	GA	Gradient descent
2003	[15]	Poisson	ANN	Backpropagation	No
2005	[25]	Ordinary DEs	Gram-Schmidt basis functions	GP	No
2005	[32]	Electromagnetic problems	FENN	Gradient descent	No
2006	[26]	Second order, homogeneous	Symbolic expressions	Graph-based GP	No
2006	[1]	Ordinary and Partial DEs	Symbolic expressions	GE	No
2008	[3]	Elliptic PDEs	Symbolic expressions	GP	Gradient Boosting
2008	[35]	Schrodinger	Analytical basis functions	GA	No
2009	[2]	Ordinary and Partial DEs	ANN	GE	BFGS
2009	[29]	First order DEs	ANN	PSO	No
2009	[27]	Ricatti	Symbolic expressions	GE	No
2009	[17]	Schrödinger	ANN	Backpropagation	No
2010	[36]	LDE, constant coefficients	ANFIS	Backpropagation	No
2010	[28]	Simple linear ODEs	Symbolic expressions	CGP	No
2010	[37]	Fractional DEs	ANN	GA	Active set algorithm
2010	[38]	Schrodinger	Exponential functions	GA	No
2011	[39]	Fractional Ricatti	ANN	PSO	SA
2011	[18]	Allen-Cahn	ANN	Residual subsampling	No
2011	[33]	Navier-Stokes	ANN	GA	No
2012	[23]	Schrödinger	Parameterized basis functions	GA	No
2013	[4]	ODEs	Fourier series	PSO	No
2013	[21]	Systems of Fuzzy DEs	FNN	Backpropagation	No
2013	[40]	Electronic circuits	ANN	GA	No
2014	[5]	Poisson	Polynomials	DE	No
2014	[41]	Wind Speed time series	ANN	SA, GSD	No
2014	[42]	Pantograph functional DE	ANN	SA, PS, GA	ASA
2015	[30]	Second order PDEs	LS-SVM	-	No

Table 2.6: Survey of works about solving DEs with heuristic methods.

some fixed rule that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. The concept was originally discovered in the 1940s by Stanislaw Ulam and John von Neumann while they were contemporaries at Los Alamos National Laboratory. It was not until the 1970s and Conway's Game of Life, a two-dimensional cellular automaton, that interest in the subject expanded beyond academia. Since then, a lot of works have used CA approach. As an example, Monteagudo and Santos [43, 44] present a CA model for cancer growth where emergent dynamics of tumor growth at cellular level is studied.

Although the evolution of CA is governed by certain updating rules rather than differential equations, exists a close relationship between both concepts. That is, given differential equations, we can construct a rule-based cellular automaton, or vice versa. As an example, considered the one dimensional heat equation:

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2},$$

where T is temperature and κ is the thermal diffusivity. The simplest discretization of the above heat equation is the central difference for spatial derivative and forward scheme for time derivatives:

$$T_i^{n+1} - T_i^n = \frac{\kappa \Delta t}{(\Delta x)^2} \left(T_{i+1}^n - 2T_i^n + T_{i-1}^n \right),$$

where i and n are the spatial and time indices. If we choose the time steps and spatial discretization such that $\kappa \Delta t / (\Delta x)^2 = 1$, we finally obtain

$$T_i^{n+1} - T_i^n = T_{i+1}^n - 2T_i^n + T_{i-1}^n,$$

which actually is a cellular automata.

As we see with the previous example, it is relatively straightforward to derive the updating rules for cellular automata from the corresponding partial differential equations. However, the reverse is usually very difficult. There is no general method available to formulate the continuum model or differential equations for given rule-based cellular.

Several works in the literature can be found that use CA to model different dynamic systems. Thus Puffer et al. [11] present a mixed approach between CA and Artificial Neural Network (ANN) to solve differential equations. This work will be described in more detail in subsection 2.3.6 devoted to ANN.

2.3.2 Differential Evolution

Differential Evolution (DEv) is originally due to Storn and Price in 1997 [45]. DEv is a very simple population based, stochastic function minimizer which is very powerful at the same time. DEv managed to finish 3rd at the First International Contest on Evolutionary Computation (1stICEO) which was held in Nagoya in 1996. DE turned out to be the best genetic type of algorithm for solving the real-valued test function suite of the 1st ICEO (the first two places were given to non-GA type algorithms which are not universally applicable but solved the test-problems faster than DEv). The crucial idea behind DEv is a scheme for generating trial parameter vectors. Basically, DEv adds the weighted difference between two population vectors to a third vector.

An interesting approach for solving differential equations using DEv can be consulted in a work by Panagant and Bureerat [5] where the two dimensional Poisson equation with different source terms is solved. Partial differential equation solutions are approximated by a polynomial expansion

$$\Psi(x,y) = a_1 + \sum_{i=1}^{n} a_{i+1}x^i + \sum_{i=1}^{n} a_{i+n+1}y^i + \sum_{i=1}^{n}\sum_{j=1}^{n} a_{(i-1)n+j+2n+1}x^i y^j,$$

where a_i are polynomial coefficients to be determined and n is a maximum of the polynomial order for x and y. The number of terms a_i is $(n+1)^2$, which means that there are the same design variables for an optimization problem. A minimization problem is defined, where the design variables a_i are seek in order to minimize the functional $\int_\Omega |R(x,y)|\, dA$, where R is a residual function, subject to the constrains defined by the boundary conditions. The constrained optimization problem is transform into an unconstrained one adding the

boundary conditions term to the functional defined with the residuals. The optimization problem is solved using Differential Evolution.

2.3.3 Genetic Algorithm

The father of the original Genetic Algorithm (GA) was John Holland who invented it in the early 1970's. Genetic Algorithms are the most popular of all population-based methods in Evolutionary Computing. A population of candidate solutions (called individuals or phenotypes) to an optimization problem is evolved toward the best solutions. Each candidate solution has a set of chromosomes which can be mutated and recombined with other individuals. Traditionally, solutions are binary coded as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process. The population in each iteration is called a generation. In each generation, the fitness of every individual in the population is evaluated, which is usually the value of the objective function for that individual in the optimization problem being solved. The fittest individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached.

The use of GA for solving differential equations is not common, although some examples can be consulted in the literature. Thus, in a set of articles from MacNeil, [35, 38, 23] the solution of the Schrödinger equation is presented. Specifically speaking, the equation to solve is the two-center Coulomb electrostatic problem:

$$H\Psi = E\Psi,$$

where H is the Hamiltonian operator, E is an energy eigenvalue, and Ψ is the Schrödinger wave function. The wave function Ψ must not only solve the equation, but is subject to

several other constraints. The equation error is defined then as $error = (H - E)\,\Psi$. The fitness function for the GA is defined using the square error of the function:

$$Fitness = \frac{1}{\iiint |error|^2 \, dxdydz}$$

A parameterization of the wave function was selected ($\Psi = c \exp(-f \cdot r)$), so the number of unknowns is considerably reduced. MacNeil presents in his last paper [23] a more complex problem, a multi-atom molecule of H_2^+. In this problem, the GA always find anomalous solutions, called by the authors "parasitic solutions". To deal with these solutions, the author propose three alternatives: to change the basis functions, to change the fitness functions, or to reformulate the main equation.

2.3.4 Genetic Programming

The first statement of modern "tree-based" genetic programming was given by Nichael L. Cramer (1985). This work was later greatly expanded by John R. Koza, a main proponent of GP who has pioneered the application of genetic programming in various complex optimization and search problems.

GP is another evolutionary paradigm inspired by biological evolution to find computer programs that perform a user-defined task. Essentially GP is a set of instructions and a fitness function to measure how well a computer has performed a task. GP uses parse trees as chromosome, creating expressions in a given formal syntax.

Several works in the literature solves ED using GP. Thus, Howard and Roberts [24] propose solving the convective-diffusion equation using polynomials. The equation is

$$T_{xx} - P_e T_x = 0$$

where the unknown function T is the temperature and P_e is the Peclet number. The boundary conditions are $T(0) = 1$ and $T(1) = 0$. The original equation is transformed in order to fulfill by construction the boundary conditions. The fitness function is computed with the

least square errors of the residuals. An analytical expression for the fitness function can be obtained because the solutions are expressed by means of polynomials.

Another interesting work can be consulted in Kirstukas et al. [25], where a GP approach from an engineering perspective is employed. This paper presents a novel technique for performing the fitness evaluation by analytically evaluating derivatives of the candidate solution functions. The analytical solution approach provides a significant advantage in rapid reanalysis of the problem when the boundary conditions, initial conditions, or material properties change. Because symbolic differentiation is used, symbolic constants can be included in the differential equations. The method is applied to linear and non-linear DEs. Homogeneous and non-homogeneous linear DEs are solved in two steps. The first step is relatively time consuming and employs GP techniques to find basis functions that span the solution space. The second step is very fast and uses a Gram-Schmidt algorithm to compute the basis function multipliers to develop a complete solution for a given set of boundary conditions. For non-homogeneous linear DEs, in the first step the particular solution for the non-homogeneous linear DE is found in parallel with finding complementary solutions based on the homogeneous part of the DE. Following the same process as used for the homogeneous linear differential equation is used except that the particular solution is added to the scaled complementary solutions to satisfy the boundary conditions. For non-linear equations there is not a direct and widely applicable methodology for obtaining analytical solutions. In this case, the complete solution including the boundary conditions is evolved simultaneously. Candidate solutions are expressed with a LISP-like notation to display the functions using standard ASCII output.

Bryden et al. [26] explores evolutionary algorithms that use combinatorial graphs to limit possible crossover partners. These graphs limit the speed and manner in which information can spread giving competing solutions time to mature. This use of graphs is a computationally inexpensive method of picking a global level of trade-off between exploration and exploitation. Among other examples, this paper presents the solution of a simple differential equation ($y'' - 5y' + 6y = 0$) by a GP technique and using the graph based ideas. Candidate solutions are expressed using parse trees representing mathematical expressions $f(x)$. The fitness

function was computed as the sum of the error function $E(x) = f(x)'' - 5f'(x) + 6f(x)$ over 100 equally spaced samples. This is the squared deviation points in the range from agreement with the differential equation.

Another GP approach can be seen in Sobester et al. [3], where they propose a technique for the mesh-free solution of elliptic partial differential equations where least-squares collocation principle has been employed to define an appropriate objective function. In that work no particular function basis is used, but symbolic regression is performed. This makes the search space very large. In the case of problems defined on geometrically simple domains, the solution evolved by GP is modified adding additional terms, such that the boundary conditions are satisfied by construction. To satisfy the boundary conditions for geometrically irregular domains, GP model is combined with a radial basis function network.

Seaton et al. [28] investigate the influence of the complexity when symbolic solutions to a differential equation are found. They show that reducing the search space can improve significantly the algorithm performances. A possible approach for reducing the search space dimension is using some kind of function basis for building candidate solutions. The equations presented in the paper are first and second ordinary linear and non linear differential equations. The fitness function is built adding to the residuals the error at the boundary conditions. The solutions are expressed as mathematical expressions using the following functions and operators: $+$, $-$, $/$, $*$, sin, cos, ln and exp.

2.3.5 Grammatical Evolution (GE)

Grammatical evolution (GE) is a relatively new evolutionary computation technique pioneered by Conor Ryan, JJ Collins and Michael O'Neill in 1998 [46] at the BDS Group in the University of Limerick. It is related to the idea of genetic programming in that the objective is to find an executable program, or program fragment, that will achieve a good fitness value for the given objective function. In most published work on Genetic Programming, a LISP-style tree-structured expression is directly manipulated, whereas Grammatical Evolution applies genetic operators to an integer string, subsequently mapped to a program (or

similar) through the use of a grammar. One of the benefits of GE is that it is possible to apply standard mutation and recombination operators because candidate solutions are expressed by means of linear codons. A more deep description of GE will be provided in section 3.1.

In the same way than GP, GE has been applied to solve DEs. Thus, Tsoulos and Lagaris [1] solve a set of ordinary and partial DEs in a closed form analytical form using GE. If the grammar has a rich function repertoire, and the DE has a closed form solution, it is very likely that the method will recover it. If however the exact solution can not be represented in a closed form, the method will produce a closed form approximant. Fitness function is built adding to the residuals at the inner collocation points, the errors at the boundary conditions. To compute the fitness function, derivatives must be obtained. This is done using a symbolic mathematical engine.

GE has been used as well for enhancing the constructed neural method in Tsoulos et al. [2]. The main advantage of this approach is that the user does not choose a priori the number of neuron cells. In that contribution local search is employed over some individuals. That paper is based on a previous work from the same authors [47] where a new method for neural network evolution that evolves the network topology along with the network parameters is described. The proposed method uses GE to encode both the network and the parameters space. In [2], each dependent variable (one for DEs and several for systems of DEs) is computed as the output of a neural network. This output is a summation of different sigmoidal units:

$$N(\mathbf{X}, \mathbf{P}) = \sum_{i=1}^{H} \beta_j \sigma \left(\sum_{j=1}^{D} \alpha_{ij} x_j + b_i \right), \tag{2.3}$$

where D independent variables are considered, H is the number of hidden units, α_{ij} is the weight connection between input j and hidden unit i, β_j is the connection weight between the hidden unit j and the unique output. The coefficient b_i is the bias of the hidden unit i. The independent variables vector is $\mathbf{X} = [x_i]$. The vector with all the weights and bias is $\mathbf{P} = [\alpha_{ij}, \beta_j, b_i]$. Only the sigmoidal function is used as activation function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.4}$$

The fitness evaluation uses penalization. Thus, the proposed penalty function is used to force the neural network to train on the boundary conditions (PDEs) or the initial conditions (ODEs). The error function represents the neural network's misclassification rate and is necessary in order to measure its efficiency. A Powell's variant of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) local optimization method is used each 20 generations over 20 individuals. Several problems are solved with a very good performance, achieving errors between 10^{-5} and 10^{-9}. The generations needed are between 43 and 1422. Solutions found have good generalization properties because the errors remain low not only in the training points, but as well in test points.

Balbasubramaniam et al. [27] propose a novel approach to find the solution of the matrix Riccati differential equation (MRDE) for nonlinear singular systems using GE. The needs for solving this equation often arise in analysis and synthesis such as linear quadratic optimal control systems, robust control systems, performance criteria, stochastic filtering and control systems, model reduction, differential games, etc. The GE solution is compared with a more classical one using Runge Kutta (RK) method. The GE solutions are expressed as mathematical expressions given by the grammar. Standard terminals are used, such as $+$, $-$, $/$, $*$, sin, cos, exp and ln. Fitness function is built as in other approaches adding to the residuals computed in the collocation points the errors at the boundary conditions. The paper shows that solving MRDE is equivalent to solving the system of nonlinear DEs. According to the authors, a GE approach can yield a solution of MRDE significantly faster than the RK method.

2.3.6 Heuristics using Artificial Neural Networks

Artificial neural networks (ANNs) are computational models inspired by an animal's central nervous systems (the brain, in particular) which are able to learn as well as recognize patterns. Artificial neural networks are generally presented as systems of interconnected *neurons* which can compute values from inputs. Like other machine learning methods (systems that learn from data) neural networks have been used to solve a wide variety of tasks that are

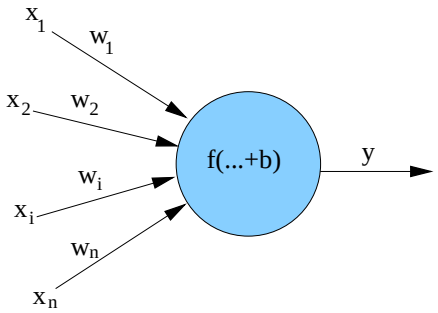

Figure 2.3: Schematic of an artificial neuron.

hard to solve using ordinary rule-based programming, including computer vision and speech recognition.

A neural network consists of an interconnected group of artificial neurons. The neurons are grouped in layers. Neurons of individual layers are independent of each other. This naturally enables a degree of parallelism in the implementation. In Fig. 2.3 an individual neuron is plot. The output of a neuron is the nonlinear weighted sum of the inputs computed using an activation function:

$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{2.5}$$

where y is the neuron output, x_i are all the inputs to the neuron, b is a bias and f is the activation function. Note that the inputs are added with different weights w_i.

Several activation function can be used. Even it is possible to used different activation functions in the same network. In Table 2.7 some activation functions are given. As we will see in next sections, derivatives of the activation functions would be useful.

Artificial neural network types vary from those with only one or two layers of single direction logic, to complicated multi-input many directional feedback loops and layers. On the whole, these systems use algorithms in their programming to determine control and organization of their functions. One of the most common type of interconnection is a feed-forward one, where connections between the units do not form a directed cycle. This is

Name	Function $f(x)$	First Derivative
Linear	x	1
Sigmoidal	$1/\left(1+e^{-x}\right)$	$f(x)\left(1-f(x)\right)$
Arctangent	$\arctan\left(x\right)$	$1/\left(1+x^2\right)$
Hyperbolic tangent	$\tanh\left(x\right)$	$1-f^2(x)$

Table 2.7: Activation functions for ANNs.

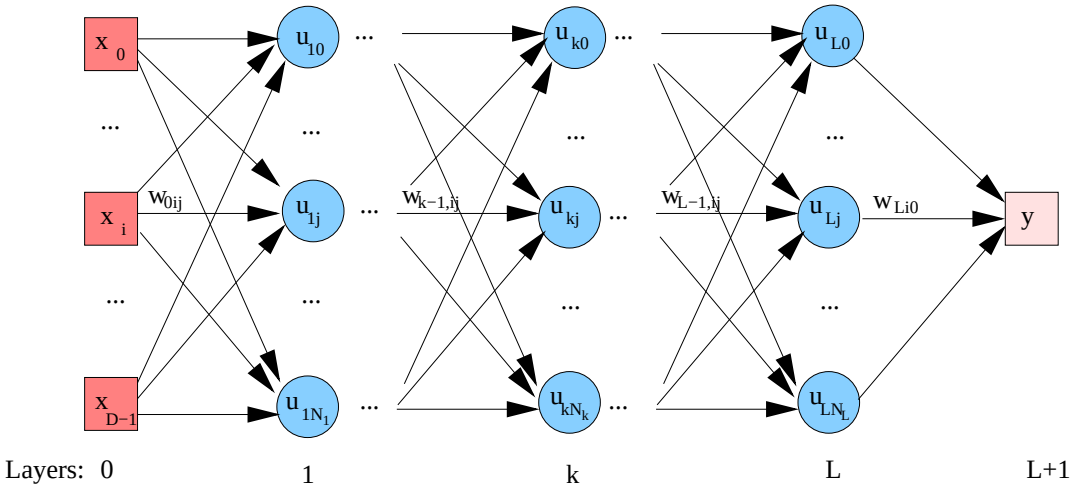

Figure 2.4: Generic Neural Network with L hidden layers. For the sake of clarity, only one output is plot.

different from recurrent neural networks. The feed-forward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

In the classic use of ANNs, several type of learning algorithms are reported. The learning process consist of seeking the optimum values of the weights and bias to solve a specific task. Generally speaking, exist three types of learning: *Supervised*, *Unsupervised* and *Reinforcement* learning. Most of the algorithms used in training artificial neural networks employ some form of gradient descent, using *backpropagation* to compute the actual gradients. It is common to split the data in two sets: a trained set of data, and test data.

Although in this survey the papers that solve DEs using heuristics are presented according to the optimization algorithm (GA, Differential Evolution, Genetic Programming, etc), here we will provide those works which use ANN to express the candidate solutions. As we will see in the next lines, we adopt this approach because of the great number of works using ANN to solve DEs.

We can find several works where ANNs are used to solve differential equations (a survey about solving DE with a special type of neural networks and radial basis functions can be consulted in [19]). As we will see in the next lines, ANNs are used to expressed the solution of a particular DE. Classic training algorithms such as backpropagation, are not used because the cost function not only depends on the ANN output, but as well on its derivatives. Therefore the data can not be split in two different sets. Normally, some heuristic method is employed to adjust the ANN parameters (weights, bias, etc). However, as it will be indicated in the next lines, some works use backpropagation in very specific problems, where the output of the ANNs can be known in advance considering special characteristic of the equation or the type of solution.

Thus, Puffer et al. [11] present a special type of ANN called cellular neural networks (CNN) to solve DEs. CNNs are a parallel computing paradigm similar to neural networks, with the difference that communication is allowed between neighbouring units only. The dynamical behaviors of CNN processors can be expressed mathematically as a series of ordinary differential equations, where each equation represents the state of an individual processing unit. The behaviour of the entire CNN processor is defined by its initial conditions, the inputs, the cell interconnection (topology and weights), and the cells themselves. Interactions between cells are local and usually translation invariant, i.e. a connection from a cell j towards another cell i only exists if j is part of i's neighborhood $N(i)$ and its type and strength depends only on the relative position of j with respect to i. Thus the number of connections increases only linearly with the number of cells. State and output $v_{x,i}^{m}$ of a cell i in layer m are real numbers, their dynamics being determined by state equations of the form

$$\frac{dv_{x,i}^{m}(t)}{dt} = -g\left(v_{x,i}^{m}(t)\right) + \sum_{m'=1}^{M} \sum_{i+l \in N(i)} a_{l}^{m'm}\left(v_{y,i+l}^{m'}(t), v_{y,i}^{m}(t); p_{a,l}^{m'm}\right) + \sum_{i+l \in N(i)} b_{l}^{m}\left(v_{u,i+l}^{m}(t), v_{u.i}^{m}(t); p_{b,l}^{m}\right)$$

with $v^m_{y,i+l} = f_{out}\left(v^m_{x,i}\right)$. The feedback connection from a cell $i+l$ in layer m' towards another cell i in layer m is defined by the weight function $a^{m'm}_l$, while the functions b^m_l define the connection from an input node $v^m_{u,i+l}$ towards a cell i, the parameter vectors $p^{m'm}_{a,l}$ and $p^m_{b,l}$ have to be determined (e.g. by a learning algorithm). This type of networks are used for approximating the solutions of some derivative equations. CNN are suitable for reproducing the behavior of very non-linear differential equations regarding with time as independent variable. Because in each neuron a derivative must be solved, it is not straightforward to compute the outputs of the network. The results show that, depending on the training pattern, solutions of various PDE can be approximated with high accuracy by a simple CNN structure. Results for two nonlinear PDE, Burgers' equation and the Korteweg-de Vries equation, are discussed in detail. The cost function is computed as the mean square error of the difference between the output of the CNN and a reference solution. Therefore, because a reference solution is needed, this approach is opposite from the aforementioned works: here the solution is known and what it is sought is an DE which produce the observed outputs. To train the CNN, a Downhill Simplex method is used which requires no explicit gradient information.

Lagaris et al. [12] use a feed-forward neural network to codify the solutions of DEs. The trial solutions are computed as a sum of two parts. The first part satisfies the initial/boundary conditions and contains no adjustable parameters. The second part is constructed so as not to affect the initial/boundary conditions. To fix ideas, the authors present the solution to the following first order ODE:

$$\frac{dy}{dx} = f\left(y, x\right),$$

in the domain $x \in [0, 1]$ with the boundary condition $y\left(0\right) = A$, being A a constant. The proposed approach consist of expressing the trial solution as

$$y_t\left(x\right) = A + xN\left(x, \mathbf{p}\right)$$

where $N\left(x, \mathbf{p}\right)$ is the output of a feed-forward neural network with one hidden unit and weights $\mathbf{p}$. Note that by construction, y_t satisfies the boundary conditions for all possible

$N(x, \mathbf{p})$. The cost function to be minimized is built as

$$E(\mathbf{p}) = \sum_i \left(\frac{dy_t(x_i)}{dx} - f(y_t(x_i), x_i) \right)^2,$$

where x_i are the collocation points. As we can observe, this approach is problem dependent because in some cases could be difficult to split the candidate solutions in the two terms. Neural network weights and bias are optimized using a quasi-Newton Broyden-Fletcher-Goldfarb-Shanno method. This decompositions has the advantage of reducing the difficulty in the optimization of the neural network, but is not a general approach. The neural networks only have one hidden layer of 10 units and can be expressed as equation (2.3). The activation functions are the sigmoid functions, Eq. (2.4). Authors show that is easy to symbolically compute the derivatives of the neural network respect to the weights and the inputs. A grid of 10 points is used in the ordinary and system differential equations are used, wheres 100 points are used in the partial derivative equations problems. The method exhibits excellent generalization performance since the deviation at the test points was in no case greater than the maximum deviation at the training points. This is in contrast with the finite element method where the deviation at the testing points was significantly greater than the deviation at the training points. The accuracy was measured using the exact solutions, and was around 10^{-6} and 10^{-7}.

He et al. [13] present and ANN used as a controller of a first order system. The controller must maintain the output of the system in a desired level. Therefore, the ANN can be trained in the classic sense using backpropagation algorithm. Authors present an extended backpropagation algorithm. Some advantages of this approach can be devised, as easily and quickly finding approximate solutions for complicated first-order partial differential equations. The differential equation solved is a very particular one (linear first order equation $x' = f(x)$). Let the output of an n-layer feed-forward network be $a_i^n(w, b)$, (where w and b are the weights and the bias of the network, respectively). The neural network can be described with the

following expressions

$$o^k = w^k a^{k-1} + b^k \qquad a^k = f^k\left(o^k\right) \qquad k = 1, 2, \ldots, n$$

where $a^0 = x$ is an n_0-dimensional input vector, a^k and f^k are the n_k-dimensional output and the activation function vectors in the kth layer, respectively. The activation functions in the hidden layers of the network were taken to be hyperbolic tangent sigmoid functions. The numbers of nodes in the hidden layers were fixed to 30. The networks were trained using a modification of the backpropagation algorithm taking into account the derivatives of the neural network outputs:

$$\triangle w^k = -\eta \frac{\partial E}{\partial w^k} \qquad \triangle b^k = -\eta \frac{\partial E}{\partial b^k} \ ,$$

being E the error of the neural network taking into account the outputs and the derivatives and η the learning rate.

Aarts and Veer [34] present a method to solve a one dimensional unsteady groundwater in a confined semi-infinite aquifer towards open water, without entrance resistance:

$$u_{xx} - u_t = 0,$$

with the boundary conditions $u\left(0, t\right) = \sin\left(t\right)$ and $u\left(x, 0\right) = \exp\left(-\sqrt{0.5}x\right)\sin\left(-\sqrt{0.5}x\right)$. Candidate solutions are expressed with a combination of ANNs. The tuning of the network parameters are performed using an evolutionary algorithm. The topology of the ANNs are built using information of the specific DE to be solved. So it is not a generic approach. The fitness function is computed comparing the ANNs outputs with the DE at several collocation points, that is, the residuals are employed for this task.

The same approach than [12] has been successfully applied to a system of partial differential equations which models a non-steady fixed bed non-catalytic solid-gas reactor in Parisi et al. work [14]. As usual, residuals (errors of the differential equation at one specified point) at the inner and boundary collocation points are used to compute the fitness function. The

training (weights and bias tuning) was performed using first a GA and then a gradient descent method. The solution was found with uniform accuracy and the trained neural network provides a compact expression for the analytical solution over the entire finite domain. The neural network approximation has two units in the input layer, five units in the hidden layer and one unit in the output layer. The output neurons have activation functions as well:

$$N(\mathbf{X}, \mathbf{P}) = g\left(\sum_{i=1}^{H} \beta_j g\left(\sum_{j=1}^{D} \alpha_{ij} x_j + b_i\right)\right), \tag{2.6}$$

being all the activation functions

$$g(x) = \frac{2}{1 + \exp(-2x)} - 1.$$

Modification of the activation functions is needed because any output value different from the range of the activation function output can be achieved. The training set was composed by 110 equidistant points. The genetic algorithm takes around 500 generations and the gradient descent routine takes approximately $2 \cdot 10^4$ iterations to converge.

Sun et al. [15] present a neural network trained in order to give the solution of the Poisson differential equation, but in a traditional way, giving as training points the true solution of the equation. The main idea is that Poisson PDE has a closed solution in simple domains, such as a sphere. However, authors work with more complex domains to model the behavior of human tissues. Concretely speaking, the human brain is model using a spheroid. The network has only one hidden layer. Bipolar sigmoid and linear activation functions are employed. Several popular training algorithms, including resilient backpropagation, scaled conjugate gradient, Fletcher-Reeves, Polak-Ribiere, and Powell Beale were experimented. The scaled conjugate gradient algorithm performed the best, although the differences between algorithms were small. Although heavy computation is required to train the network when the computational system is constructed, this is a one-time procedure which does not affect the on-line performance.

Ramuhalli et al. [32] propose a finite-element neural network (FENN) obtained by embed-

ding a finite-element model in a neural network architecture that enables fast and accurate solution of the forward problem. Results of applying the FENN to several simple electromagnetic forward and inverse problems are presented. It is based on a discretization of the domain and it is a very specific method, only thought for solving a FEM solver in parallel. In the general case with M elements and N nodes in the FEM mesh, the input layer will have M inputs. The hidden layer has N^2 neurons. Each neuron in the hidden layer acts as a summation unit. The outputs of the hidden layer neurons are the elements K_{ij} of the global stiffness matrix. Each output neuron is also a summation unit followed by a linear activation function. As a conclusion, the major advantage of the FENN is that it represents the finite-element model in a parallel form, enabling parallel implementation in either hardware or software. Further, computing gradients in the FENN is very simple. But a mesh connectivity is required.

Shirvany et al. [17] use Multilayer Perceptron and Radial Basis Function neural network to solve the nonlinear Schröndinger equation in hydrogen atom:

$$\left.\begin{array}{l} H\psi\left(\mathbf{r}\right) = f\left(\mathbf{r}\right) \ in \ D \\ \\ \psi\left(\mathbf{r}\right) = 0 \ on \ \partial D \end{array}\right\},$$

where H is a differential operator, $f\left(\mathbf{r}\right)$ is a unknown function, $D \subset \mathbb{R}^3$ and ∂D is the boundary of D. The collocation method is adopted, discretization the continue domain in a finite number of points. As usual, the cost function for training the network is the sum squared error (SSE) of residuals. For solving the optimization problem, the gradient-descent backpropagation algorithm is used. This method can be used because of the particular form of the equation. For that, eigenvalues of the system must be used. The strategy consists of using a first guess for the eigenvalues. Knowing the eigenvalues, for a giving network, not only the output can be computed in the collocation points, but as well the SSE values. If after some steps, the energy function (residuals) did not converge to zero, the eigenvalues were wrong. Therefore, the eigenvalues should be increased and tried again. However, after

performing this process if the error cannot be reduced, the network is allowed to grow, i.e., the number of hidden neurons is increased by one.

Although Tsoulos et al. [2] was already mentioned in previous section because GE is employed, the candidate solutions are expressed using a feed-forward artificial neural network with one hidden level and one output. The output of the constructed neural network is a summation of different sigmoidal units.

Raja et al. [37] present a GA approach for solution of FDEs. The general form of FDEs solved in this work is

$$D^v y(t) = f(t, y(t), y'(t)),$$

with initial and boundary conditions given by

$$D^k y(0) = c_k, D^k y(t_0) = b_k,$$

with $k = 0, 1, 2, \cdots, \lceil v \rceil - 1$. In this method, a feed forward ANN is used to accurately model the equation and a GA is applied for learning of weights. The fitness function is built adding the square of the equation residuals at the some collocation points. Standard log-sigmoid activation functions of the ANN are replaced by exponential functions to facilitate the fractional derivative computations. The design scheme has been successfully applied to solve different types of linear and nonlinear ordinary FDEs. The results were compared with exact solutions, approximate analytic solution and standard numerical techniques. In case of linear ordinary fractional differential equations, relatively more accurate solutions were obtained than standard numerical methods. However, for complex nonlinear fractional differential equation, the same scheme is applicable, but with reduced accuracy. The advantage of this approach is that it provides the solution on continuous entire finite domain unlike the other numerical techniques.

In Yazdi and Pourreza [36] paper, a novel structure of unsupervised adaptive network-based fuzzy inference system (ANFIS) is presented to solve differential equations, where an ANN and a fuzzy system are combined. The presented solution of DE consists of two parts; the first part satisfies the initial/boundary condition and has no adjustable parameter

whereas the second part is an ANFIS which has no effect on initial/boundary conditions and its adjustable parameters are the weights of ANFIS. The unsupervised training of the network is performed using a hybrid learning algorithm which combines the least square method and the backpropagation algorithm. For that, the desired outputs to a given input must be computed. As in other approaches, this is done computing the residuals of the given equation. To do that, derivatives of outputs are calculated numerically.

Chen et al. [18] present a mesh-free numerical method for solving PDEs based on integrated radial basis function networks with adaptive residual sub-sampling training scheme. Numerical experiments solving several PDEs show that this algorithm with the adaptive procedure requires fewer neurons to attain the desired accuracy than conventional radial basis function networks. No detail description is given in the paper, but the training process is based on the residuals of the DE, both in the inner and the boundary condition points. Due to the particular type of problem solved, a linear equation system arise, which is solved by linear least-square programming.

A different approach dealing with ANNs consist of solving a family of DEs using traditional methods and training a neural network for building surrogate models. Following this line, El-Emam and Al-Rabeh [33] present a new hybrid adaptive neural network with modified adaptive smoothing errors based on GA to construct a learning system for complex problem solving in fluid dynamics. The system can predict an incompressible viscous fluid flow represented by stream function through symmetrical backward-facing steps channels.

Mosleh [21] presents a novel approach to solve system of fuzzy differential equations (SFDEs) with fuzzy initial values by applying the universal approximation method (UAM) through an artificial intelligence utility in a simple way. The model finds the approximated solution of SFDEs inside of its domain for the close enough neighborhood of the fuzzy initial points. The author proposes a learning algorithm for adjusting the fuzzy weights. At the same time, some examples in engineering and economics are designed. Due to the stochastic nature of the DE involved, training techniques of the ANN requires a mathematical background that is out of the scope of this survey.

In Zjavka and Abraham [40] paper, a method to predict failures in electronic circuits is

presented. For that, a special ANN is employed called differential polynomial neural network, which constructs and substitutes an unknown general sum partial differential equation with a total sum of fractional polynomial terms. Therefore, strictly speaking, in this paper authors do not present a method to solve DEs. In a later paper [41] by the same authors, the same techniques are applied to predict time series of wind speed. To adjust the networks, two different techniques are employed: simulated annealing (SA) and gradient steepest descent (GSD). As the target of these works is not solving a specific DE, but substitute an unknown DE by a simpler equation to achieve predictive capabilities in time series models, this paper is as well out of the scope of this survey.

In Raja [42] paper, a solution for the Pantograph functional DE is presented:

$$\frac{d^2y\left(t\right)}{dt^2} = f\left(t, y\left(t\right), y\left(\lambda t\right)\right),$$

with $t \in (0, T)$ and the boundary conditions $y\left(0\right) = b$ and $y\left(T\right) = c$, where b, c and λ are constants. Functional DE are a generalization of delay DE. To approximate the equation, feed-forward ANNs are appropriately combined to define an objective function. The weights of these networks are optimized to minimize the objective function value with help of SA, PS and GAs, Active set algorithm (ASA) and their hybrid combination. The cost functions are built as usual adding the equation residuals with the errors at the boundary conditions.

2.3.7 Particle Swarm Optimization

Particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. PSO optimizes a problem by having a population of candidate solutions, also called particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position but, is also guided toward the best known positions in the search-space, which are updated as better positions found by other particles. This is expected to move the swarm toward the best solutions.

PSO is originally attributed to Kennedy, Eberhart and Shi [48, 49] and was first intended for simulating social behavior, as a stylized representation of the movement of organisms in a bird flock or fish shoal. The algorithm was simplified and it was observed to be performing optimization. The book by Kennedy and Eberhart [50] describes many philosophical aspects of PSO and swarm intelligence. An extensive survey of PSO applications is made by Poli [51].

Several works using PSO to solve differential equations can be found in the literature. Thus in Khan et al. [29] paper, first order differential equations are expressed using ANNs, and the PSO is employed to adjust the network weights. The solutions are achieved on the continuous grid of time instead of discrete unlike numerical techniques. The fitness function is built with the DE residuals at the collocation points.

In Raja et al. [39] the same approach using in [37] is applied to solve the fractional order of the Ricatti differential equation:

$$\frac{d^v y\left(t\right)}{dt^v} = p\left(t\right) + q\left(t\right) y\left(t\right) + r\left(t\right) y^2\left(t\right),$$

with $0 < t \leq T$. Feed-forward artificial neural network is employed for accurate mathematical modeling and learning of its weights is made with heuristic computational algorithm based on PSO as a tool for the rapid global search method, and SA for efficient local search. The scheme is equally capable of solving the integer order or fractional order Riccati differential equations. The fitness function of each particle is calculated by defining an unsupervised error function which is formed by linear combination of the equation residuals in a predefined temporal instants.

Babaei [4] employs the Fourier series expansion, calculus of variation, and particle swarm optimization in the formulation of the problem. Both boundary value problems and initial value problems are treated in the same way. Boundary and initial conditions are both modeled as constraints of the optimization problem. The constraints are imposed through the penalty function strategy. The fitness function is built adding the residuals at the collocation points and adding the errors at the boundary conditions. Only ordinary DE problems are presented.

2.3.8 Support Vector Machines (SVM)

A support vector machine (SVM) constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space. It is a nonlinear generalization of the *Generalized Portrait* algorithm developed in Russia in the sixties by Vapnik, Lerner and Chervonenkis [52, 53]. In 1992, Boser, Guyon and Vapnik suggested a way to create nonlinear classifiers using kernel functions [54]. The current standard algorithm was proposed by Cortes and Vapnik in 1993 and published in 1995 [55]. Least Squares SVM (LS-SVM) is a modification of the original SVM approach where one finds the solution by solving a set of linear equations instead of a convex quadratic programming problem for classical SVMs.

Although SVMs have been traditionally used in classification and regression tasks, some works have been reported using SVMs for solving DEs. Thus, Mehrkanoon and Suykens [30] present a method to solve second order partial differential equations with variable coefficients. Given a general differential equation $\mathcal{L}u(z) = f(z)$ with $z \in \Sigma \subset \mathbb{R}^2$ subject to the boundary conditions $\mathcal{B}u(z) = g(z)$ with $z \in \partial\Sigma$, the approximated solution function is expressed as $u(z) = w^T\varphi(z) + d$, where w and d are parameters of the model that have to be determined. Function φ defines a kernel function in the form $K(z, r) = \varphi(z)^T \varphi(r)$. Then, the original problem is transformed into a minimization one

$$\min_{w,d,e} \frac{1}{2} w^T w + \frac{\gamma}{2} e^T e$$

subject to a set of constraints given by the collocation points and the boundary conditions. γ is a positive parameter, and components of vector e are defined as the errors between the approximated solution given by the LS-SVM and the differential equation at the collocation points and boundary condition points. Note that the transformed problem is a quadratic minimization under linear equality constraints, which enables an efficient solution. Authors test the algorithm on a set of linear PDEs in rectangular and irregular domains. The LS-SVM approach allows them obtain the solution of a linear PDE by solving a system of linear equations. For a nonlinear PDE one requires applying a Newton-type iterative method.

2.4 Conclusion

One the state of the art about solving DEs with heuristic methods has been exposed, some conclusions can be commented.

First of all, we can say that these non classical methods to solve DEs are relative recent, with the first papers appearing at the last decade of 20th century. So it is a young field with around 20 years of history. Some interest in the research community is observed, appearing new works every year. This interest can be partially explained thanks to the great advances in Evolutionary Computing techniques and in the computer hardware.

Works in the literature can be classified following different criteria. Thus, regarding how the candidate solutions are expressed, the most extended method (more than the 50% of the papers reported in this survey) employ artificial neural networks. Other classification can be done according the optimization process employed to adjust candidate solutions. A relative high number of papers use other techniques related with Genetic Programming, or some modern variation such as Grammatical Evolution. In order to increase the quality of the solutions, a reduce number of works (less than 20%) employ some type of local search, although it is not observed any predominant algorithm.

Regarding the global search process, there is not a predominant algorithm. It is important to notice here that backpropagation, one of the most extended training algorithm for ANN, in this survey is not the most extended. This is because ANN are not used as in the classical approach of machine learning. On the contrary, ANNs must adjust no only the unknown solution, but as well its derivatives.

Some papers focus on a very specific type of equation, such as Ricatti, Navier-Stokes or Schrödinger equations. Those papers which try to solve a wider set of equations, only present relative simple problems. Therefore we can conclude that the field of solving DEs using heuristic methods, or more specific ones such as Evolutionary Algorithms, is not completely explored a new ideas can be proposed.

Chapter 3

Background

In this chapter, all the search algorithms used in the present thesis are described: Grammatical Evolution (GE), Evolution Strategies (ES), Covariance Matrix Adaptation Evolution Strategies (CMA-ES) and Downhill Simplex (DS) method.

3.1 Grammatical Evolution (GE)

Grammatical Evolution (GE) was created by Conor Ryan, J. J. Collins and Michael O'Neill in 1998 [56] at the Biocomputing-Developmental Systems Group in the University of Limerick. GE is an evolutionary algorithm that can evolve complete programs in an arbitrary language using variable-length binary strings [46]. The binary genome determines which production rules in a Backus-Naur form (BNF) grammar definition are used in a genotype-to-phenotype mapping process to create a program.

Although GE is not the first instance in which grammars have been used with evolutionary approaches to automatic programming, GE presents a unique way of using grammars in the process of automatic programming. Variable-length binary string chromosomes are used. Each consecutive group of 8 bits represent a *codon*. The integer values associated to each codon are used in a mapping function to select an appropriate production rule from the BNF definition. GE does not suffer from the problem of having to ignore codon integer values because it does not generate illegal values. The issue of ensuring a complete mapping

of an individual onto a program comprised exclusively of terminals is partly resolved using *wrapping*. This technique involves re-using parts of the genome which has already being expressed and draws inspiration from the overlapping genes phenomenon exhibited by many bacteria, viruses, and mitochondria.

Since the publication of the first paper in 1998, a lot of works have been reported about GE. Some of them present variations of the original GE algorithm, such as O'Neill and Ryan [57] where several crossover operators are investigated, or O'Neill et al. [58] where a position-independent variation on Grammatical Evolution's genotype-phenotype mapping process called π-GE is reported. Left Hand Side (LHS) crossover is a structure preserving crossover in GE described in Harper and Blair [59, 60]. In other work by Harper and Blair [61] dynamically defined functions is presented. Several extensions to standard GE has been presented in Nicolau and Dempsey [62]. Ortega et al. [63] present a GE approach with semantics. A GE variation guided by reinforcement was presented in Migo and Aler [64]. Byrne and O'Neil [65] study the difference between Persistent Random Constants and Digit Concatenation as methods for generating constants. A good survey of works using GP, and in particular GE, can be consulted in Langdon and Gustafson[66]. A specific book about GE was published in 2004 by Alfonseca and Ortega [67] and some Ph.D. thesis are available [68, 69, 70] in the literature.

3.1.1 Backus-Naur Form

Backus-Naur Form (BNF) is a notation for expressing the grammar of a language in the form of production rules. BNF grammars consist of *terminals*, which are items that can appear in the language, and *nonterminals* which can be expanded into one or more terminals and nonterminals. A grammar can be represented by the tuple $\{N, T, P, S\}$, where N is the set of nonterminals, T the set of terminals, P a set of production rules that maps the elements of N to T, and S is a start symbol that is a member of N. When there are several productions that can be applied to one particular element in N, the choice is delimited with the symbol '|'.

A simple example of a grammar for symbolic regression could be:

```
N={expr,op,func}
T={sin,cos,tan,+,-,/,*,x,1.0,(,)}
S=<expr>
```

And the production rules P the following:

```
(1)<expr>::=<expr><op><expr>    (0)
           | (<expr><op><expr>) (1)
           | <func>(<expr>)     (2)
           | <var>              (3)
(2)<op>::=+  (0)
         | - (1)
         | / (2)
         | * (3)
(3)<func>::=sin  (0)
           | cos (1)
           | tan (2)
(4)<var>::=x    (0)
          | 1.0 (1)
```

where the numbers in parenthesis are used to numerate and identify each rule associated to one nonterminal.

3.1.2 Mapping process

The genotype is used to map the start symbol onto terminals by reading codons of 8 bits and generating a corresponding integer value from which an appropriate production rule is selected by using the mapping function

$$rule = (codon\,integer\,value)\,mod\,(number\,of\,rules\,for\,the\,current\,nontermial)$$

where modular arithmetic is used. Each time a production rule has to be selected to map from a nonterminal, another codon is read. In this way, the system traverses the chromosome. During the genotype-to-phenotype mapping process it is possible for individuals to run out

of codons. In this case, we wrap the individual and reuse the codons, that is, the traverse of the chromosome continues from the beginning. If the decoding process has not finished at the end of the chromosome, another wrap step is performed. A limit in the wrap steps should be specified because some ill cases need an infinite number of codons.

In GE, each time the same codon is expressed, it will always generate the same integer value, but depending on the current nonterminal to which it is being applied, it may result in the selection of a different production rule. What is crucial, however, is that each time a particular individual is mapped from its genotype to its phenotype, the same output is generated. This results because the same choices are made each time. It is possible that an incomplete mapping could occur even after several wrapping events and, in this case, the individual in question is penalized. The selection and replacement mechanisms then operate accordingly to increase the likelihood that this individual is removed from the population. An incomplete mapping could arise if the integer values expressed by the genotype were applying the same production rules over and over.

To reduce the number of invalid individuals being passed from generation to generation, a steady-state replacement mechanism is employed.

In the following example, it is show how the genotype $(10, 28, 1, 98, 45, 3, 1, 12, 35)$ of an individual is decoded to finally obtain the phenotype $\cos(\sin(1.0) + x)$ using the grammar already given in the previous section. Note that one wrapping process was needed.

```
Genotype:  10 28 1 98 45 3 1 12 35
Step 0 :  10 28 1 98 45 3 1 12 35 -> <exp>
Step 1 :  10 28 1 98 45 3 1 12 35 -> 10 mod 4 = 2 -> <func>(<expr>)
Step 2 :  10 28 1 98 45 3 1 12 35 -> 28 mod 3 = 1 -> cos(<expr>)
Step 3 :  10 28 1 98 45 3 1 12 35 -> 1 mod 4 = 1 -> cos((<expr><op><expr>))
Step 4 :  10 28 1 98 45 3 1 12 35 -> 98 mod 4 = 2 -> cos((<func>(<expr>)<op><expr>))
Step 5 :  10 28 1 98 45 3 1 12 35 -> 45 mod 3 = 0 -> cos((sin(<expr>)<op><expr>))
Step 6 :  10 28 1 98 45 3 1 12 35 -> 3 mod 4 = 3 -> cos((sin(<var>)<op><expr>))
Step 7 :  10 28 1 98 45 3 1 12 35 -> 1 mod 2 = 0 -> cos((sin(1.0)<op><expr>))
Step 8 :  10 28 1 98 45 3 1 12 35 -> 12 mod 4 = 0 -> cos((sin(1.0)+<expr>))
Step 9 :  10 28 1 98 45 3 1 12 35 -> 35 mod 4 = 3 -> cos((sin(1.0)+<var>))
Step 10:  10 28 1 98 45 3 1 12 35 -> 10 mod 2 = 0 -> cos((sin(1.0)+x))
Phenotype:  cos((sin(1.0)+x))
```

3.1.3 Evolutionary Algorithm

As the population being evolved comprises simple binary strings, the standard GE algorithm does not employ any special crossover or mutation operators and an unconstrained search is performed on these strings due to the genotype-to-phenotype mapping process that will generate syntactically correct individuals. Any evolutionary algortihm can be adopted, although the most common option should be a variable-length genetic algorithm with standard genetic operators of mutation and crossover.

3.2 Evolution Strategies (ES)

Evolution Strategies (ES) were created in the early 1960s and developed further in the 1970s and later by Ingo Rechenberg, Hans-Paul Schwefel and their co-workers. Several good introductory articles can be consulted in the literature, such as [71, 72].

ES belong to the family of evolutionary algorithms that address optimization problems in continues domains by implementing a repeated process of (small) stochastic variations followed by selection: in each generation, new offspring (or candidate solutions) are generated from their parents (candidate solutions already visited), their fitness is evaluated, and the better offspring are selected to become the parents for the next generation.

One of the main contributions of ES to the field of Evolutionary Computing is the *self-adaptation* of the mutation parameters . In general, self-adaptivity means that some parameters of the ES are varied during a run. For that, the parameters are included in the chromosomes and co-evolve with the solutions.

A great number of ES references can be consulted in the Beyer's ES tutorial [71], although this work dates from 2002. A lot of papers devoted to improve the performance of the standard ES are available in the literature. Thus, in order to improve the *self-adaptive* property of strategy parameters, a new extended ES called Robust-ES is proposed in Ohkura et al. [73]. An interesting application using ES and Fourier series to difficult optimization tasks can be consulted in Leung and Liang [74]. In Shir and Back [75, 76] niching methods are applied

to ES. In Kramer and Schwefel[77] several new constraint handling methods for ES are presented. In Debski et al. [78] three new mechanisms for maintaining population diversity in (μ, λ)-ES are introduced: deterministic modification of standard deviations, crowding, and elitism. Another important variation of ES is Natural-ES [79], where self-adaptation of the mutation matrix is derived using a Monte Carlo estimate of the natural gradient towards better expected fitness. Several papers presents theory and applications of this new approach, such as Glasmachers et al. [80].

3.2.1 Representation

Evolution strategies most commonly address the problem of continuous black-box optimization. The search space is the continuous domain, $\mathbb{R}^n$, and solutions in search space are n-dimensional vectors, denoted as $\mathbf{x}$. We consider an objective or fitness function $f : \mathbb{R}^n \to \mathbb{R}, \mathbf{x} \to f(\mathbf{x})$ to be minimized. We make no specific assumptions on f, other than that f can be evaluated for each $\mathbf{x}$, and refer to this search problem as black-box optimization. The objective is, loosely speaking, to generate solutions ($\mathbf{x}$-vectors) with small f-values (assuming a minimization problem) while using a small number of f-evaluations.

Standard representation of the objectives variables x_i are adopted, so the genotype space is identical to the phenotype space. However, modern ES employs self adaptation, which requires coding in the genotype several strategy parameters related to the mutation process.

3.2.2 Mutation Operators

The mutation operator introduces *small* variations by adding a point symmetric perturbation to the result of recombination. This perturbation is drawn from a multivariate normal distribution, $N(0, \mathbf{C})$, with zero mean (expected value) and covariance matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$. We have $\mathbf{x} + N(0, \mathbf{C}) \sim N(\mathbf{x}, \mathbf{C})$, meaning that $\mathbf{x}$ determines the expected value of the new offspring individual. We also have $\mathbf{x} + N(0, \mathbf{C}) \sim \mathbf{x} + \mathbf{C}^{1/2} N(0, \mathbf{I})$, meaning that the linear transformation $\mathbf{C}^{1/2}$ generates the desired distribution from the vector $N(0, \mathbf{I})$ that has independent and identically distributed $N(0, 1)$ components.

Based on multivariate normal distributions, three different mutation operators can be distinguished [81]: *uncorrelated with one step size, uncorrelated with n step sizes,* and *correlated*. In the following subsections, these operators are described.

Uncorrelated Mutation with One Step Size

The covariance matrix is proportional to the identity, i.e., the mutation distribution follows $\sigma N(0, \mathbf{I})$ with step-size $\sigma > 0$. The distribution is spherical and invariant under rotations about its mean. Other authors [72] call this type of mutation operator *spherical* or *isotropic*. The same distribution is used to mutate each component x_i, therefore we only have one strategy parameter σ in each individual. The mutation mechanism is sketched according to the following expressions:

$$\sigma' = \sigma e^{\tau N(0,1)},$$

$$x_i' = x_i + \sigma' N_i(0,1),$$

where $N(0,1)$ denotes a random variable with standard normal distribution. The proportionality constant τ is an external parameter called *learning rate* and is normally proportional to $1/\sqrt{n}$ being n the problem dimensionality.

Uncorrelated Mutation with n Step Sizes

The covariance matrix is a diagonal matrix, i.e., the mutation distribution follows $N(0, diag(\boldsymbol{\sigma}))$, where $\boldsymbol{\sigma}$ is a vector of coordinate-wise standard deviations and the diagonal matrix $diag(\boldsymbol{\sigma})$ has eigenvalues σ_i^2 with eigenvectors $\mathbf{e}_i$. The principal axes of the ellipsoid are parallel to the coordinate axes. This case includes the previous isotropic case. This operator is also known *axis-parallel* [72].

The mutation mechanism is sketched according to the following expressions:

$$\sigma_i' = \sigma_i e^{\tau' N(0,1) + \tau N_i(0,1)},$$

$$x_i' = x_i + \sigma_i' N_i(0,1),$$

where τ' is proportional to $1/\sqrt{2n}$ and τ is proportional to $1/\sqrt{2\sqrt{n}}$ and, as in the previous case, are called *learning rates.*

Correlated Mutation

In this type of operator, the covariance matrix is symmetric and positive definite (i.e. $\mathbf{xCx} > 0$ for all $\mathbf{x} \neq \mathbf{0}$), generally non-diagonal and has $(n^2 + n)/2$ degrees of freedom (control parameters). The general case includes the previous axis-parallel and spherical cases.

The rationale behind correlated mutations is to allow the ellipsoids to have any orientation by rotating them with the covariance matrix $\mathbf{C}$. The scaling factors σ_i and rotating angles α_{ij} are related with the elements of the covariance matrix c_{ij} in the following way:

$$\left.\begin{array}{r} c_{ii} = \sigma_i^2 \\ c_{ij,i\neq j} = \frac{1}{2}\left(\sigma_i^2 - \sigma_j^2\right)\tan\left(2\alpha_{ij}\right) \end{array}\right\}$$

Several techniques are reported in the literature about how to evolve the covariance matrix $\mathbf{C}$.

3.2.3 Recombination

Recombination is the operator used to obtain one child or several children given at least two parents. There are two recombination variants depending on how the parent alleles are mixed. Thus, in *discrete recombination* one of the parents alleles is randomly chosen with equal chance for either parents. On the other hand, in *intermediate recombination* the value of the parent alleles are averaged.

Other classification of recombination operator can be followed regarding the number of parents involved. When only two parents are used to create a child, *local recombination* is used. On the contrary, we can use *global recombination*, where the complete population behaves as a parent. In this case, the exact number of parents in global recombination cannot

be defined in advance.

ES typically uses global recombination. Besides, discrete recombination is used for the object variable part, meanwhile intermediary recombination is recommended for the strategy part [81]. With this approach, the scheme preserves diversity allowing the trial of very different combination of values, whilst the strategy parameters are adapted in a cautious mode.

3.2.4 Parent selection

Parent selection is not biased by fitness value, they are randomly selected. As global recombination is normally used, instead of talking about parent individuals, the whole population behaves as *parent* population.

3.2.5 Survivor Selection

Normally two selection schemes are used in ES. The (μ, λ) scheme creates λ children individuals using a parent population composed of μ individuals. The best μ are chosen deterministically only from the offspring population. The $(\mu + \lambda)$ scheme as well selects deterministically μ individuals, but considering a population formed by the union of parents and offspring. Classical ES generally use (μ, λ) selection because the algorithm is in principle able to leave local optima and facilitate the self-adaptation mechanism. The selective pressure is very high because λ is typically much higher than μ. A 1/7 ratio is recommended [81].

3.3 Covariance Matrix Adaptation ES (CMA-ES)

The covariance matrix adaptation evolution strategy (CMA-ES) [82, 83, 72] is a *de facto* standard in continuous domain evolutionary computation. It is based in standard ES, but enhances the performances using several principles which they will described in following sections. The algorithm was developed by Hansen in 2001.

Algorithm 3.1 Simple random search algorithm.

Initialize $\mathbf{x}$ randomly
Until termination criteria **do**
    Sample a new position $\mathbf{y}$ in the surrounding of $\mathbf{x}$
    if $f(\mathbf{y}) < f(\mathbf{x})$ **then** $\mathbf{x} = \mathbf{y}$
Return $\mathbf{x}$

The CMA-ES is an attractive option for non-linear optimization if "classical" search methods, such as quasi-Newton methods or conjugate gradient methods, fail due to a non-convex or rugged search landscape. In fact, CMA-ES belongs to *random search* algorithms, also known as *direct-search*, *derivative-free*, or *black-box* methods. In Algorithm 3.1, a simple example of this type algorithm is sketched.

Learning the covariance matrix in the CMA-ES method is analogous to learning the inverse Hessian matrix in a quasi-Newton method. In the end, any convex-quadratic (ellipsoid) objective function is transformed into the spherical function. This can improve the performance on ill-conditioned and/or non-separable problems by orders of magnitude.

The CMA-ES overcomes typical problems that are often associated with evolutionary algorithms:

- Poor performance on badly scaled and/or highly non-separable objective functions.
- The inherent need to use large population sizes. The CMA-ES algorithm is designed in order to avoid having all the population in a same subspace even in small population sizes.
- Premature convergence of the population. Step-size control in CMAES prevents the population to converge prematurely.

An interesting feature of CMA-ES is its invariance properties, which can explain part of its good performance in different fields. Invariance properties of a search algorithm denote identical behavior on a set, or a class of objective functions. Thus, CMA-ES presents translation invariant. Further invariances, e.g. to certain linear transformations of the search space,

are highly desirable: they imply uniform performance on classes of functions and therefore allow for generalization of empirical results. In summary, the CMA-ES exhibits the following invariances.

- Invariance to order transformations, preserving the objective function value (i.e. strictly monotonic). The algorithm only depends on the ranking of function values.
- Invariance to angle transformations (rigid), preserving the search space (rotation, reflection, and translation) if the initial search point is transformed accordingly.
- Invariance to a scaling of variables if the initial diagonal covariance matrix is scaled accordingly.
- Invariance to any invertible linear transformation of the search space.

Fig. 3.1 shows a simple example on a two-dimensional problem[1]. The spherical optimization landscape is depicted with solid lines of equal fitness values. Individuals in the population are represented with dots. The distribution of the population is marked with dotted lines. Note how the distribution expands, rotates and contracts along the generations. On this simple problem, the population concentrates over the global optimum within a few generations.

A list of applications of CMA-ES is provided by Hansen in his web page[2]. Although the list is not exhaustive and is not updated, it shows the good acceptance of the algorithm in a lot of fields. Thus, in Hohm and Zitzler [84] we can see an application in molecular biology where CMA-ES is used for parameter estimation of ordinary differential equation-based gene regulatory network models. A photographic supra-projection application is presented in Santamaría et al. [85] where a forensic process that aims to identify a missing person from a photograph and a skull found is described. The craniofacial superimposition as a 3D-2D image registration problem is solved by means of a CMA-ES. In a more recent work [86], the craniofacial superimposition problem is improved taking into account the different

[1]Figure extracted from http://en.wikipedia.org/wiki/CMA-ES
[2]https://www.lri.fr/~hansen/cmaapplications.pdf

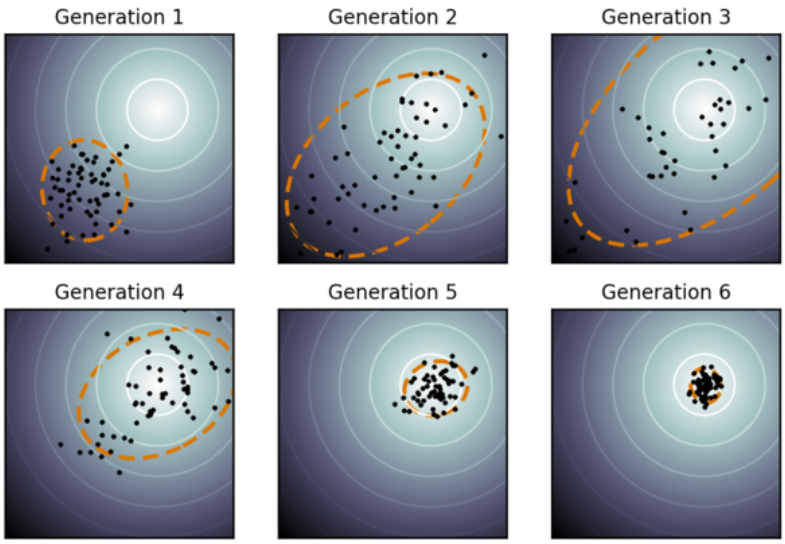

Figure 3.1: Example of a run with CMA-ES on a simple two-dimensional problem.

sources of uncertainty. As in the previous work, the optimization problem is solved using CMA-ES. Colutto et al. [87] present a generalization of CMA-ES in Riemannian manifolds. More concretely, the CMA-ES algorithm is used to the segmentation of 3-D voxel images obtaining a 3D model of a human brain from medical images. In [88] a design of resonator dielectric antenna using a CMA-ES is presented. Other interesting medical application work is Gong et al. [89], where a multiple-object 2-D/3-D registration technique for non-invasively identifying the poses of fracture fragments in the space of a preoperative treatment plan. Other application of antenna design can be consulted in Gegory et al. [90], where several symmetries are exploited in order to obtain an ultra-wideband device.

3.3.1 Sampling

A population of new search points (individuals, offspring) is generated by sampling a multivariate normal distribution. The basic equation for sampling the search points for generation g is as follows

$$\mathbf{x}_k^{(g+1)} \sim \mathbf{m}^{(g)} + \sigma^{(g)} N\left(\mathbf{0}, \mathbf{C}^{(g)}\right), \tag{3.1}$$

for k going from 1 to all the population λ. In the above expression $\sim$ denotes the same distribution on the left and right side, $N\left(\mathbf{0}, \mathbf{C}^{(g)}\right)$ is a multivariate normal distribution with zero mean and covariance matrix $\mathbf{C}^{(g)}$, $\mathbf{x}_k^{(g+1)} \in \mathbb{R}^n$ is the k-th offspring (individual, search point) from generation $g+1$, $\mathbf{m}^{(g)} \in \mathbb{R}^n$ is the mean value of the search distribution at generation g, $\sigma^{(g)}$ is the step size at generation g and $\mathbf{C}^{(g)} \in \mathbb{R}^{n \times n}$ is the covariance matrix at generation g.

3.3.2 Selection and Recombination

The new mean $\mathbf{m}^{(g+1)}$ of the search distribution is a weighted average of μ selected points from the sample $\mathbf{x}_1^{(g+1)}, \cdots, \mathbf{x}_\lambda^{(g+1)}$:

$$\mathbf{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}^{(g+1)}, \tag{3.2}$$

where $\mu \leq \lambda$ is the parent selection size, w_i are positive weight coefficients for recombination ($\sum_{i=1}^{\mu} w_i = 1$) and $\mathbf{x}_{i:\lambda}^{(g+1)}$ i-th best individual out $\mathbf{x}_1^{(g+1)}, \cdots, \mathbf{x}_\lambda^{(g+1)}$ from Eq. (3.1).

3.3.3 Adapting the Covariance Matrix

How the Covariance Matrix $\mathbf{C}$ is updated is the central part of the CMA-ES algorithm. Here we do not try to give a detailed description of the process, neither the theoretical foundations of the algorithm. The interested reader can consult several Hansen's papers, such as [83]. However, some basics will be described in this section.

A maximum-likelihood principle, based on the idea to increase the probability of successful candidate solutions and search steps, is exploited by the CMA-ES algorithm. The mean of the distribution is updated such that the likelihood of previously successful candidate solutions is maximized. The covariance matrix of the distribution is updated (incrementally) such that the likelihood of previously successful search steps is increased. Both updates can be interpreted as a natural gradient descent. Also, in consequence, the CMA-ES conducts an iterated principal components analysis of successful search steps while retaining all principal

axes.

3.3.4 Step Size Control

Two search paths are maintained, s_σ and s_c. The first path, s_σ, accumulates steps in the coordinate system where the mutation distribution is isotropic and which can be derived by scaling in the principal axes of the mutation ellipsoid only. Under neutral selection, $s_\sigma \sim N(0, \mathbf{I})$ and $\log \sigma$ is unbiased. The second path, s_c , accumulates steps, disregarding σ, in the given coordinate system. The covariance matrix update consists of a rank-one update, based on the search path s_c , and a rank-μ update with μ nonzero recombination weights w_k. Under neutral selection the expected covariance matrix equals the covariance matrix before the update. The updates of $\mathbf{x}$ and $\mathbf{C}$ follow a common principle. The mean $\mathbf{x}$ is updated such that the likelihood of successful offspring to be sampled again is maximized. The covariance matrix $\mathbf{C}$ is updated such that the likelihood of successful steps $(\mathbf{x}_k - \mathbf{x})/\sigma$ to appear again, or the likelihood to sample (in direction of) the path s_C, is increased. For further details about the updating of the covariance matrix $\mathbf{C}$ we can consult [83].

3.4 Downhill Simplex Method

Downhill Simplex method, also known as *Nelder-Mead* or *Amoeba* method, was proposed by John Nelder and Roger Mead in 1965 [91] and is a technique for minimizing an objective function in a many-dimensional space. The method uses the concept of a *simplex*, which is a special polytope of $n + 1$ vertices in n dimensions. The algorithm generates a new vertex by extrapolating the behavior of the objective function, measured at each test point arranged as a simplex, and applying geometric rules such as *reflection*, *expansion*, *contraction* and *reduction*. The algorithm then chooses to replace the worst of these vertices with the new test point and so the technique progresses.

Several applications of Downhill Simplex method can be consulted in the literature. Thus, Robin et al. [92] present an optimization of Electron-Beam Lithography step in the fabrication

Algorithm 3.2 Downhill Simplex algorithm

0: Given a polytope $\mathbf{x}_1, \cdots, \mathbf{x}_{n+1}$ **while** not happy
1: **Sort:** $f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_{n+1})$
2: $\mathbf{x}_0 = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$
3: $\mathbf{x}_r = \mathbf{x}_0 + \alpha(\mathbf{x}_0 - \mathbf{x}_{n+1})$ **//Reflection**
    if $f(\mathbf{x}_1) \leq f(\mathbf{x}_r) < f(\mathbf{x}_n)$ **then**
        $\mathbf{x}_{n+1} = \mathbf{x}_r$
        go to 0
4: **if** $f(\mathbf{x}_r) < f(\mathbf{x}_1)$ **then** //**Expansion**
    $\mathbf{x}_e = \mathbf{x}_0 + \gamma(\mathbf{x}_0 - \mathbf{x}_{n+1})$
    if $f(\mathbf{x}_e) < f(\mathbf{x}_r)$ **then**
        $\mathbf{x}_{n+1} = \mathbf{x}_e$
        go to 0
    else
        $\mathbf{x}_{n+1} = \mathbf{x}_r$
        go to 0
5: $\mathbf{x}_c = \mathbf{x}_0 + \rho(\mathbf{x}_0 - \mathbf{x}_{n+1})$ **//Contraction**
    if $f(\mathbf{x}_c) < f(\mathbf{x}_{n+1})$ **then**
        $\mathbf{x}_{n+1} = \mathbf{x}_c$
        go to 0
6: $\mathbf{x}_i = \mathbf{x}_1 + \sigma(\mathbf{x}_i - \mathbf{x}_1) \qquad \forall i \in \{2, \ldots, n+1\}$//**Reduction**
    go to 0

of microwave electronic circuits. A fast on-line optical signal-to-noise ratio measurement technique which uses the Downhill Simplex algorithm as a minimum finder is proposed and tested in Floridia and Moraes [93]. A benchmarking analysis is presented by Hansen [94]. Neto et al. [95] report a simple and fast method to estimate the fiber dispersion and laser chirp[3] parameters on a dispersive Intensity Modulation and Direct Detection (IM/DD) optical channel.

3.4.1 Algorithm description

Many variations exist depending on the actual nature of the problem being solved. Algorithm 3.2 describes a variation of the original algorithm. The parameters α, γ, ρ and σ are respectively the *reflection*, the *expansion*, the *contraction* and the *shrink* coefficients.

[3] A chirp is a signal in which the frequency increases or decreases with time.

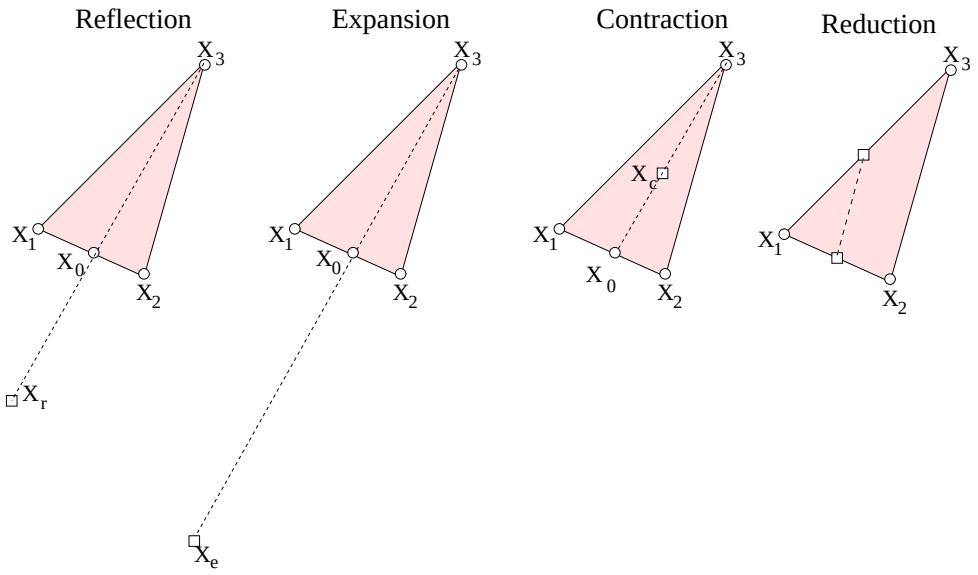

Figure 3.2: Geometric operations in a polytope performed by Downhill Simplex method.

Standard values are $\alpha = 1$, $\gamma = 2$, $\rho = -1/2$ and $\sigma = 1/2$. The initial simplex is important, indeed, a too small initial simplex can lead to a local search, consequently the algorithm can get more easily stuck. So this simplex should depend on the nature of the problem. Fig. 3.2 shows in a graphical way all the possible geometric operations to compute the new candidate node of the simplex in a simple two dimensional problem.

Chapter 4

Novel Methods for Solving DEs

This chapter describes the main contributions of the Thesis. Three novel algorithms to solve differential equations (DEs) are presented. All of them can be generically described by Fig. 4.1. As we see in the figure, the original problem of solving a DE is transformed into an optimization problem. This new problem is solved using an evolutionary algorithm. Depending on the approach, a local search can be applied on the best individual found by the evolutionary algorithm. Finally, the best individual at the end of the local search represent the solution to the original problem.

The methods are chronologically sorted. First a summary of the problem formulation is given in Section 4.1. Section 4.2 describes a new method to solve DEs based on GE. A baseline algorithm is first commented, and then several enhancements are sketched. Section 4.3 describes a different approach to solve DEs based on Fourier Series and ES. Finally, the last main contribution to this thesis is provided in section 4.4, where DEs are solved using a new method based on Gaussian kernels and CMA-ES.

4.1 Problem Statement: Summary

The mathematical problem statement was given in the Chapter 1 of this document. For the sake of clarity, following a brief summary is provided. The original problem to be solved can

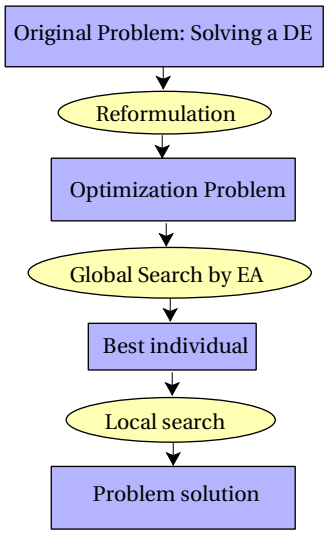

Figure 4.1: Block diagram of the generic proposed algorithms. The global search of the differential equation (DE) solution is made using an evolutionary algorithm (EA).

be formulated according to expression (1.1) subject to boundary conditions given by (1.2). Depending on the dimensionality of the independent and dependent variable space, ordinary differential equation (ODE), a system of differential equations (SODE) or partial differential equation (PDE) can be formulated. ODEs can be linear can be linear (LODE) or non linear (NLODE).

The solution satisfying (1.1) and (1.2) can be computed solving a *Constrained Optimization Problem* given by (1.3). The problem is then discretized using a set of n_C collocation points situated within the domain and as well n_B points on the boundary. Finally the original problem is transformed into a *Free Constrained Optimization Problem* defining a cost function according to (1.6).

4.2 Solving DEs with Grammatical Evolution: DESGE Algorithm

This part of the thesis describes the first contribution to the field of solving DEs with Evolutionary Computing. An algorithm based on Grammatical Evolution (GE) is presented. Al-

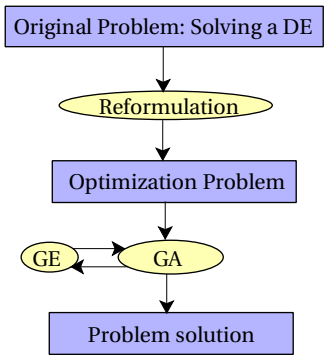

Figure 4.2: Block diagram of the proposed algorithm based on Grammatical Evolution. The global search of the DE solution is made using an genetic algorithm, which drives a grammatical evolution (GE) engine.

though other works in the literature solve DE using GE [1, 2], we adopt the same approach in this fist contribution because symbolic mathematical expressions are the more direct approach to express the candidate solutions. GE gives a framework to handle these symbolic expressions in a natural way.

As it is shown in Fig. 4.2, a Genetic Algorithm (GA) based on Grammatical Evolution is employed. Opposite to Fig. 4.1, no local search is used. Candidate solutions are expressed as generic mathematical expressions. To check if that expressions are solution of the ED, a symbolic mathematical engine is needed. Thus, Section 4.2.1 described the particular engine selected in the implementation. Then, Section 4.2.2 gives a complete description of the baseline algorithm. Finally Section 4.2.3 describes several enhancements implemented trying to improve the quality of the results and the convergence of the global algorithm.

4.2.1 GiNaC: A Symbolic Mathematical Engine

GiNaC [96] is a free computer algebra system released under the GNU General Public License. The name is a recursive acronym for "GiNaC is Not a CAS" (Computer Algebra System). What distinguishes GiNaC from most other computer algebra systems is that it does not provide a high-level interface for user interaction. Rather, it encourages its users to write symbolic algorithms directly in C++, which is GiNaC's implementation programming

language. Algebraic syntax is achieved in C++ through the use of operator overloading.

GiNaC uses the CLN library [97] for implementing arbitrary-precision arithmetic. Symbolically, it can do multivariate polynomial arithmetic, factor polynomials, compute GCDs (greatest common divisors), expand series, and compute with matrices. It is equipped to handle certain non-commutative algebras which are extensively used in theoretical high energy physics: Clifford algebras, Lie algebras, and Lorentz tensors.

All the needed features needed for developing a DE solver based on GP has been checked:

- Create a symbolic expression from a string of characters (decoding).
- Symbolic differentiation.
- Expression substitution.
- Symbolic evaluation (exact solution).
- Numerical evaluation.
- Create a string of characters from a symbolic expression (encoding).

All these operations have been checked executing a C++ program included in the Appendix B.

4.2.2 Algorithm Description

Following lines describe the algorithm implemented to solve a generic DE based on Grammatical Evolution paradigm. An example of a configuration file for the program can be consulted in Appendix C.

Grammar

The grammar can be configured easily with the input file, depending on the problem to solve. A grammar $\{N, T, P, S\}$ is composed by 4 elements as it was described in Section 3.1: nonterminals N, terminals T, product rules P and a start symbol S. Following the grammar used to solve differential equations is provided:

```
N={<exp>,<op>,<func>,<con>,<var>,<ephcon>}
T={+,-,*,^,(,),2.3323, ... ,sin,cos,...}
S=<exp>

P:
<exp>::= (<exp><op><exp>) //Parenthesis needed for ^
       | (<exp>)
       | <var>
       | <con>            //Optional
       | <fun>(<exp>)     //Optional
       | <ephcon>         //Optional

<op> ::= + | - | * | / | ^

<fun>::= sin | cos | tan | atan | exp | log | ...

<var>::= x | z | y | ...
```

Parenthesis in the first production rule of `<exp>` is needed because `^` symbol in GiNaC can't work with expression such as `4^x^3`. The number of functions, constants, operators and variables are defined in the input file.

In the grammar, the symbol `<ephcon>` represents a *Ephemeral constants,* which is a numerical-constant generation method. It can be seen as a translation of the classical genetic programming's ephemeral random constant to the grammatical evolution framework. Ephemeral constants have been implemented according to Nicolau and Dempsey [62]. In this approach data which are needed for constant creation, are stored in the genotype. Therefore the numerical constants can evolve with the algorithm. User can choose the number of codons n and the range $[a, b]$. Then, the ephemeral constant is computed as

$$a + \frac{\sum_{i=0}^{n-1} c_i 256^i}{255 \sum_{i=0}^{n-1} 256^i} (b - a)$$

where c_i is the integer number codified by codon i, which goes from 0 to 255 (each codon is codified by an `unsigned char` C type, which is 1 byte size). Ephemeral constants are transformed into character strings using $2n$ digits. In this way, the accuracy of the expression obtained in the decoding process depends on the number of bytes of the ephemeral constants. Nevertheless, the symbolic library GiNaC represents internally the constant with 20 digits

so it has not sense to use more than 10 bytes for each ephemeral constant. However it is straightforward to force GiNaC to work with a higher accuracy.

Fitness Function

The fitness function is computed adding errors at the collocation and boundary points using the original simple fitness function (1.6), but slightly modified to:

$$F(\mathbf{y}) = \frac{1}{m \cdot (n_C + n_B)} \left[\sum_{i=1}^{n_C} \left\| \mathbf{L}\mathbf{y}\left(\mathbf{x}_i\right) - \mathbf{f}\left(\mathbf{x}_i\right) \right\|^2 + \varphi \sum_{j=1}^{n_B} \left\| \mathbf{B}\mathbf{y}\left(\mathbf{x}_j\right) - \mathbf{g}\left(\mathbf{x}_j\right) \right\|^2 \right]. \tag{4.1}$$

Note that the cost function is obtained dividing the residuals by the total number of collocation points $m \cdot (n_C + n_B)$ in a similar way as in [14]. Other authors [12, 3, 27] do not make this normalization, which makes their values more dependent on the number of collocation points. The penalty parameter φ controls the relative weights assigned to boundary condition points compared with the inner collocation points (normally $n_C \gg n_B$). As it has been commented previously, derivatives and the numerical evaluation of the expressions at the collocation points are performed using GiNaC library.

Selection, Crossover and Mutation

Parent selection is performed using tournament method. The number of parents can be selected in the input file.

Two crossover operators have been implemented: *standard one point crossover* and *LHS (Left Hand Side) crossover* according to [59]. The last one is a structure preserving operator and needs for each individual to store some extra information created during the decoding process. The first crossover point in the first parent is selected randomly. The first crossover point in the second parent is constrained to that of a codon that expands (or it is associated with) the same type of non-terminal as the codon following the crossover point of the first parent. The second crossover point in each parent is selected so that the codon sequence in between the two crossover points fully expands the non-terminal designated by the first

crossover point.

These two operators are applied according to the following strategy:

- When the maximum wrapper value is 1, LHS crossover is used if both parents are feasible and it exists a correct first point crossover in the second parent.
- Otherwise standard crossover is employed. If the maximum wrapper is greater than 1, always standard crossover is used.

For the sake of clarity, an example of a standard crossover is shown (parents⇒children).

$$\begin{array}{ll} P1:\ (x * C2) & Ch1:\ (x * C2) \\ & \Rightarrow \\ P2:\ (cos(sin((((log(C4) + x)/(C0)))))) & Ch2:\ (cos(sin(((C8))))) \end{array}$$

It is important to notice that this and the following examples are shown at phenotype level, meanwhile the operator is performed at genotype level. Observe that in the second child appear a constant $C8$ which is not in any parent. Even more, it is possible to obtain one or both infeasible children using two feasible parents. An example of LHS crossover could be

$$\begin{array}{ll} P1:\ (exp(sin(exp(C8))) + x) & Ch1:\ (C3 + x) \\ & \Rightarrow \\ P2:\ sin(C3) & Ch2:\ sin(exp(sin(exp(C8)))) \end{array}$$

We can observed that the parent structures have been preserved. Parents have interchanged sub-expressions $exp(sin(exp(C8)))$ and $C3$.

Once a new population is created, mutation operator is applied in some individuals. Two different mutation operators are designed. One of them modifies the codons adding a random variable with uniform distribution between -26 and 26. These limits are a 10% of the maximum integer number codified by a codon (8 bits). The second mutation operator interchanges codons inside the same individual. Each mutation operator is applied with certain probability defined in the control file.

To fix ideas, following the source code of the two mutation operators is provided:

```
//Individual
void GODChromosome::MutateIntra(double Pmut)
{
  for(unsigned int e=0; e<codon.size(); e++)
  for(unsigned int c=0; c<codon[e].size(); c++)
    if(RandomDouble() <= Pmut)
    {
      if(RandomBit()) codon[e][c] += (unsigned char)RandomInt(26); //+10%
      else            codon[e][c] += (unsigned char)(256-RandomInt(26)); //-10%
    }
  fitnessUpdated = false;
}

void GODChromosome::MutateExtra(double Pmut)
{
  if(codon.size()<2) return;
  if(RandomDouble() > Pmut) return;

  vector<unsigned char> aux;
  unsigned int i = RandomInt(codon.size());
  unsigned int j = RandomInt(codon.size());
  aux = codon[i];
  codon[i] = codon[j];
  codon[j] = aux;
  fitnessUpdated = false;
}

//Population
void GODPopulation::MutateIntra(double pMutation)
{
  for(unsigned int i=0; i<chromosome.size(); i++)
    chromosome[i].MutateIntra(pMutation);
}

void GODPopulation::MutateExtra(double pMutation)
{
  for(unsigned int i=0; i<chromosome.size(); i++)
    chromosome[i].MutateExtra(pMutation);
}

//Main loop
for(generation=0; generation<evaluator.generations; generation++)
{
  //Dump
  // ...

  //Stop criteria
  // ...
```

```
  //Store Best individuals
  // ...

  //Cross operator. Population is considered even
  for(unsigned int cross=0; cross<population.chromosome.size()/2; cross++)
    population.CrossChromosomes(evaluator.tournament,
                                evaluator.pRecombination,
                                childs.chromosome[2*cross  ],
                                childs.chromosome[2*cross+1]);
  //Generation Model
  population = childs;

  //Mutate Intra
  population.MutateIntra(evaluator.pMutationIntra);

  //Mutate Extra
  population.MutateExtra(evaluator.pMutationExtra);

  //Elitism: substitute worst individuals by the bests in previus generation
  // ...
}
```

A generational model is employed, i. e., the old population is substituted by the population generated with the children individuals. Elitism of one or two individual is used.

4.2.3 Enhancing DESGE Algorithm

Several enhancements to the baseline algorithm have been implemented and tested.

Ephemeral Local Search

An *Ephemeral Local Search* is done over an individual choosing one random ephemeral constant in his phenotype. A Newton-Raphson method is performed modifying the ephemeral constant. The number of iterations of this local search can be specified in the input file. If the new individual has a better fitness than the original one, the original individual is swapped with the new one.

Local Search is applied only over feasible individuals. If an individual codes a set of expressions (for system of equations problems), local search is applied over each equation.

Following the details of the Newton-Raphson method are explained. If the individual has several ephemeral constants, one of them, c for instance, is randomly chosen. Calling $F(c)$

the fitness of this individual, the Newton-Raphson algorithm is as follows:

$$c_{i+1} = c_i - 0.8\frac{F(c_i)}{F(c_i) - F(c_{i-1})}\left(c_i - c_{i-1}\right),$$

where c_{i+1} is the new ephemeral constant at step $i+1$ computed with previous values c_i and c_{i-1}. Note that a relaxation factor of 0.8 is used. This relaxation improves the performance of the Newton-Raphson algorithm. If c_0 is the initial value of the ephemeral constant at the individual before doing the local search, the next constant is obtained as $c_1 = c_0 + \triangle$ always fulfilling that $c_1 \in [a, b]$. The increment $\triangle$ is computed as $\triangle = \zeta\frac{b-a}{100}$ where ζ is a random variable of uniform distribution between -1 and 1. The Newton-Raphson is repeated until a predefined maximum number of iterations is achieved. The constant c is only changed if during the process a lower fitness value has been detected. There are other two stop conditions: $|c_i - c_{i-1}| < 10^{-2B}$ and $|F(c_i) - F(c_{i-1})| < 10^{-2B}$, where B is the ephemeral constant number of bytes.

Ephemeral Constant Local Search is done only over feasible individuals in the population with a probability p_{eph} for each individual. If an individual has several ephemeral constants, one is randomly chosen. Ephemeral Local Search is always performed in each generation over the best individual in the population ($p_{eph} = 1$ for that individual).

A more advance local search has been investigated: Newton-Raphson using derivatives. In the ephemeral local search described above, a complete fitness evaluation was required in each sub-step of the Newton-Raphson algorithm. Furthermore, the fitness function derivative was made numerically, so at least two fitness evaluation was needed for estimating the derivative. For the sake of simplicity, ordinary differential equation with dependent variable y and independent variable x is considered, although the method could be applied for any kind of differential equation problem. Let's consider that local search is done over and individual with a phenotype given by an expression g, that is, a possible solution has the shape $y = g(x)$. The idea is to choose one ephemeral constant in the phenotype and substitute its numerical value by a symbol κ. In this way, using (4.26), the fitness of the phenotype will be

a function instead of a numerical value

$$F\left(g\left(\kappa\right)\right) \equiv F\left(\kappa\right)$$

Defining the derivative of the fitness function as

$$F'\left(\kappa\right) = \frac{\partial F}{\partial \kappa}$$

the Newton-Raphson can be computed with the following algorithm

$$\kappa_{n+1} = \kappa_n - 0.8\frac{F\left(\kappa_n\right)}{F'\left(\kappa_n\right)} \tag{4.2}$$

Eq. (4.2) is thought for finding the root of the fitness function. Due to definition of the fitness function, it never has negative values, so equation (4.2) is numerically unstable. A possible solution is to find the minimum value of the symbolic fitness function. This is the same than finding the roots of the first derivative. Therefore a possible new approach is to modified the Newton-Raphson algorithm using the second derivative $F''\left(\kappa\right) = \partial^2 F/\partial \kappa^2$:

$$\kappa_{n+1} = \kappa_n - 0.8\frac{F'\left(\kappa_n\right)}{F''\left(\kappa_n\right)} \tag{4.3}$$

This new iterative scheme is numerically more stable, less steps are needed for the convergence and better accuracy is obtained in the solution. Nevertheless there are two drawbacks: Second symbolic derivative must be computed, which add a little overhead; and the iterative scheme could converge into a local maximum instead of a local minimum.

The final algorithm for ephemeral constant local search is a trade-off between both approaches:

- The local search is done over the best individual of the population. In the other individuals, the local search is done with a certain probability $p_{LocalSearch}$, typically 0.5.

- Newton-Raphson is applied using equation (4.2) or (4.3) with a probability of 0.5. In

this way, equation (4.2) gives good convergence, and equation (4.3) allows to scape from local maximum and explore other possible minimums.

- A predefined Newton-Raphson number of steps are performed, typically 20. This steps could be lower if the stop criteria is fulfilled, i.e, $|\kappa_{n+1} - \kappa_n| < 10^{-20}$ or $|F'(\kappa_n)| < 10^{-20}$. If second derivative is used, the last stop criteria is changed to $|F''(\kappa_n)| < 10^{-20}$. All the Newton-Raphson steps do not required a full fitness evaluation, only the symbolic fitness function must be computed once at the beginning of the algorithm.
- The final value of the constant κ is only modified if a better fitness is found.

Gaussian local search

A new concept for improving the solver performance has been checked. For that, a new set of defined functions (*Gaussians*) can be used in the search process. The Gaussian functions are coded in the genotype in a optimized way. For problems of just one independent variable x, the Gaussian will have 4 ephemerals constant as parameters:

$$G(x; c_1, c_2, c_3, c_4) = (c_1 + c_2(x - c_4)) \exp\left(-c_3(x - c_4)^2\right)$$

If the range of the constant includes negative values, c_3 is changed with its square c_3^2. The target of these functions is to control in the point c_4 the value of the dependent variable of the differential equation problem modifying c_1, and the derivatives of the candidate solutions by means of c_2 parameter. With c_3 we control the width of the "bell".

Note that c_4 appears twice in the phenotype. In the genotype only is coded once, so any operator applied over a Gaussian which modifies this constant will be applied in a preserving way. For cases of more independent variables, the Gaussian functions are defined conveniently. For instance, for two independent variables x and y we have

$$G(x, y; c_1, c_2, c_3, c_4, c_5, c_6, c_7) = (c_1 + c_2(x - c_6) + c_3(y - c_7)) \exp\left(-c_4(x - c_6)^2 - c_5(y - c_7)^2\right)$$

A Gaussian Local Search operator over an individual consist of adding a Gaussian to the

individual and performing ephemeral local searches over all the Gaussian ephemeral constants except those which place the Gaussian in the domain (c_4 in the previous one dimensional example). If the new individual has a better fitness, it is swapped by the original one. This operation is performed only over feasible individuals. The ephemeral constant are initialized to $(c_{max} - c_{min})/100$ except the constant which place the Gaussian, where c_{max} and c_{min} are the maximum and minimum collocation points in the domain. The Gaussian is placed at the collocation point where the residual of the main equations or the boundary conditions are higher. For that reason it is mandatory that all the collocation points are inside the range of ephemeral constants.

Before adding a Gaussian, it is checked if there is a previous Gaussian in the genotype of the individual at the same collocation point. If so, an ephemeral constant local search is performed without adding a Gaussian. In this way, we try to minimize bloat. However Gaussian local search is more expensive that ephemeral local search and can provoke bloat. Therefore Gaussian local search is only used when any fitness variation of the best individual is detected along some number of generations controlled with the parameter `MAX_GENERATION_INERTIA`. As for Ephemeral Local Search, all individuals are selected with a probability of p_{eph}, except the best individual which is always selected.

Addition Crossover

A new crossover operator was implemented. Given two parents with phenotypes f_1 and f_2 the two children will be

$$Ch1: \ \alpha f_1 + (1-\alpha) f_2$$

$$Ch2: \ (1-\alpha) f_1 + \alpha f_2$$

being α a random ephemeral constant between 0 and 1. Addition crossover is only used if both parents are feasible. It is mandatory, if addition crossover is used, that 0 and 1 values are inside the range of the ephemeral constants. If there are several dependent variables, different α values will be used for each expression of the genotype.

A new parameter `ADDITION_CROSSOVER_RATIO` controls the ratio between LHS crossover and addition crossover. Each crossover type is applied over the whole population. With 0 only LHS crossover is used.

Grammar for real numbers

A specific grammar to code real numbers was tested using scientific notation with 3 digits in the exponent and a fixed number of digits in the mantissa (which can be selected by the user). An example grammar of 3 digits in the mantissa is shown:

```
<real> = <dig>.<dig><dig>e<dig><dig><dig>  |
          <dig>.<dig><dig>e-<dig><dig><dig> |
         -<dig>.<dig><dig>e<dig><dig><dig>  |
         -<dig>.<dig><dig>e-<dig><dig><dig>

<dig> = <GECodonValue_0_9>
```

The meaning of label `<GECodonValue_0_9>` is the same that in paper [62]. It allow us extract codon values and integrate them directly into the generated phenotype strings. During the mapping process, if this symbol is encountered, it is replaced with the numerical value of the current codon, and this codon is consumed. The *0* and *9* specifies a lower and upper limit for this value, so the codon value is mapped to the range specified, using the *mod* operator.

This grammar has several advantages: scientific notation allows to represent a high range of real numbers, so it is not necessary to specify limits; and the implementation is easy because the ephemeral constant size in the genotype is always known. Sometimes could be interesting to restrict the ephemeral constant range. For that, the user can specify a range limit. This is particular useful to avoid numerical problems with GiNaC library when high numbers appear, as it was commented in section 4.2.1. If the float value decoded from a chromosome is outside the feasible range, the actual value is set to the closer limit.

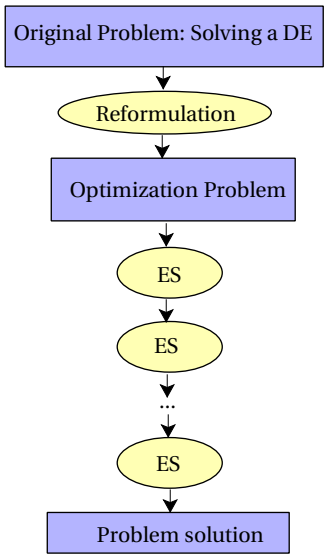

Figure 4.3: Block diagram of the proposed algorithm based on Evolution Strategies. Note that several steps are used sequentially.

4.3 Solving DEs with Evolution Strategies and Fourier Series: DESES Algorithm

In the present section, a different approach for solving differential equations is reported. Fig. 4.3 gives a general overview of the algorithm. Candidate solutions are expressed as partial sums of Fourier series. In order to simplify the problem, an even periodic expansion of the solutions is done in such a way that all the sine coefficients are vanished. This representation can be regarded equivalent to a Discrete Cosine Transform (DCT) [98] which has been successfully used in several science and engineering applications, as for lossy compression of audio (MP3) and image (JPEG). With the chosen solution representation, and as it was commented in the previous approach, the problem of solving differential equations is transformed into an optimization one. The differential equation residuals and the boundary condition errors are minimized. The optimal Fourier coefficients are tuning using Evolution Strategies (ESs). In order to facilitate the process, the harmonic searching is done in a progressive way starting with the lowest order harmonic and using a different ES cycle to find the optimum value for each one. This sequential process is represented in Fig. 4.3 with different ES steps.

Following the details of our approximation are exposed.

4.3.1 Representation of Candidate Solutions

In the proposed approach, each component $y(\mathbf{x})$ of the trial solution is expressed as a partial sum of a Fourier series. The periodic expansion of $y(\mathbf{x})$ from the original definition range to all $\mathbb{R}^d$ is always performed using even functions. Therefore all the sine Fourier coefficients are vanished. In order to define this expansion, first some notation must be introduced. For each coordinate x_k with $k = 1, \cdots, d$, variables $x_{k,min}$ and $x_{k,max}$ are defined as the minimum and maximum values among the inner collocation points C and the boundary condition points B. Using these values and an user defined parameter $\xi \geq 0$ called *range extension*, a new coordinate origin c_k and a semi-period L_k are defined as

$$c_k = x_{k,min} - \xi\left(x_{k,max} - x_{k,min}\right) \tag{4.4}$$

$$L_k = \left(x_{k,max} - x_{k,min}\right)(1 + 2\xi) \tag{4.5}$$

Then each component $y(\mathbf{x})$ of the solution vector is expressed as a partial sum of Fourier Cosine series:

$$y(x_1, \ldots, x_d) = \frac{a_0}{2} + \sum_{n_1,\ldots,n_d=1}^{N} a_{n_1,\ldots,n_d} \prod_{k=1}^{d} \cos\left(\frac{\pi n_k}{L_k}\left(x_k - c_k\right)\right) \tag{4.6}$$

where a_0 and $a_{n_1,\ldots,n_d}$ are the unknown coefficients or *harmonics* and N is an user parameter which determines the number of harmonics used. The total number of harmonics for each component will be $1+N^d$. By definition, this expanded function is periodic in each dimension with period $2L_k$ and, in addition, is defined everywhere, continuous and infinitely differentiable. However, according to Eq. (1.6), this function will be only evaluated in the original definition range. Therefore the expansion can be done anyhow with the following constraints: the expanded function must be periodic, even and solution to the original problem in Ω and $\partial\Omega$.

Range extension parameter ξ is needed in order to suppress the intrinsic limitations of

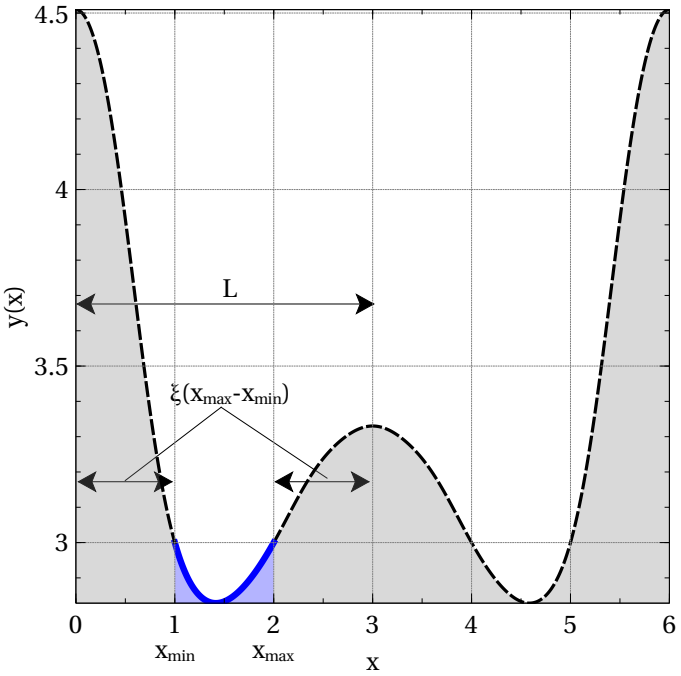

Figure 4.4: Even periodic expansion example of a function originally defined in the range $[1, 2]$.

even functions at the boundaries regarding the first partial derivative:

$$\left.\frac{\partial y\left(x_1, \ldots, x_d\right)}{\partial x_k}\right|_{x_k=c_k} = 0. \tag{4.7}$$

Note that if $\xi = 0$ the null first derivative will be obtained at points $x_k = x_{k,min}$, which could be interesting in some particular cases. Nevertheless, a general problem will have not-null first derivatives at boundaries. Because non discontinuities are introduced, neither in the expanded function itself nor in their derivatives, convergence problems regarding Gibbs phenomenon (large oscillations of the nth partial sum of the Fourier series near the discontinuity jump) are avoided. According to expression (4.6), Fig. 4.4 shows an even periodic expansion example of a function originally defined in the range $[1, 2]$, in one dimension ($d = 1$) with a *range expansion* $\xi = 1$. Note the null first derivatives at points $x = 0$, $x = 3$ and $x = 6$.

4.3.2 Fitness Function

The same fitness function than DESGE algorithm is emplyed, Eq. (4.1). Computing the cost function implies obtaining derivatives of expression (4.6). A generic derivative operator D can be expressed as

$$D = \frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}} \frac{\partial^{\lambda_2}}{\partial x_2^{\lambda_2}} \cdots \frac{\partial^{\lambda_d}}{\partial x_d^{\lambda_d}}. \tag{4.8}$$

Using Eq. (4.6), the differential operator D applied to the dependent variable y yields

$$D\left(y\right) = \sum_{n_1,\ldots,n_d=1}^{N} a_{n_1,\ldots,n_d} \prod_{k=1}^{d} \left(\frac{\pi n_k}{L_k}\right)^{\lambda_k} \cos^{(\lambda_k)}\left(\frac{\pi n_k}{L_k}\left(x_k - c_k\right)\right) \tag{4.9}$$

where the nth derivative of cosine function can be computed as

$$\cos^{(n)} x = \begin{cases} \cos x & if \quad n\%4 = 0 \\ -\sin x & if \quad n\%4 = 1 \\ -\cos x & if \quad n\%4 = 2 \\ \sin x & if \quad n\%4 = 3 \end{cases}. \tag{4.10}$$

4.3.3 New symbolic expression interpreter

The symbolic engine GiNaC [96] has been substituted by an in-house symbolic interpreter. Because the individual phenotype structure is known a priori, symbolic derivatives can be computed directly in the code using the expressions in (4.10). Therefore, the use of GiNaC library is not necessary. In fact, depending on the derivative equation complexity, GiNaC library makes the computation inefficient.

An in-house C++ class dealing with symbolic expressions has been developed. It consists of binary trees. Each node in the tree is a pointer to some object representing a mathematical function, an operator, a variable or a number. The leaves of the the tree are variables and

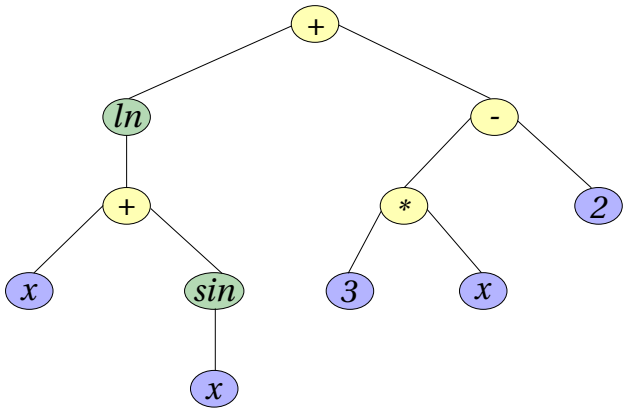

Figure 4.5: Tree representing the math expression $\ln(x + \sin x) + 3x - 2$

numbers. Recursive functions are used, allowing an easy implementation. The only drawback of this new approach is that the user must provide the expression in a special way for building the tree. Tree nodes are filled starting with the most left free positions. In future versions an interpreter could be used for making the input file more friendly.

For instance, the following expression

$$\ln(x + \sin x) + 3x - 2$$

is stored as the tree showed in Fig. 4.5. User must provide the tree nodes in the following order[1]:

$$+\ \ln\ +\ x\ \sin\ x\ -\ *\ 3\ x\ 2$$

Using this new approach has two main advantages. Firstly, it is not necessary to link with GiNaC and CLN libraries, simplifying the source code, the maintenance and reducing the executable size. Secondly, the performance has been increased. For instance, the CPU time needed for a run of NLODE2 problem (the benchmark problems will be described in Section 5) has been decreased 7 times, almost one order of magnitude.

[1]The same expression could be coded with different trees.

4.3.4 Algorithm Description

In this section a global strategy for solving the original problem is presented. The basic idea consists of introducing harmonics one by one in the evolutionary process starting with the lower order ones. This strategy is based on the assumption that the absolute values of Fourier coefficients decrease when the harmonic number is increased. There are several works that give bounds to the Fourier coefficients assuming some properties to the original function [99]. For instance, if we assume that y is an absolutely continuous function of one real variable, then the nth Fourier coefficient a_n fulfills

$$|a_n| \leq \frac{K}{n}, \tag{4.11}$$

where the constant K only depends on y but not on n. Better bounds can be given if more features are assumed on y. This property has been observed experimentally in all the test problems. In Fig. 4.6 the absolute value of the first ten Fourier coefficients $|a_n|$ are shown for LODE1 case (the test cases will be described in section 5.3). We can observe the decay of the absolute values when the harmonic number is increased. Note the logarithm scale on the vertical axis.

The global algorithm consists of several basic steps, called *ES steps*. Each *ES step* corresponds to a closed evolutionary strategy and is instantiated as it will be explained in next subsection. In addition, a global flag of three possible values (*active*, *inactive* and *frozen*), named $flag_{\mathbf{n}}$, is associated to each harmonic coefficient $a_{\mathbf{n}}$. Each $flag_{\mathbf{n}}$ is initialized before running an *ES step*, does not evolve and has the same value for all individuals in the population. If a harmonic is *active*, it participates for computing the fitness value of the individual, using Eq. (4.26), and is evolved by the *ES step*. If a harmonic is *inactive*, it is not used (zero value) for computing the fitness value and is not evolved. Finally, if the harmonic is *frozen*, it is used for computing the fitness value but is not evolved by the *ES step*. Representing each *ES step* by $ES\,[m_{low}, m_{high}]$, each harmonic in an individual is classified as follows:

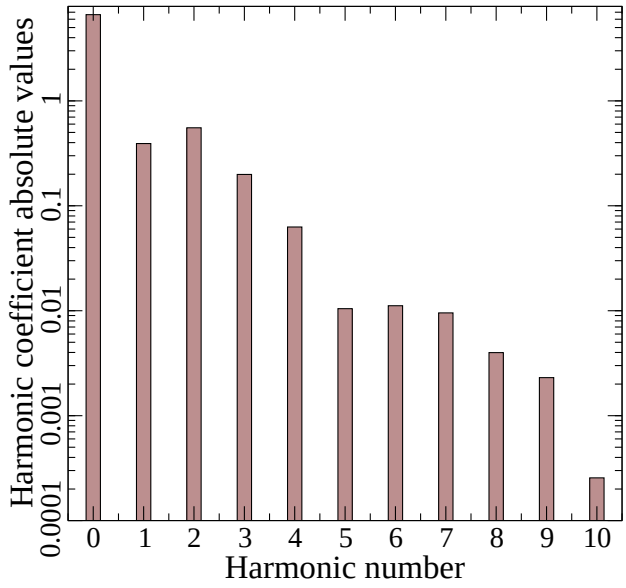

Figure 4.6: First 10 Fourier coefficients computed for LODE1 case.

- Harmonic $a_{n_1,\ldots,n_d}$ will be *active* if there is at least one index $n_i \in [m_{low}, m_{high}]$ and the remaining indexes are $n_j \leq m_{high}$.
- A harmonic will be considered *frozen* when for all indexes $n_j < m_{low}$.
- A harmonic will be considered *inactive* if there is at least one index $n_j > m_{high}$.

Using the aforementioned notation, the global strategy can be implemented by a sequence of *ES steps* represented by the algorithm shown in Fig. 4.7.

Note that there is always an *ES step* of type $ES\,[0, m]$ after of an *ES step* of type $ES\,[m, m]$. As can be seen easily, a new harmonic is tuned (*activated*) in an *ES step* of type $ES\,[m, m]$. Then, all the harmonics lower or equal than the new one are tuned in a step of type $ES\,[0, m]$. With this policy the search space dimension is reduced making the searching process more systematic and the optimization problem easier. The steps $ES\,[0, 0]$ and $ES\,[1, 1]$ are not considered because the algorithm starts tuning the two first harmonic simultaneously ($ES\,[0, 1]$). The final step is called *Fine Tuning* phase and its aim is to adjust finely all the harmonics simultaneously, so all of them are activated. The stop criterion for

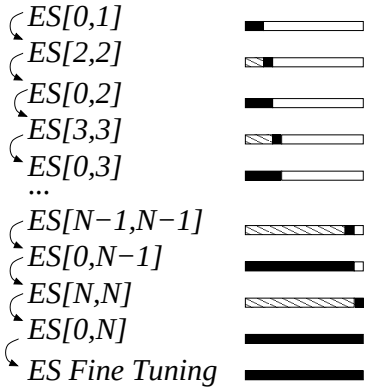

Figure 4.7: Global strategy as a sequence of *ES steps*. Each bar represents, going from left to right, the *frozen* harmonics (pattern filled), the *active* (black color) and the *inactive* ones (white color).

each *ES step* is fulfilled when the best fitness during a predefined number of consecutive generations is not modified within a given tolerance or when a predefined maximum number of generations G are fulfilled. These three parameters (number of generations, tolerance and G) are input parameter for the algorithm.

Before running each *ES step*, the population must be initialized. The initialization policy is different for *ES steps* of types $ES\,[m,m]$, $ES\,[0,m]$ or the *Fine Tuning* phase. The population initialization in each *ES step* of type $ES\,[m,m]$ can be summarized as follows:

$$a_{\mathbf{n}} = \begin{cases} U\,(\alpha,\beta) & if \quad flag_{\mathbf{n}} = active \\ \hat{a}_{\mathbf{n}} & if \quad flag_{\mathbf{n}} = frozen \\ 0 & if \quad flag_{\mathbf{n}} = inactive \end{cases}, \tag{4.12}$$

where $U\,(\alpha,\beta)$ is a random sample from a continuous uniform distribution in the range $[\alpha,\beta]$, being α and β user defined parameters. The symbol $\hat{a}_{\mathbf{n}}$ denotes the harmonic with index $\mathbf{n}$ of the best individual at the final population of the previous *ES step* run. This policy avoids the algorithm being trapped in local optima when a new harmonic is used. On the other hand, it is not performed any particular initialization in steps of type $ES\,[0,m]$ or *Fine Tuning* steps,

that is, the initial population is copied from the final population of the previous *ES step*.

Mutation strengths are initialized in a similar way. For steps of type $ES\,[m,m]$ it is used the next expression:

$$\sigma_{\mathbf{n}} = \begin{cases} U\,(\gamma,\delta) & if \quad flag_{\mathbf{n}} = active \\ \\ 0 & if \quad flag_{\mathbf{n}} = inactive \quad or \quad frozen \end{cases}, \tag{4.13}$$

where the range of the random variable $U\,(\gamma,\delta)$ is as well a user defined parameter. Initialization for $ES\,[0,m]$ is done in a range ten times lower than in expression (4.13) in order to re-adjust lower harmonics when the new harmonic has been computed:

$$\sigma_{\mathbf{n}} = \begin{cases} U\,(\gamma,\delta)\,/10 & if \quad flag_{\mathbf{n}} = active \\ \\ 0 & if \quad flag_{\mathbf{n}} = inactive \quad or \quad frozen \end{cases}. \tag{4.14}$$

Finally, all the mutation strengths in *Fine Tuning* phase are initialized to a small value (10^{-7} in the experiments).

In PDEs, an increment of the harmonic order implies a non linear increment in the number of harmonics depending on the problem dimension. For instance, for a two dimensional problem, going from $ES\,[0,3]$ to $ES\,[0,4]$ implies an increment in the number of harmonics of 7, as we can observe in Eq. (4.15):

$$\begin{bmatrix} & & \\ a_{11} & a_{12} & a_{13} \\ & & \\ a_{21} & a_{22} & a_{23} \\ & & \\ a_{31} & a_{32} & a_{33} \\ & & \end{bmatrix} \Longrightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & & & \\ a_{21} & a_{22} & a_{23} & a_{24} \\ & & & \\ a_{31} & a_{32} & a_{33} & a_{34} \\ & & & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}. \tag{4.15}$$

Therefore, in PDE problems, the maximum number of new coefficients to tune in each $ES\,[m,m]$ phase could be specified as another algorithm input. In this way, this type of phases would be split into several sub-phases.

Evolution Strategy

The optimization problem of searching the best set of harmonic coefficients is solved using a ES which is an optimization technique based on ideas of adaptation and evolution [71]. Among all the Evolutionary Computing paradigms, ES has been chosen because they are typically used for continuous parameter optimization problems and its very useful feature: self-adaptation of strategy parameters [81]. There is a strong emphasis on mutation for creating offspring. In this approach uncorrelated mutation with several step sizes is used. Regarding the genotype coding, each component of the solution vector is represented by

$$\left[\underbrace{a_0, a_{1,\cdots,1}, \cdots, a_{N,\cdots,N}}_{\mathbf{a}}, \underbrace{\sigma_0, \sigma_{1,\cdots,1}, \cdots, \sigma_{N,\cdots,N}}_{\boldsymbol{\sigma}}\right]. \tag{4.16}$$

In the case of a system of equations, as many vectors as Eq. (4.16) will be handle as dependant variables. The first part of the genotype, $\mathbf{a}$, codifies all the harmonic coefficients needed for building the individual's phenotype using Eq. (4.6). For each individual, a fitness value can be computed using Eq. (4.26). The second part of the vector, $\boldsymbol{\sigma}$, codifies the mutation strengths for each harmonic. In system of equations, as many coefficients and sigmas will be handle as number of dependent solution.

Within one ES generational cycle, λ offspring individuals are generated from a set of μ parent individuals using recombination and mutation operators. Then selection operator chooses those individuals which will form the population in the next generation. The process will be repeated in a close loop until some stop condition is fulfilled.

Global discrete recombination is used for the harmonics, and global intermediate recom-

bination is performed for the mutation strengths. That is,

$$\left.\begin{array}{l} a_{\mathbf{n}}^{child} = a_{\mathbf{n}}^{parent1} \quad or \quad a_{\mathbf{n}}^{parent2} \\ \\ \sigma_{\mathbf{n}}^{child} = \left(\sigma_{\mathbf{n}}^{parent1} + \sigma_{\mathbf{n}}^{parent2}\right)/2 \end{array}\right\} \tag{4.17}$$

where $\mathbf{n} \equiv n_1, \cdots, n_d$ is an index vector according to expression (4.6) and the parents are chosen randomly among all the population. This scheme of recombination is the most used in ES implementations because preserves diversity within the phenotype space, allowing the trial of very different combinations of values, whilst the averaging effect of intermediate recombination assures a more cautious adaptation of mutation strengths [81].

Each λ offspring individual is mutated using independent random samples from a standard normal distribution $N(0,1)$

$$\sigma'_{\mathbf{n}} = \sigma_{\mathbf{n}} \exp\left(\tau' N(0,1) + \tau N_i(0,1)\right) \tag{4.18}$$

$$a'_{\mathbf{n}} = a_{\mathbf{n}} + \sigma'_{\mathbf{n}} N_i(0,1) \tag{4.19}$$

where τ and τ' are the learning rates defined as

$$\left.\begin{array}{l} \tau' = f_\tau / \sqrt{2m\left(1+N^d\right)} \\ \\ \tau = f_\tau / \sqrt{2m\sqrt{1+N^d}} \end{array}\right\},$$

being f_τ a learning rate factor defined by the user and $1+N^d$ the total number of harmonics for each component, and m the number of dependent variables or equations. In order to

avoid too small mutation steps, a threshold ϵ is applied in the following way:

$$if\ \sigma'_{\mathbf{n}} < \epsilon \Longrightarrow \sigma'_{\mathbf{n}} = \epsilon \tag{4.20}$$

The threshold used in the present work is $\epsilon = 10^{-20}$. In order to guarantee extinction of misfit individuals, the classical selection process (μ, λ) is used. However, it is modified adding elitism of one individual. That is, the next generation is formed by the best μ individuals among the λ mutated offspring and the best individual among all the parents. In this way, the fitness of the best individual in the population is a monotonic function with the generation number. Two stop conditions are checked: maximum number of generations or unchanged fitness of the best individual during a predefined number of generations. The tolerance used in the present work to distinguish unchanged fitness values is 10^{-9}.

4.3.5 Separation of Variables

As we will see in Chapter 5.3, solving PDEs are more difficult compared with ODEs because the number of unknowns to be tuned is bigger (for example, from 11 unknowns in ODEs we move to 101 in PDEs). Two strategies based on the separation of variables of PDEs have been tested: separation of variables is a method for solving ordinary and partial differential equations, in which algebra allows us to rewrite an equation so that each of two variables occurs on a different side of the equation[2]. The first strategy consist of reducing the number of unknowns from 101 to just 21 changing the expression (4.6) by

$$y\left(x_1, \ldots, x_{D-1}\right) = \frac{a_0}{2} + \sum_{n_0,\ldots,n_{D-1}=1}^{N} \prod_{i=0}^{D-1} a_{i,n_i} \cos\left(2\pi \sum_{k=1}^{D} \frac{n_k}{L_k}\left(x_k - B_k\right)\right). \tag{4.21}$$

As we see the original unknowns $a_{n_0,\cdots,n_{D-1}}$ have been substituted by a_{i,n_i}. The accuracy is decreased, but the search space is reduced facilitating the optimization process.

The other separation of variables approach consists of using all the possible harmonics $a_{n_0,\cdots,n_{D-1}}$, but when a new harmonic n is taking into account, instead of performing several

[2]http://en.wikipedia.org/wiki/Separation_of_variables

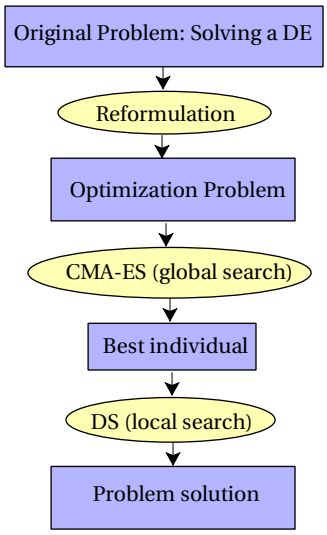

Figure 4.8: Block diagram of the proposed algorithm. The global search of the Differential Equation (DE) solution is made using Covariance Matrix Adaptation Evolution Strategies (CMA-ES) and the local search by mean of a Downhill Simplex (DS) algorithm.

steps for using the new unknowns gradually, a separation of variables simplification is used only in this step. Only $a_{n\cdots n}$ is searched, meanwhile the rest of new unknowns are computed with the following expression:

$$a_{n_0\cdots n_{D-1}} = \prod_{i=0}^{D-1} a_{n_i\cdots n_i}.$$

Both strategies have been tested with several PDEs. Unfortunately, results obtained are worse than those obtained with the method without separation of variables.

4.4 Solving DEs with CMA-ES and Gaussian Functions: DESCMA-ES Algorithm

In this section an heuristic method based on *Covariance Matrix Adaptation Evolution Strategies* (CMA-ES) for solving differential equation is presented. Candidate solutions are expressed as a weighted sum of *Gaussian* functions, which can be considered a special type of radial basis functions. To increase the accuracy of the method, a local search is applied to

the best solution found by the CMA-ES using the *Downhill Simplex* (DS) algorithm.

A block diagram of the algorithm is showed in Fig. 4.8. As we see, the original problem of solving a particular DE is reformulated as a new optimization problem. The solution is obtained making a global search by mean of an evolutionary algorithm (CMA-ES). On the best individual obtained in such search, a local search is performed employing a Downhill Simplex (DS) algorithm.

Following it is described the individual representation and the fitness function used in this new approach. These elements will define the new algorithm implemented.

4.4.1 Representation of Candidate Solutions

In the previous approaches, the candidate solutions were expressed using general symbolic mathematical expressions (DESGE algorithm) and Fourier series (DESES algorithm). The first approach make the optimization problem very large, because it is not known a priori how the solution will be built using the terminals of the grammar. In the second approach, the search space is reduced using a basis function. However, the algorithm is not efficient when the problem dimension is lager than one.

In order to facilitate the search, it has been used a different functional basis to express the candidate solutions: *radial basis* functions have been adopted. A radial function is a real-valued function whose value depends only on the distance from the origin, so that $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$; or alternatively on the distance from some other point $\mathbf{c}$, called a *center*, so that $\phi(\mathbf{x}, \mathbf{c}) = \phi(\|\mathbf{x} - \mathbf{c}\|)$. The norm is usually the Euclidean distance, although other distance functions are also possible.

Among the family of radial basis function, Gaussian functions are chosen because its good behavior to approximate any continuous function [100]. A Gaussian kernel is defined as

$$\Phi(\mathbf{x}, \mathbf{c}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{c}\|^2\right), \tag{4.22}$$

where $\mathbf{c} \in \mathbb{R}^d$ is a vector defining the center of the Gaussian, and $\gamma > 0$ is a scalar which control the dispersion of the distribution. A candidate solution $y(\mathbf{x})$ is then expressed combining

n Gaussian kernels in the following way:

$$y(\mathbf{x}) = \sum_{i=1}^{n} w_i \Phi(\mathbf{x}, \mathbf{c}_i) = \sum_{i=1}^{n} w_i \exp\left[-\sum_{j=1}^{d} \gamma_i (x_j - c_{ij})^2\right]. \tag{4.23}$$

For each Gaussian we have $d+2$ degrees of freedom: the d components of the centers $\mathbf{c}_i$, the dispersion parameter γ_i and the weight w_i. This problem can be seen as an optimization problem with $n(d+2)$ degrees of freedom. If the function to be estimated is a vector function with m dimensions in the output domain, we apply the previous ideas to all of m dimensions. Therefore we obtain the following degrees of freedom or *number of unknowns* (variables):

$$N = m \cdot n \cdot (d+2). \tag{4.24}$$

For instance, in the case of an ordinary DE ($m = d = 1$), each genotype would be represented by

$$[w_1, \gamma_1, c_1, \cdots, w_n, \gamma_n, c_n]. \tag{4.25}$$

Finally, the original problem has been transformed into a *Free Constrained Optimization Problem*. Therefore, non-constrains in the variables are considered. In particular, γ values are not limited to positive values. Although standard Gaussian kernels have positive γ values, no constrains are applied over gammas because it has been observed experimentally better convergence if $\gamma \in \mathbb{R}$. For example, for $\gamma = 0$ a constant function can be approximated with only one kernel. This has the disadvantage that these non-standard Gaussian kernels are not bounded when $\gamma < 0$, that is, $\lim_{x\to\infty} \Phi(x, c) = \infty$. However, we are only interested in solving differential equations in bounded domains.

4.4.2 Fitness Function

In the present approach, the fitness function is computed with a modification of Eq. (4.1) as follows:

$$F(\mathbf{y}) = \frac{1}{m \cdot (n_C + n_B)} \left[\sum_{i=1}^{n_C} \xi(\mathbf{x}_i) \left\|\mathbf{L}\mathbf{y}(\mathbf{x}_i) - \mathbf{f}(\mathbf{x}_i)\right\|^2 + \varphi \sum_{j=1}^{n_B} \left\|\mathbf{B}\mathbf{y}(\mathbf{x}_j) - \mathbf{g}(\mathbf{x}_j)\right\|^2\right], \tag{4.26}$$

where a weighting factor $\xi(\mathbf{x}) \in \mathbb{R}^+$ only dependent on the collocation points $\mathbf{x}_i$ is introduced. The weighting factor $\xi(\mathbf{x})$ can be used to modify the algorithm convergence behavior increasing the relative errors in some domain locations. In the present approach, it is used to increase the weights of the inner collocation points closer to boundary points in the following way:

$$\xi(\mathbf{x}_i) = \frac{1+\kappa\left(1-\frac{\min_{\forall \mathbf{x}_j \in B} \|\mathbf{x}_i - \mathbf{x}_j\|}{\max_{\forall \mathbf{x}_k \in C}\left(\min_{\forall \mathbf{x}_j \in B} \|\mathbf{x}_k - \mathbf{x}_j\|\right)}\right)}{1+\kappa}, \tag{4.27}$$

where $\kappa \geq 0$ is a user parameter called *inner weighting factor*. The previous expression assigns a maximum value of 1 to all the collocation points closest to the boundary $\partial\Omega$, and a value of $1/(1+\kappa)$ to those collocation points with the maximum distance to $\partial\Omega$, i. e., the more interior points. When $\kappa = 0$, the standard fitness function is recovered because $\xi(\mathbf{x}) = 1$. Note that the weighting factor only depends on the collocation points. So it can be computed in a pre-processing step, not adding any extra computational cost to the algorithm. It is also important to notice that any connectivity is needed, so the method maintains its mesh-free feature.

According to Eq. (4.26), not only the candidate solution function is needed, but as well its derivatives. For that reason we have chosen Gaussian kernels: they are infinitely differentiable and the derivatives can be obtained straightforward. Nevertheless, there is not any general expression to obtain all the derivatives for any arbitrary order. The first derivative respect to component k is:

$$\frac{\partial y(\mathbf{x})}{\partial x_k} = -2\sum_{i=1}^{n} w_i \gamma_i (x_k - c_{ik}) \Phi(\mathbf{x}, \mathbf{c}_i). \tag{4.28}$$

The second derivative is a bit more complex. If $k \neq l$ we have

$$\frac{\partial^2 y(\mathbf{x})}{\partial x_k \partial x_l} = \sum_{i=1}^{n} w_i 4\gamma_i^2 (x_k - c_{ik})(x_l - c_{il}) \Phi(\mathbf{x}, \mathbf{c}_i), \tag{4.29}$$

otherwise

$$\frac{\partial^2 y(\mathbf{x})}{\partial x_k^2} = \sum_{i=1}^{n} w_i \gamma_i \left[4\gamma_i (x_k - c_{ik})^2 - 2\right] \Phi(\mathbf{x}, \mathbf{c}_i). \tag{4.30}$$

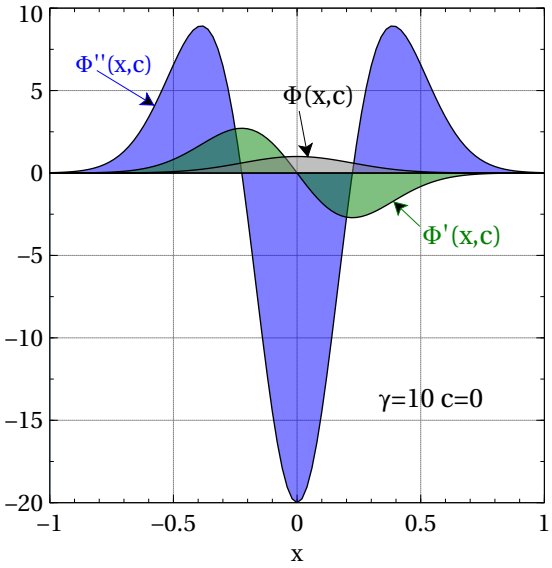

Figure 4.9: A Gaussian kernel $\Phi(x, c)$ and its first and second derivatives.

Higher order derivatives can be obtained by hand. This will depend on the DE to solve. In Fig. 4.9 it is plotted a Gaussian kernel $\Phi(x, c)$ in one dimension with center in the origin $c = 0$ and $\gamma = 10$. It is plotted as well the first and second derivatives. We can observe that although the function's zone of influence is located near the center, this is not true for the first and second derivatives. In fact, the first derivative has a zero value at the center and a local maximum and minimum at $x = c \pm \sqrt{1/2\gamma}$. The second derivative has two local maximum and one global minimum. $\Phi(x, c)$ and $\Phi''(x, c)$ are even functions, whereas $\Phi'(x, c)$ is an odd function. Therefore, the problem of adjusting simultaneously the values and derivatives of a function using kernels is not straightforward.

4.4.3 Algorithm Description

Following it is described the two search algorithms (global and local) used in this third approach.

Global Search of the solution: CMA-ES algorithm

In our second approach to solve DE, ES algorithm was adopted. Some convergence problems have been observed in some differential equations. Besides, the implementation is not

straightforward because several ES steps have been used. Therfore, a more powerful evolutionary algorithm has been adopted in this third approach: CMA-ES has been selected (see Chapter 3 for a detailed description). The transformed optimization problem appearing in DEs can be non-separable. CMA-ES is a second order approach estimating a positive definite matrix within an iterative procedure. This makes the method feasible on non-separable and/or badly conditioned problems [101]. Another interesting property of CMA-ES is that is feasible on non-smooth and even non-continuous problems. And finally, it does not require a tedious parameter tuning to be applied.

CMA-ES is used to search the best weights, centers and gammas, that is, those which get the minimum fitness value using Eq. (4.26). First of all, the unknowns are randomly initialized. As we see in Fig. 4.9, the influence of the kernel derivatives is located far away from its center. Therefore, the initial values of centers, c_{ik}, are set randomly in an extended range in the form

$$c_{ik} \in [x_{k,min} - \beta R_k, x_{k,max} + \beta R_k] \tag{4.31}$$

being R_k the original ranges for dimension k, $R_k = x_{k,max} - x_{k,min}$, and β an initialization parameter which control the range length. The domain range $[x_{k,min}, x_{k,max}]$ are obtained with all the collocation points within sets B and C, Eq. (1.4) and (1.5). Non-constraints are used in the variables (unknowns). Therefore it must be said that Eq. (4.31) is only applied in the initialization of the population, but not in the rest of generations.

All the defaults parameters for CMA-ES proposed by Hansen in his CMA-ES public implementation [83] are adopted, except the offspring number, λ, and the population size, μ. The default value for the offspring number is

$$\lambda_{default} = 4 + \lfloor 3 \ln N \rfloor , \tag{4.32}$$

and the population size is obtained dividing by 2: $\mu_{default} = \lfloor \lambda_{default}/2 \rfloor$. In these expressions N is the number of unknowns. In our case, the offspring number is slightly increased by multiplying the default value by a constant and, consequently, given that μ depends on λ, the population size is also increased. These changes have experimentally shown to produce

better results and decrease the dispersion between the runs. Therefore the final values for this couple of parameters were $\lambda = 3 \cdot \lambda_{default}$ and $\mu = \lfloor \lambda/2 \rfloor = \lfloor 3 \cdot \lambda_{default}/2 \rfloor \approx 3 \cdot \mu_{default}$. The N value, necessary to compute $\lambda_{default}$, is given by the Eq. (4.24).

Several generations are computed until some of the default stop criteria [83] are met. Then the best individual found at the end of the run is returned by the global search.

Local search of the solution: Downhill Simplex method

Once the CMA-ES algorithm is finished, a Downhill Simplex (DS) method (see Chapter 3 for a detailed description) is applied on the best individual obtained in the last generation of the evolutionary algorithm. For that the first simplex is computed applying sequentially a random increment to each unknown, maintaining the rest of unknowns unmodified. Expressed in a mathematical way, let $\mathbf{S}_0$ be a vector of size N with all the unknowns (w_i, γ_i and c_{ij}) obtained from the CMA-ES. The size of the vector is $N = n \cdot m \cdot (2 + d)$. A simplex is formed by the $N + 1$ nodes $\{\mathbf{S}_0, \mathbf{S}_1, \cdots, \mathbf{S}_N\}$. The nodes $\mathbf{S}_k = \{s_{k1}, s_{k2}, \cdots, s_{kN}\}$ of the initial simplex is computed applying a random step $\rho_k \in [-\triangle, \triangle]$ to each component in the following way:

$$s_{kl} = \begin{cases} s_{0l}, & if \quad k \neq l \\ \\ s_{0l} + \rho_k, & if \quad k = l \end{cases}, \tag{4.33}$$

where $\triangle \in \mathbb{R}^+$ is a user parameter and $k = 1, \cdots, N$.

Then several iterations on the simplex are performed applying the DS rules depending on the fitness values at the simplex nodes. Several stop criteria are checked: a maximum number of fitness evaluations, a distance from the best and worst simplex nodes below a predefined tolerance (parameter convergence criterion), and fitness value difference from the best and worst nodes below a threshold (target convergence criterion).

DS method, as other heuristics algorithms, can converge to local optimum depending on initialization issues. To reduce this effect, the algorithm is restarted several times recomputing the first simplex as it previously was described using as initial node the best one found so

far.

4.4.4 Other possible kernels

Gaussian kernels can be substituted by other different kernels, such as arctan kernel:

$$\Phi\left(\mathbf{x},\mathbf{c},v,a\right)=\arctan\left(v\mathbf{x}\mathbf{c}+a\right)=\arctan\left(v\sum_{j=1}^{d}x_{j}c_{j}+a\right) \tag{4.34}$$

As we see, each kernel has one center $\mathbf{c}$ and two parameters v and a. Candidate solutions are expressed as linear combination of n kernels:

$$y\left(\mathbf{x}\right)=\sum_{i=1}^{n}w_{i}\arctan\left(v_{i}\sum_{j=1}^{d}x_{j}c_{ij}+a_{i}\right).$$

We need (at least) the first and second derivatives:

$$\frac{\partial y\left(\mathbf{x}\right)}{\partial x_{k}}=\sum_{i=1}^{n}w_{i}\frac{v_{i}c_{ik}}{1+\left(v_{i}\sum_{j=1}^{d}x_{j}c_{ij}+a_{i}\right)^{2}},$$

$$\frac{\partial^{2}y\left(\mathbf{x}\right)}{\partial x_{k}\partial x_{l}}=-\sum_{i=1}^{n}w_{i}\frac{2v_{i}^{2}c_{ik}c_{il}\left(v_{i}\sum_{j=1}^{d}x_{j}c_{ij}+a_{i}\right)}{\left[1+\left(v_{i}\sum_{j=1}^{d}x_{j}c_{ij}+a_{i}\right)^{2}\right]^{2}}.$$

Fig. 4.10 shows one example kernel and its first and second derivatives with $v=10,c=1$ and $a=0$ in one dimension.

In order to understand why arctan kernels could achieve better performance than Gaussian kernels, the following experiment is performed. First one arctan kernel is approximated with only two Gaussian kernels solving a symbolic regression problem (Fig. 4.11, left). Then the same is test is performed in the opposite way, thus a Gaussian kernel is approximated with only 2 arctan kernels (Fig. 4.11, right). As we see, the arctan kernels obtain a better approximation.

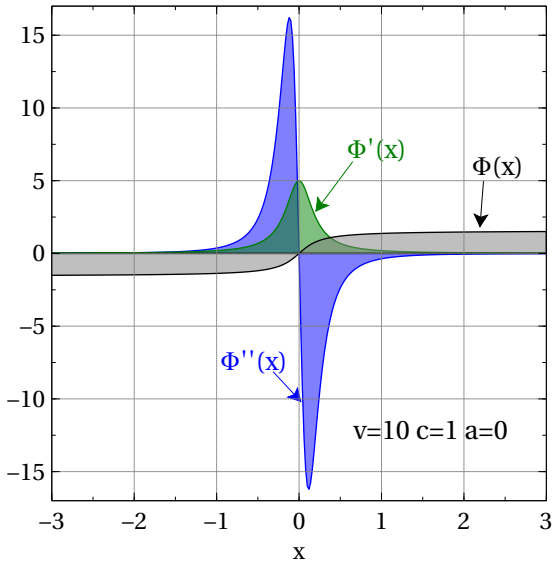

Figure 4.10: An one-dimensional example of arctan kernel with its first and second derivatives.

Other possibility is the polynomial kernel:

$$\Phi\left(\mathbf{x},\mathbf{c},v,a\right)=\left(\mathbf{xc}+a\right)^{v}=\left(\sum_{j=1}^{d}x_{j}c_{j}+a\right)^{v}$$

As usual, candidate solutions are expressed as linear combination of n kernels:

$$y\left(\mathbf{x}\right)=\sum_{i=1}^{n}w_{i}\left(\sum_{j=1}^{d}x_{j}c_{ij}+a_{i}\right)^{v_{i}}.$$

In the same way we need (at least) the first and second derivatives:

$$\frac{\partial y\left(\mathbf{x}\right)}{\partial x_{k}}=\sum_{i=1}^{n}w_{i}v_{i}c_{ik}\left(\sum_{j=1}^{d}x_{j}c_{ij}+a_{i}\right)^{v_{i}-1},$$

$$\frac{\partial^{2}y\left(\mathbf{x}\right)}{\partial x_{k}\partial x_{l}}=\sum_{i=1}^{n}w_{i}v_{i}\left(v_{i}-1\right)c_{ik}c_{il}\left(\sum_{j=1}^{d}x_{j}c_{ij}+a_{i}\right)^{v_{i}-2}$$

If $v_i \notin \mathbb{Z}$ the solution can be unfeasible when the base is negative. Therefore v_i is forced to

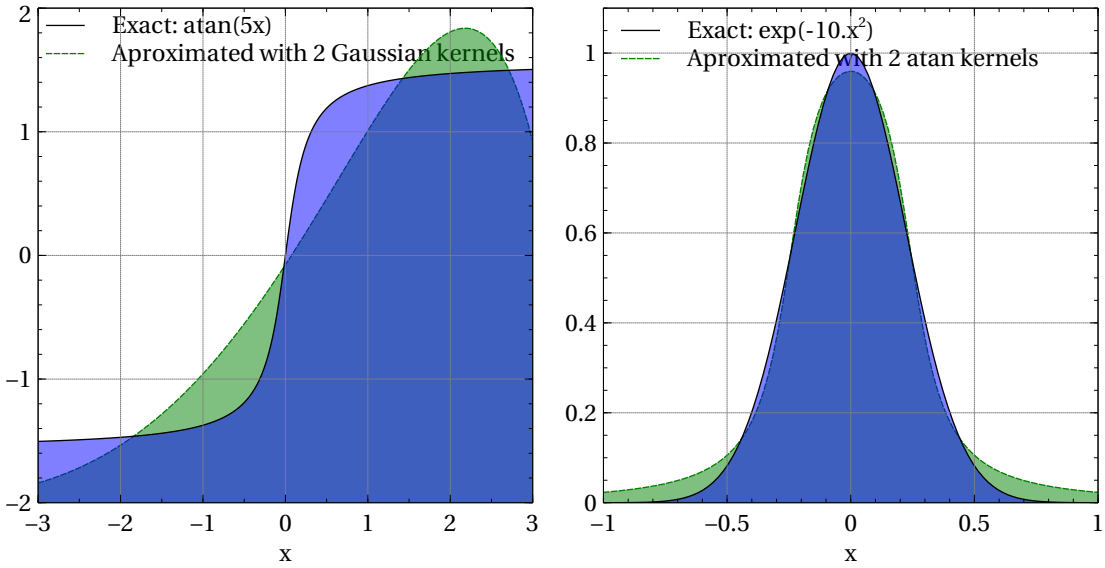

Figure 4.11: Approximating an arctan kernel with 2 Gaussian kernels (left) and approximating a Gaussian kernel with two atan kernels (right).

be a natural number.

Other possibility is the Dirichlet kernel:

$$\Phi\left(\mathbf{x},\mathbf{c},a\right)=\frac{\sin\left[\left(a+0.5\right)\left\|\mathbf{x}-\mathbf{c}\right\|\right]}{\sin\left[0.5\left\|\mathbf{x}-\mathbf{c}\right\|\right]}=\frac{\sin\left[\left(a+0.5\right)\sqrt{\sum_{j=1}^{d}\left(x_j-c_j\right)^2}\right]}{\sin\left[0.5\sqrt{\sum_{j=1}^{d}\left(x_j-c_j\right)^2}\right]},$$

where a normally is a natural number. As usual, candidate solutions are expressed as linear combination of n kernels:

$$y\left(\mathbf{x}\right)=\sum_{i=1}^{n}w_i\Phi\left(\mathbf{x},\mathbf{c}_i,a_i\right).$$

Due to the high complexity of the second derivative, only the first derivative is provided:

$$\frac{\partial y\left(\mathbf{x}\right)}{\partial x_k}=\sum_{i=1}^{n}w_i x_k\frac{a_i\cos\left[\left(a_i+0.5\right)\left\|\mathbf{x}-\mathbf{c}_i\right\|\right]\sin\left[0.5\left\|\mathbf{x}-\mathbf{c}_i\right\|\right]-0.5\sin\left[\left(a_i+0.5\right)\left\|\mathbf{x}-\mathbf{c}_i\right\|\right]\cos\left[0.5\left\|\mathbf{x}-\mathbf{c}_i\right\|\right]}{\sin^2\left[0.5\left\|\mathbf{x}-\mathbf{c}_i\right\|\right]\left\|\mathbf{x}-\mathbf{c}_i\right\|}.$$

Chapter 5

Results and Discussion

In this chapter, the three algorithms proposed for solving differential equations (DEs) are tested. For that, first of all, a set of problems to be solved are described in section 5.1, where equations, boundary conditions, domains and exact solutions (if they exist) are provided. Once all the test problems are presented, the proposed algorithms are tested in section 5.2. Finally, several comparisons between them and with other methods in the literature are given in section 5.3.

5.1 Benchmarking Problems

The proposed algorithms have been applied over a total of 32 differential equations extracted from the literature related to this domain of application. In order to demonstrate the capabilities of the proposed approaches, a wide range of problems are studied: linear ordinary DEs (LODEs), non linear ordinary DEs (NLODEs), systems of ordinary DEs (SODEs) and partial DEs (PDEs). Tables 5.1 and 5.2 show all the problems used and the references of papers where they were also solved.

Case	Equation	Range	Boundary Conditions	Used in
LODE1	$y' = (2x - y)/x$	$x \in [1, 2]$	$y(1) = 3$	[1, 2]
LODE2	$y' = (1 - y\cos(x))/\sin(x)$	$x \in [1, 2]$	$y(1) = 3/\sin 1$	[1, 2]
LODE3	$y'' = 6y' - 9y$	$x \in [0, 1]$	$y(0) = 0; y'(0) = 2$	[2]
LODE4	$y'' + \frac{1}{5}y' + y = -\frac{1}{5}e^{-x/5}\cos(x)$	$x \in [0, 1]$	$y(0) = 0$ $y(1) = \sin(1)/e^{0.2}$	[1, 2]
LODE5	$xy'' + y' = \cos(x)$	$x \in [0, 1]$	$y(0) = 0; y'(0) = 1$	[12, 1, 2]
LODE6	$y'' + 2xy = 0$	$x \in [0, 1]$	$y(0) = 0; y'(0) = 1$	[1, 2]
LODE7	$y''(x^2 + 1) - 2xy - x^2 - 1 = 0$	$x \in [0, 1]$	$y(0) = 0; y'(0) = 1$	[1, 2]
LODE8	$y' + 2y = 1$	$x \in [0, 10]$	$y(0) = 1$	[36]
LODE9	$y' + 2y = \sin(x)$	$x \in [0, 10]$	$y(0) = 1$	[36]
LODE10	$y'' = -16\pi^2 \sin(4\pi x)$	$x \in [0, 1]$	$y(0) = 2; y(1) = 2$	[18]
LODE11	$u'' + 2u' + 5u = 0$	$x \in [0, \pi]$	$u(0) = 0; u'(0) = 1$	[4]
NLODE1	$y' = 1/(2y)$	$x \in [1, 4]$	$y(1) = 1$	[1, 2]
NLODE2	$(y')^2 + \log y =$ $\cos^2 x + 2\cos x + 1 + \log(x + \sin x)$	$x \in [1, 2]$	$y(1) = 1 + \sin 1$	[1, 2]
NLODE3	$y''y' = -4/x^3$	$x \in [1, 2]$	$y(1) = 0; y'(1) = 2$	[1, 2]
NLODE4	$x^2y'' + (xy')^2 + 1/\log x = 0$	$x \in [e, 2e]$	$y(e) = 0; y'(e) = 1/e$	[1, 2]
NLODE5	$y'' - yy'/(x\sin x^2) = -4x^2 \sin x^2$	$x \in [1, 2]$	$y(1) = \sin 1; y(2) = \sin 4$	-
NLODE6	$10^{-4}y'' + y - y^3 = 0$	$x \in [-1, 1]$	$y(-1) = -1; y(1) = 1$	[18]

Table 5.1: Test cases (linear and non-linear equations): differential equations, ranges and boundary conditions.

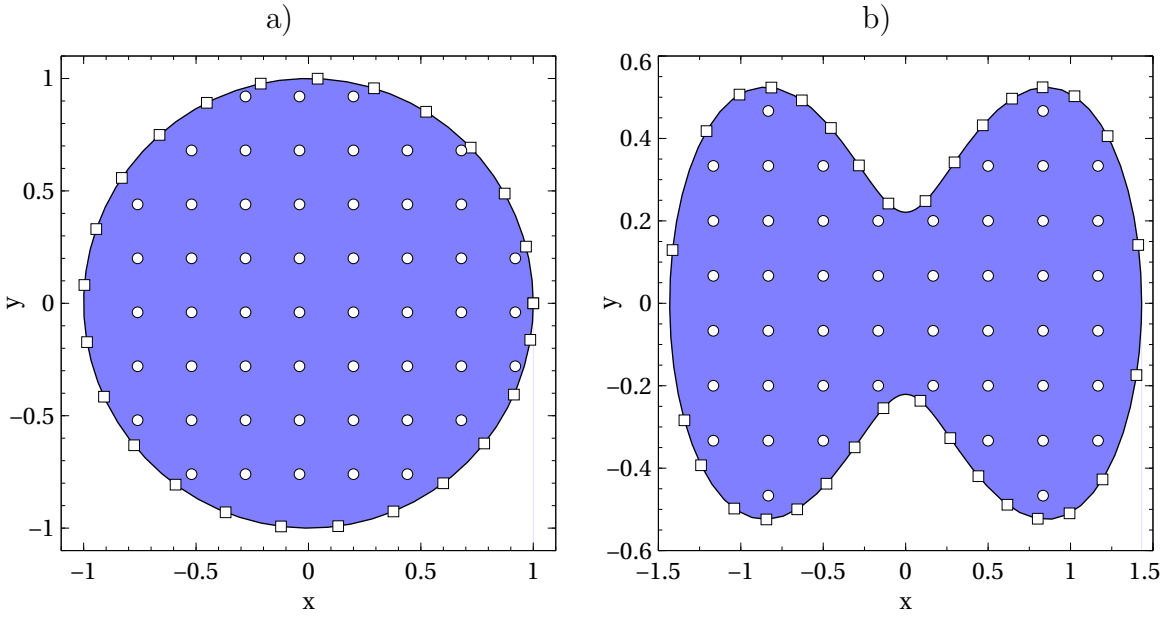

Figure 5.1: Non-rectangular domains. Unit circle for PDE5 (a), and Cassini's oval for PDE6 (b).

Case	Equation	Range	Boundary Conditions	Used in
SODE1	$\left.\begin{array}{c} y_1' = \cos x + y_1^2 + \\ + y_2 - \left(x^2 + \sin^2 x\right) \\ y_2' = 2x - x^2 \sin x + y_1 y_2 \end{array}\right\}$	$x \in [0, 1]$	$\left.\begin{array}{c} y_1(0) = 0 \\ y_2(0) = 0 \end{array}\right\}$	[1, 2]
SODE2	$\left.\begin{array}{c} y_1' = (\cos x - \sin x)/y_2 \\ y_2' = y_1 y_2 + e^x - \sin x \end{array}\right\}$	$x \in [0, 1]$	$\left.\begin{array}{c} y_1(0) = 0 \\ y_2(0) = 1 \end{array}\right\}$	[1, 2]
SODE3	$\left.\begin{array}{c} y_1' = \cos x \\ y_2' = -y_1 \\ y_3' = y_2 \\ y_4' = -y_3 \\ y_5' = y_4 \end{array}\right\}$	$x \in [0, 1]$	$\left.\begin{array}{c} y_1(0) = 0 \\ y_2(0) = 1 \\ y_3(0) = 0 \\ y_4(0) = 1 \\ y_5(0) = 0 \end{array}\right\}$	[1, 2]
SODE4	$\left.\begin{array}{c} y_1' = -\sin\left(e^x\right)/y_2 \\ y_2' = -y_2 \end{array}\right\}$	$x \in [0, 1]$	$\left.\begin{array}{c} y_1(0) = \cos 1 \\ y_2(0) = 1 \end{array}\right\}$	[1, 2]
SODE5	$\left.\begin{array}{c} I' = -I - V \\ V' = 2I - V \end{array}\right\}$	$t \in [0, 1.5]$	$\left.\begin{array}{c} I(0) = 2 \\ V(0) = 2 \end{array}\right\}$	[4]
SODE6	$\left.\begin{array}{c} \left[1 + (y')^2\right] y = k^2 \\ k' = 0 \end{array}\right\}$	$x \in [0, 1]$	$\left.\begin{array}{c} y(0) = 0 \\ y(1) = 2 \end{array}\right\}$	[4]
PDE1	$\left.\begin{array}{c} \Psi_{xx} + \Psi_{yy} = \\ e^{-x}\left(x - 2 + y^3 + 6y\right) \end{array}\right\}$	$x, y \in [0, 1]$	$\left.\begin{array}{c} \Psi(0, y) = y^3 \\ \Psi(1, y) = \left(1 + y^3\right) e^{-1} \\ \Psi(x, 0) = x e^{-x} \\ \Psi(x, 1) = (x + 1) e^{-x} \end{array}\right\}$	[12, 1, 3, 2, 5]
PDE2	$\Psi_{xx} + \Psi_{yy} = -2\Psi$	$x, y \in [0, 1]$	$\left.\begin{array}{c} \Psi(0, y) = 0 \\ \Psi(1, y) = \sin(1)\cos(y) \\ \Psi(x, 0) = \sin(x) \\ \Psi(x, 1) = \sin(x)\cos(1) \end{array}\right\}$	[1, 2, 5]
PDE3	$\Psi_{xx} + \Psi_{yy} = 4$	$x, y \in [0, 1]$	$\left.\begin{array}{c} \Psi(0, y) = y^2 + y + 1 \\ \Psi(1, y) = y^2 + y + 3 \\ \Psi(x, 0) = x^2 + x + 1 \\ \Psi(x, 1) = x^2 + x + 3 \end{array}\right\}$	[1, 2, 5]
PDE4	$\Psi_{xx} + \Psi_{yy} = -\Psi\left(x^2 + y^2\right)$	$x, y \in [0, 1]$	$\left.\begin{array}{c} \Psi(0, y) = 0 \\ \Psi(1, y) = \sin(y) \\ \Psi(x, 0) = 0 \\ \Psi(x, 1) = \sin(x) \end{array}\right\}$	[1, 5]
PDE5	$\left.\begin{array}{c} \Psi_{xx} + \Psi_{yy} = 4x\cos x + \\ + \left(5 - x^2 - y^2\right)\sin x \end{array}\right\}$	$x^2 + y^2 \leq 1$	$\Psi(x, y) = 0$ *in* $\partial\Omega$	[3, 5]
PDE6	$\Psi_{xx} + \Psi_{yy} = 2e^{(x-y)}$	$R^2(\theta) \leq \cos(2\theta) + \sqrt{1.1\sin^2(2\theta)}$	$\left.\begin{array}{c} \Psi = e^x\left(e^{-y} + \cos y\right) \\ in\ \partial\Omega \end{array}\right\}$	[3, 5]
PDE7	$u_x + u_t = 0$	$x, t \in [0, 2]$	$\left.\begin{array}{c} u(0, t) = e^{-2(t-1)^2} \\ u(x, 0) = e^{-2(x+1)^2} \end{array}\right\}$	-
PDE8	$u_{xx} - u_{tt} = 0$	$x, t \in [0, 1]$	$\left.\begin{array}{c} u(0, t) = 0 \\ u(1, t) = 0 \\ u(x, 0) = 0 \\ u_t(x, 0) = \pi\sin(\pi x) \end{array}\right\}$	-
PDE9	$u_t - u u_x = u_{xx}$	$x, t \in [0, 1]$	$\left.\begin{array}{c} u(0, t) = 1 + 2/(t + 1) \\ u(x, 0) = 1 + 2/(x + 1) \\ u_x(0, t) = -2/(t + 1)^2 \end{array}\right\}$	-

Table 5.2: Test cases (systems and partial equations): differential equations, ranges and boundary conditions.

Following some notes will be given of some equations of the benchmark set. Thus, the original problem LODE11 was presented in [4] as an integro-differential equation $u' + 2u + 5\int_0^x u(t)\,dt = 1$ in the range $[0, \pi]$ with the boundary condition $u(0) = 0$. Taking derivatives and using the fundamental theorem of calculus [6], the original problem is transformed into an ordinary differential equation as it is presented in Table 5.1. SODE5 and SODE6 were presented as well in [4]. The former describes a simple engineering application dealing with electric current and potential difference in an electronic circuit. The later describes the *brachistochrome* problem, which consists on finding the curve along with a particle slides from a given point to lower one without friction in the least time. The problem is formulated as a system of differential equations because a constant k must be determined. Note that PDE5 and PDE6 are defined on non-rectangular domains (Fig. 5.1). To show the potential of our approximations, some partial differential equations different from *Poisson* problems presented in Sobester et al. [3] are defined in Table 5.1. Thus, PDE7 describes a *traveling wave* with velocity +1. PDE8 correspond to the one dimensional *wave* equation with constant speed of wave propagation $c = 1$. Finally, PDE9 is the inviscid *Burgers' equation* which is a prototype for equations for which the solution can develop discontinuities (shock waves).

The exact solutions are provided in Table 5.3 with the exception of *Allen-Cahn* equation [18] in NLODE6, which does not have a close analytical solution. The solution of SODE6 is a parametric equation of a *cycloid*. The constant $k^2 \simeq 4.81125$ is obtained solving a transcendental equation.

Measuring the Solution Quality

Fitness function value is not a good measure for comparing the final solutions obtained in different problems because it is very dependent on some algorithm parameters as for example the boundary condition penalty φ. Moreover, the fitness value can be modified depending on how the differential equation is provided to Eq. (4.26). For instance, LODE1 equation can be provided as $y' - (2x - y)/x$, $y'x - (2x - y)$ or even $[y' - (2x - y)/x]/k$. Although the same

Case	Exact Solution
LODE1	$y = x + 2/x$
LODE2	$y = (x+2)/\sin(x)$
LODE3	$y = 2xe^{3x}$
LODE4	$y = e^{-x/5}\sin(x)$
LODE5	$y = \int_0^x \frac{\sin(t)}{t} dt$
LODE6	$y = \int_0^x e^{-t^2} dt$
LODE7	$y = \left(x^2+1\right)\arctan(x)$
LODE8	$y = \left(e^{-2x}+1\right)/2$
LODE9	$y = \left[6e^{-2x} + 2\sin(x) - \cos(x)\right]/5$
LODE10	$y = 2 + \sin(4\pi x)$
LODE11	$u = 0.5e^{-x}\sin(2x)$
NLODE1	$y = \sqrt{x}$
NLODE2	$y = x + \sin(x)$
NLODE3	$y = \log\left(x^2\right)$
NLODE4	$y = \log(\log(x))$
NLODE5	$y = \sin\left(x^2\right)$
NLODE6	No analytical solution
SODE1	$y_1 = \sin x$, $y_2 = x^2$
SODE2	$y_1 = \sin(x)/e^x$, $y_2 = e^x$
SODE3	$y_1 = y_3 = y_5 = \sin x$, $y_2 = y_4 = \cos x$
SODE4	$y_1 = \cos(e^x)$, $y_2 = e^{-x}$
SODE5	$I = e^{-t}\left[2\cos\left(\sqrt{2}t\right) - \sqrt{2}\sin\left(\sqrt{2}t\right)\right]$, $V = 2e^{-t}\left[\cos\left(\sqrt{2}t\right) + \sqrt{2}\sin\left(\sqrt{2}t\right)\right]$
SODE6	$x = k^2(\theta - \sin\theta)/2$, $y = k^2(1-\cos\theta)/2$, $\frac{2}{k^2} = \arccos\left(1-\frac{4}{k^2}\right) - \sin\arccos\left(1-\frac{4}{k^2}\right)$
PDE1	$\Psi = \left(x+y^3\right)e^{-x}$
PDE2	$\Psi = \sin(x)\cos(y)$
PDE3	$\Psi = x^2 + y^2 + x + y + 1$
PDE4	$\Psi = \sin(xy)$
PDE5	$\Psi = \left(x^2+y^2-1\right)\sin x$
PDE6	$\Psi = e^{(x-y)} + e^x\cos y$
PDE7	$u = e^{-2(t-x-1)^2}$
PDE8	$u = \sin(\pi t)\cdot\sin(\pi x)$
PDE9	$u = 1 + 2/(x+t+1)$

Table 5.3: Exact solutions for the test cases.

problem is solved, the fitness function values are modified: First and last equations have the same fitness landscape, but with a scale value given by the constant k. Second equation has a different fitness landscape so could have a different behavior during the evolving process.

Therefore we believe that a new metric for measuring the final solution quality obtained is necessary. Concretely, in this work we propose using the *Root of the Mean Squared Error* (RMSE) between the computed final solution $\mathbf{y}$ and the exact solution $\mathbf{y}_{exact}$:

$$RMSE = \sqrt{\frac{\sum_{i=1,\mathbf{x}_i \in C}^{n_C} \|\mathbf{y}(\mathbf{x}_i) - \mathbf{y}_{exact}(\mathbf{x}_i)\|^2 + \sum_{j=1,\mathbf{x}_j \in B}^{n_B} \|\mathbf{y}(\mathbf{x}_j) - \mathbf{y}_{exact}(\mathbf{x}_j)\|^2}{d \cdot (n_C + n_B)}}. \quad (5.1)$$

Other authors use the same measure for determine the achieved accuracy [17]. This error does not deal with the differential equation residuals, but measures distances between the computed solution and the exact one. Obviously RMSE can be only used when the exact solution is known and for benchmarking purposes.

Because NLODE6 does not have analytical solution, the RMSE for this problem has been computed using a numerical approximation of the solution computed with a Runge-Kutta relaxation method [102] discretizing the domain range in 1000 nodes.

5.2 Numerical Experiments

Once the benchmarking problems are described, the results obtained applying the three algorithms explained in Chapter 4 are provided in this section. The algorithm results are presented in chronological order, that is, according to the date of implementation and analysis during the development of the present thesis. As we will see, the most powerful algorithm is the last one, that based on Covariance Matrix Adaptation Evolutionary Strategies (DESCMA-ES), meanwhile the weak approach is the first one, based on Grammatical Evolution (DESGE). Therefore, only in the last approach an exhaustive benchmarking process is followed, solving all the equations presented in Tables 5.1 and 5.2.

Parameter	Value
Maximum wrapping	1
Population	1000
Maximum generations	2000
Fitness stop criterion	10^{-10}
Minimum chromosome size at initialization	20
Maximum chromosome size at initialization	50
Maximum chromosome size	100
Tournament size	3
Recombination probability	0.9
Mutation probability	0.2
Elitism size	3

Table 5.4: Control parameters for baseline DESGE algorithm.

5.2.1 DESGE Results

In this section the results obtained applying out first approximation to solve DEs are reported. First of all, the baseline algorithm based in Grammatical Evolution is employed to solve the LODE2 equation. Then, a more exhaustive analysis using more equations is reported using the enhanced techniques described in section 4.2.3.

Baseline algorithm

LODE2 problem has been chosen to test the baseline DESGE algorithm. Ten constants have been defined with all the integer values between 0 and 9 denoted by $C0 = 0$, $C1 = 1$, $C2 = 2$, ..., $C9 = 9$. Operators and functions used are: $+$,$-$, $*$, $/$, sin, cos, exp and log. A total of 19 collocation points have been defined equispaced along the domain range. The penalty number φ for the boundary condition is set to 100, Eq. (4.1). The best results have been obtained with the control parameters given in Table 5.4.

The solver has been run 30 times, and in 29 of them the exact solution has been found. The average number of generations needed has been 758, with a minimum of 39 (for the average

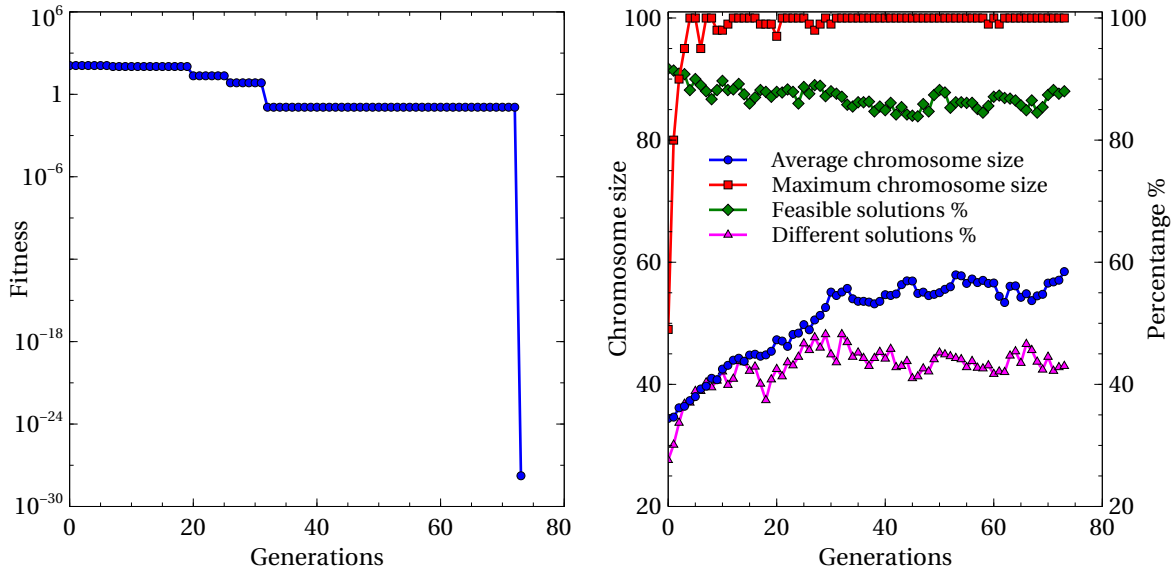

Figure 5.2: Convergence history for LODE2 problem of one successful run using baseline DESGE algorithm: fitness versus generations (left) and average chromosome size, maximum chromosome size, percentage of feasible and different solutions versus generation (right).

computation, the unsuccessful run is not taking into account). All the exact solutions have the same fitness (zero machine), but several different phenotype has been found:

$$((C2 + x)/(sin(x)))$$

$$((x + C2)/sin((x)))$$

$$((((x) + C2))/sin((x - ((log(C1))))))$$

$$(((x + (((x/x) * C2)))/sin(x)))$$

$$((C1 + (C1 + x))/sin(x))$$

Nevertheless, the symbolic engine simplifies all this solutions in what we call canonical expression: `(x+C2)*sin(x)^(-1)`.

In Fig. 5.2 one simulation history is plotted. As we can see, the exact solution is isolated in the fitness domain, that is, no smooth convergence is observed. Because of this reason is

so difficult to find the exact solution. In the figure at the right side is also shown the average chromosome size at the population, the maximum chromosome size, the percentage of feasible solutions, and the percentage of feasible and different solution within the population. Two solutions are considered different according to the fitness function, so it is not based on the genotypic space.

Enhanced DESGE

Although good results are obtained for LODE2 problem using the baseline algorithm, when more complex equations are solved, the algorithm does not behave as expected: it is not able to find the exact solution, or big dispersion in the results is observed, or the approximated solution is not enough accurate. Part of the problem is related with the inefficient behavior to seek numerical constants such as π, e or $\sqrt{2}$ without a local search. In this section, some results obtained using all the enhanced techniques described in section 4.2.3 are described. For the sake of conciseness, the four techniques (ephemeral local search, Gaussian local search, addition crossover and new grammar for real constants) are applied together. Previously, they have been applied one by one to check the benefits obtained respect to the original baseline algorithm.

Table 5.5 shows the control parameters used in this analysis. A total of 20 runs has been done and the average numbers are reported. For cases LODE3 to NLODE4 operator $\hat{}$ has been added to the operator set and the ephemeral constant are limited in the range $[-10, 10]$. In the last two columns, the averaged fitness evaluations needed by Tsoulos et al. in works [1] and [2] are presented. To do this comparison, it is important to notice that a local optimization technique is used in [2], which is a Broyden-Fletcher-Goldfarb-Shanno (BFGS) method variant due to Powell [103]. The BFGS method approximates Newton's method, a class of hill-climbing optimization techniques that seeks a stationary point of a (twice continuously differentiable) function. The fitness evaluation number in [2] is obtained estimating that the average number of neurons in the hidden layer is 10 and that 20 steps are

Parameter	Value
Ephemeral constants	7 digits, unlimited range
Constants	Not used
Maximum Wrapping	1
Population	500
Generations	2500
Max. Fitness Evaluation	10^5
Fitness Value Stop Criteria	10^{-7}
Initialization sizes	Between 20 and 50
Maximum Chromosome size	600
Tournament size	3
Recombination probability	0.9
Mutation probability	0.3
Elitism size	10
Ephemeral Local Search	20 steps, probability 0.5
Gaussian Local Search	No fitness variation after 10 generations
Crossover Type	Addition

Table 5.5: Control parameters for enhanced DESGE algorithm.

Case	Gene.	Fitness Eval.	%Success Runs	Best Fitness	Time per Run	Fitness Eval. in [1]	Fitness Eval. in [2]
LODE1	10	4511	100	1.65984e-11	15.8s	653000	59940
LODE2	141	58236	100	2.66559e-12	4m 32s	742000	42120
LODE3	168	10003	0	2.61997e-02	44m 29s	-	23220
LODE4	154	100292	0	6.90137e-02	41m 20s	714000	102060
LODE5	380	100121	0	1.6842e-06	12m 32s	441000	64260
LODE6	172	100463	0	3.8989e-02	36m 52s	451000	35100
LODE7	166	100488	0	1.1688e-01	42m 0s	444000	493020
NLODE1	2	1298	100	1.96932e-09	2.4s	182000	37260
NLODE2	2.8	1210	100	4.18804e-09	13s	86000	218700
NLODE3	4	2780	100	5.07432e-09	54s	191000	454680
NLODE4	4.6	3122	100	3.09527e-18	48s	161000	156060

Table 5.6: Results obtained using enhanced DESGE algorithm. Average values of 20 repetitions are reported. In [1] and [2], averaged fitness evaluation with 30 repetitions are provided, being the success rate 100% in all the problems.

performed[1]. Therefore, the fitness evaluation number in [2] is estimated with the following formula

$$FitnessEval = g \cdot 500 + \frac{g}{20} 20 \cdot 40,$$

being g the generation number. On the other hand, the number of fitness evaluations in [1] is straightforward because no local search is employed (a population size of 1000 individuals is used):

$$FitnessEval = g \cdot 1000.$$

The column of "successful runs" at Table 5.6 are computed considering that the exact symbolic solution has been found, that is, a canonical expression coincident with the exact solutions given in Table 5.3 is obtained by the algorithm. For these type of solutions, the ideal fitness value should be 0. However, at Table 5.6 non zero values are obtained due to round-off errors. Anyway the values are very low, between 10^{-9} and 10^{-18}. It is important to notice that the successful run rate of works [1] and [2] is 100%.

As we can see, for some equations, enhanced DESGE algorithm outperforms results presented in [1] and [2] by several order of magnitude in the fitness evaluation number (LODE1, NLODE1, NLODE2, NLODE3 and NLODE4), but for other problems it is not possible to reduce the fitness values more than a minimum value depending on the case (LODE3 to LODE7). Although some convergence problems have been observed, there are two important drawbacks to these type of algorithms using generic mathematical expressions. First of all, the exact solution only can be found if all the terminals and functions needed to express the symbolic expression are defined in the grammar. Secondly, the majority of real problems have not a closed analytical solution. In these type of problems, only approximated solutions can be found. Because of the characteristic of the method, the search process could turn inefficient. If more terminals and functions are incorporated to the original grammar, the search space increases, making the process much more difficult.

[1] According to Ioannis Tsoulos' comments: *"We have used analytical evolution of the gradient for the neural network error and not some estimation. I think that for every neural network 20 function in the BFGS local search procedure were sufficient."* One fitness evaluation is considered here for each parameter in order to compute the analytical gradient.

5.2.2 DESES Results

The method was run 10 times on every differential equation described previously using different seeds for the random number generator and averages were taken. The number of repetitions has been considered sufficient for the analysis because low dispersion in the results has been observed, as we will see in the following lines. A total of 100 equidistant collocation points have been used in all cases, except for PDE5 and PDE6. In those non-rectangular domain cases, 51 internal points and 25 boundary points have been used for PDE5 (Fig. 5.1a), and 48 internal and 32 boundary points in PDE6 (Fig. 5.1b) as in [3]. A maximum harmonic order $N = 10$ has been used, which gives a total of 11 harmonics for LODEs and NLODEs problems, 22 harmonics for SODEs cases (systems of two equations each), except SODE3 (system of five equations) with 55 harmonics, and 101 harmonics for PDEs.

Table 5.7 lists the algorithm parameter values which have been used for all test cases and the good results obtained provide evidence about the robustness of the method. However, more improvements could be obtained if the parameters were tuned specifically for each case. In this way, we can differentiate those parameters which are directly sensitive to the solution quality (maximum harmonic order N and range extension ξ) and those which affect to the convergence process (rest of algorithm parameters in Table 5.7). Regarding the former, the required maximum harmonic order N depends on the variations of the solution function and its derivatives. Generally speaking, more harmonics implies better accuracy in terms of a lower value of RMSE. As it will further be shown, good results have been obtained for $N = 10$ because the solution functions do not have high frequency harmonics. Nevertheless, in some punctual cases higher harmonics are needed. This is the case of LODE3, LODE8, LODE9, LODE10 and NLODE6 problems as it will further be described. The range extension ξ should be higher than 0 when no null first derivatives are desired in the domain boundaries. Good results have been obtained for $\xi = 1$. Low sensitivity has been observed when this value is increased or reduced, except for values close to 0. Concerning the second type of parameters, the initialization values have a high sensitivity in the results. According to Table

Parameter	**Values**
Maximum harmonic order N	10
Initial coefficients	Between $\alpha = -10^{-3}$ and $\beta = 10^{-3}$
Initial mutation strengths	Between $\gamma = 3 \cdot 10^{-4}$ and $\delta = 3 \cdot 10^{-3}$
Parent selection	$(\mu, \lambda) = (10, 400)$
Learning Rate factor f_τ	1
Boundary Condition Penalty φ	300
Range extension ξ	1
Stop criteria	15 generations with unchanged fitness or 80 generations
Maximum Generations G	3000
Max. new harmonics in each step	8

Table 5.7: Numerical values for the parameters of the DESES method.

5.7, the best results have been obtained using low values for the initial coefficients, and a similar maximum value for the initial mutation strengths. Regarding the population, as expected, better convergence is achieved increasing its size. A trade-off among performance and computational cost is achieved using a population size of $\mu = 10$. A higher selective pressure than $\lambda/\mu \simeq 7$ (according to [81]) has been observed more effective. The learning rate factor could be increased in some cases up to 2 or 3 for speed-up the convergence process. According to the definition of the fitness function, Eq. (4.1), the boundary condition penalty φ should be of the same order than the number of collocation points and the boundary point ratio n_C/n_B. Finally, in simple cases as LODEs, the stop criteria could be relaxed in order to find the solution in less number of generations.

Table 5.8 shows the result of DESES algorithm over the test cases. Average values of the 10 runs are listed for fitness and it can be seen the RMSE values of the best individual in the last generation and the number of generations used. This table also gives the standard deviation of these values, showing low dispersion in the results. These low dispersions observed imply that it is not necessary to increase the number of repetitions of each equation (10 runs). As we can appreciate, LODE3 problem results are very different from the rest of test cases. At

Case	Fitness	RMSE	Generations
LODE1	$(5.94 \pm 0.630) \cdot 10^{-7}$	$(3.70 \pm 0.166) \cdot 10^{-5}$	2581 ± 476
LODE2	$(2.51 \pm 0.128) \cdot 10^{-6}$	$(6.59 \pm 0.255) \cdot 10^{-5}$	3000 ± 0
LODE3	7.48 ± 0.0132	13.16 ± 0.0135	3000 ± 0
LODE4	$(4.96 \pm 0.003) \cdot 10^{-6}$	$(1.65 \pm 0.872) \cdot 10^{-6}$	1113 ± 38
LODE5	$(4.31 \pm 0.365) \cdot 10^{-8}$	$(9.90 \pm 0.383) \cdot 10^{-5}$	819 ± 21
LODE6	$(2.91 \pm 1.47) \cdot 10^{-8}$	$(5.44 \pm 3.64) \cdot 10^{-6}$	801 ± 74
LODE7	$(9.34 \pm 0.465) \cdot 10^{-6}$	$(1.25 \pm 0.264) \cdot 10^{-5}$	3000 ± 0
LODE8	$(1.51 \pm 0.02) \cdot 10^{-3}$	$(1.22 \pm 0.01) \cdot 10^{-2}$	3000 ± 0
LODE9	$(9.98 \pm 0.291) \cdot 10^{-3}$	$(3.15 \pm 0.052) \cdot 10^{-2}$	3000 ± 0
LODE10	$(1.46 \pm 0.205) \cdot 10^{2}$	$(3.90 \pm 0.165) \cdot 10^{-2}$	3000 ± 0
NLODE1	$(2.26 \pm 0.963) \cdot 10^{-7}$	$(7.42 \pm 1.81) \cdot 10^{-5}$	1218 ± 419
NLODE2	$(7.87 \pm 0.707) \cdot 10^{-8}$	$(5.90 \pm 0.549) \cdot 10^{-6}$	1349 ± 106
NLODE3	$(1.12 \pm 0.068) \cdot 10^{-5}$	$(3.64 \pm 0.492) \cdot 10^{-5}$	3000 ± 0
NLODE4	$(3.67 \pm 0.081) \cdot 10^{-7}$	$(8.32 \pm 0.791) \cdot 10^{-5}$	2187 ± 49
NLODE5	$(3.58 \pm 1.03) \cdot 10^{-7}$	$(3.19 \pm 0.596) \cdot 10^{-6}$	2755 ± 78
NLODE6	$(3.12 \pm 0.019) \cdot 10^{-2}$	$(3.03 \pm 0.009) \cdot 10^{-1}$	3000 ± 0
SODE1	$(2.13 \pm 0.218) \cdot 10^{-7}$	$(7.67 \pm 1.66) \cdot 10^{-5}$	1117 ± 147
SODE2	$(2.43 \pm 2.43) \cdot 10^{-8}$	$(3.90 \pm 0.377) \cdot 10^{-5}$	2701 ± 166
SODE3	$(1.73 \pm 0.588) \cdot 10^{-7}$	$(8.51 \pm 4.02) \cdot 10^{-5}$	1149 ± 48
SODE4	$(1.17 \pm 0.099) \cdot 10^{-6}$	$(4.72 \pm 0.261) \cdot 10^{-5}$	3000 ± 0
SODE5	$(3.58 \pm 1.03) \cdot 10^{-7}$	$(3.19 \pm 3.18) \cdot 10^{-6}$	2755 ± 78
PDE1	$(1.56 \pm 0.301) \cdot 10^{-2}$	$(6.37 \pm 0.733) \cdot 10^{-3}$	2700 ± 57
PDE2	$(7.18 \pm 1.77) \cdot 10^{-4}$	$(1.16 \pm 0.214) \cdot 10^{-3}$	1956 ± 139
PDE3	$(1.70 \pm 0.363) \cdot 10^{-2}$	$(5.90 \pm 0.799) \cdot 10^{-3}$	2564 ± 156
PDE4	$(6.90 \pm 0.676) \cdot 10^{-4}$	$(1.23 \pm 0.036) \cdot 10^{-3}$	2066 ± 102
PDE5	$(9.24 \pm 2.10) \cdot 10^{-4}$	$(9.06 \pm 1.29) \cdot 10^{-4}$	2599 ± 66
PDE6	$(2.22 \pm 0.624) \cdot 10^{-1}$	$(1.79 \pm 0.384) \cdot 10^{-2}$	2979 ± 42

Table 5.8: Experimental results for DESES algorithm. Data are presented giving the average values and the standard deviations.

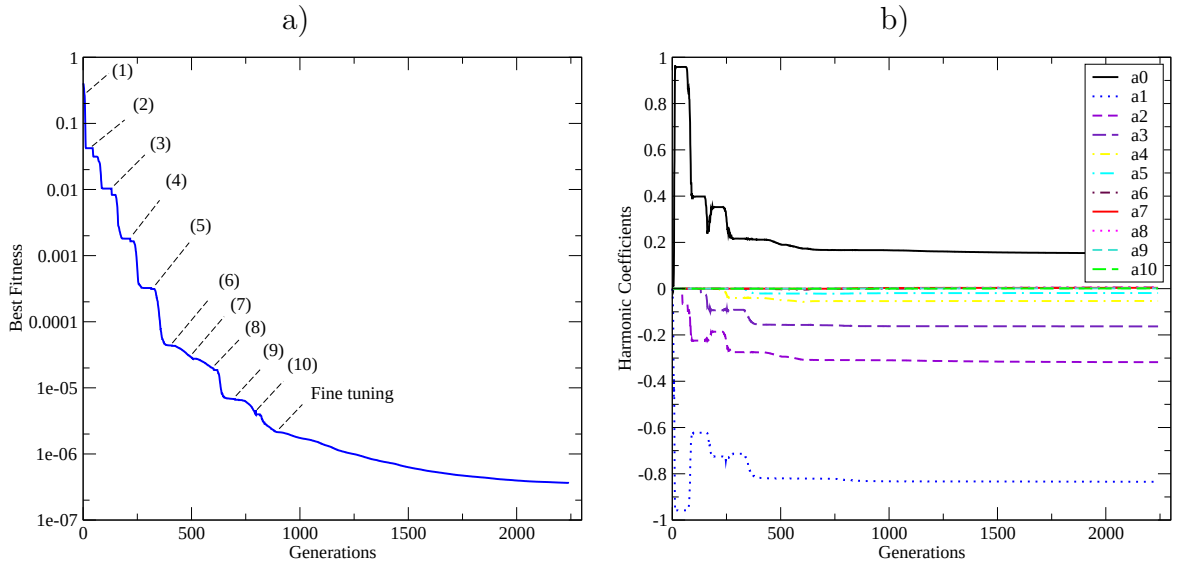

Figure 5.3: Fitness value (a) and harmonic coefficients (b) of the best individual over the generations for one run of NLODE4 case using DESES algorithm. In (a) number in parentheses indicate the number of *active* plus *frozen* harmonics. *Fine tuning* step is shown as well.

the end of this subsection further details will be given. Excluding LODE3 problem, good results have been obtained with RMSE values between 10^{-6} and 10^{-1} for one-dimensional problems, and between 10^{-4} and 10^{-2} for PDEs. As an example, Fig. 5.3 and 5.4 show plots related to the evolution of characteristic parameters of the algorithm and the quality of the solution obtained for a run of the NLODE4 case. In particular, Fig. 5.3a and Fig. 5.3b show the fitness value and harmonic coefficients of the best individual over the generations. Fig. 5.4a shows the evolution of the mutation strengths of the best individual with the generation number. Fig. 5.4b compares the exact solution with the computed one. We can appreciate the good matching obtained.

Number of harmonics study

Observing the RMSE for one dimensional problems in table 5.8, we can see that all the cases have achieved a good accuracy with RMSEs values between 10^{-5} and 10^{-6} except for cases LODE3, LODE8, LODE9, LODE10 and NLODE6. These last cases have a more complicated shape and more than 10 harmonics are needed for approximate the solution. In table

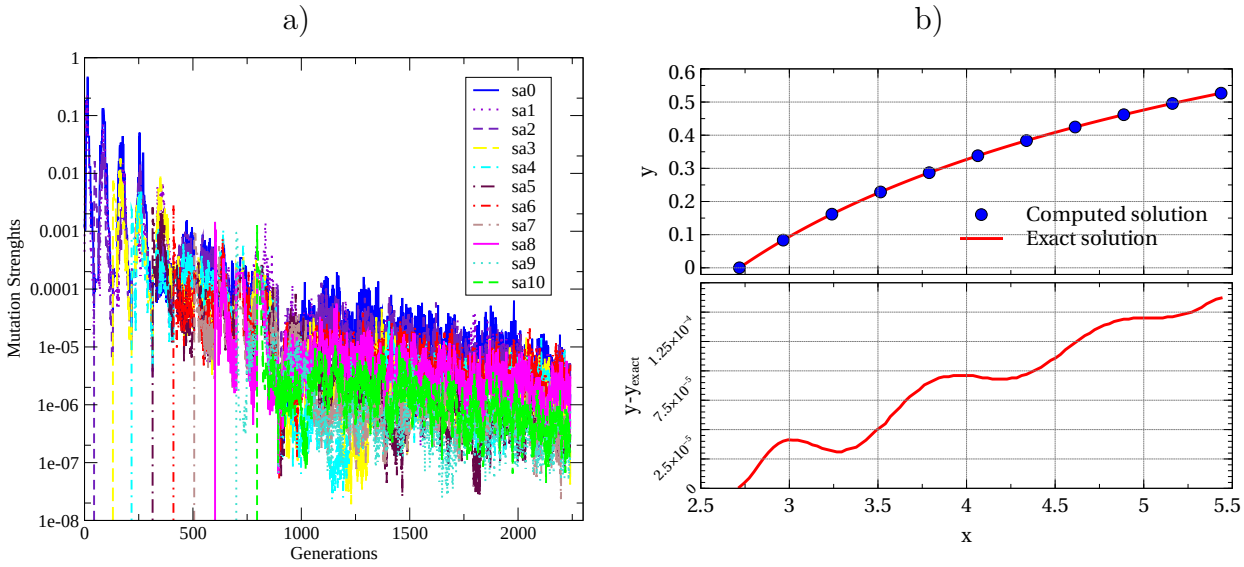

Figure 5.4: Mutation strengths of the best individual over the generations (a), comparison (b, top) and difference (b, bottom) between the exact solution and the computed one for one run of NLODE4 case using DESES algorithm. Mutation strength in (a) of *inactive* harmonics are considered out of the plot with a value close to 0. For the sake of clarity, in (b, top) figure, only 11 points are plot for the evolved solution, although 100 collocation points were taken into account.

Case	Max. Harmonic Order	Fitness	RMSE
LODE8	10	$1.51 \cdot 10^{-3}$	$1.22 \cdot 10^{-2}$
	20	$1.35 \cdot 10^{-4}$	$2.84 \cdot 10^{-3}$
	30	$1.84 \cdot 10^{-5}$	$9.24 \cdot 10^{-4}$
LODE9	10	$1.00 \cdot 10^{-2}$	$3.16 \cdot 10^{-2}$
	20	$8.32 \cdot 10^{-4}$	$7.08 \cdot 10^{-3}$
	30	$1.18 \cdot 10^{-4}$	$2.32 \cdot 10^{-3}$
LODE10	10	$1.46 \cdot 10^{2}$	$3.39 \cdot 10^{-2}$
	20	$3.27 \cdot 10^{-1}$	$1.34 \cdot 10^{-3}$
	30	$5.65 \cdot 10^{-3}$	$5.33 \cdot 10^{-4}$
NLODE6	10	$3.13 \cdot 10^{-2}$	$3.03 \cdot 10^{-1}$
	20	$1.80 \cdot 10^{-2}$	$2.17 \cdot 10^{-1}$
	30	$1.25 \cdot 10^{-2}$	$1.73 \cdot 10^{-1}$

Table 5.9: Harmonic number analysis for LODE8, LODE9, LODE10 and NLODE6 problems for DESES algorithm.

5.9 the fitness and RMSEs values running the algorithm with the same solver parameters (Table 5.7) but using 10, 20 and 30 harmonics are given. As expected, the accuracy is increased when more harmonics are used. Fig. 5.5 compares the solutions obtained for NLODE6 case with the numerical approximation using a numerical method. We can appreciate that due to the strong variation of the function near the origin 10 harmonics are not enough for a proper representation of the solution.

ODE3 discussion

Not as good results have been obtained for LODE3 problem, with a RMSE of several orders of magnitude bigger than the rest ODEs. Studying the differences between this problem and the others, we can see that the variation range of the derivatives appearing in the differential equation is around one order of magnitude higher than in the other cases:

$$\left.\begin{aligned} \frac{y'(1)}{y'(0)} &= 4e^3 \simeq 80 \\ \frac{y''(1)}{y''(0)} &= \frac{30}{12}e^3 \simeq 50 \end{aligned}\right\}. \tag{5.2}$$

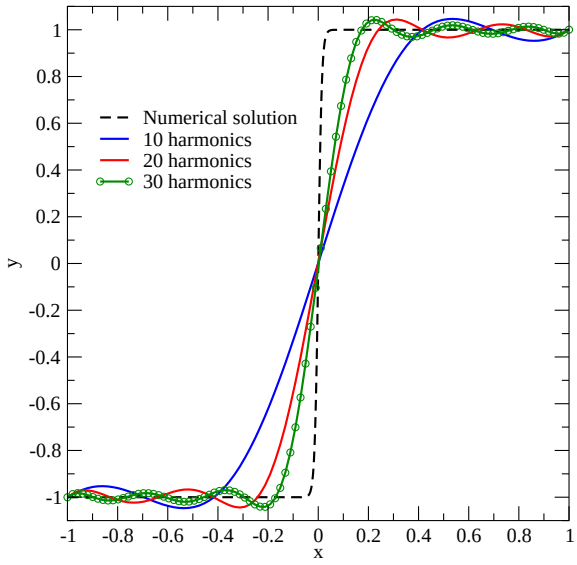

Figure 5.5: Comparison of the computed solution using 10, 20 and 30 harmonics with the numerical approximation for NLODE6 case for DESES algorithm.

In order to check if these ratios are the cause of the bad results, the same problem ODE3 has been solved using the same solver parameters (Table 5.7) but reducing the independent variable range from the original $x \in [0, 1]$ to $x \in [0, 0.2]$. In this way, the above ratios turn

$$\left.\begin{aligned} \frac{y'(0.2)}{y'(0)} &= \frac{3.2}{2}e^{0.6} \simeq 2.9 \\ \frac{y''(0.2)}{y''(0)} &= \frac{15.6}{12}e^{0.6} \simeq 2.4 \end{aligned}\right\}. \tag{5.3}$$

Algorithm performance has been recovered for this new problem, obtaining an average fitness of $5.48 \cdot 10^{-4}$ and an average RMSE of $1.02 \cdot 10^{-4}$.

Focusing in the original LODE3 problem, a study on the influence of the number of harmonic coefficients in the result quality has been performed. Several runs with an unlimited number of generations have been done. Results are shown in Table 5.10. As we can see the results improve when the number of harmonic coefficients is increased. More than 20 harmonics are needed for having a RMSE close to the rest of cases. Other important problem detected is the slow convergence rate, needing a higher number of generations.

Max. Harmonic Order	Fitness	RMSE	Generations
5	7.60	13.03	20053
10	1.49	4.45	194630
15	$2.40 \cdot 10^{-1}$	2.00	230060
20	$3.93 \cdot 10^{-2}$	$5.83 \cdot 10^{-1}$	380060

Table 5.10: Harmonic number analysis for original LODE3 problem using DESES algorithm.

5.2.3 DESCMA-ES Results

DESCMA-ES algorithm was run 50 times on the test problems given in Table 5.1 using different seeds for the random number generator and averages were taken. A total of 100 equidistant collocation points have been used in all cases, except in partial differential equations defined in a non-rectangular domain (PDE5 and PDE6), where the same points distribution used in [3] were imposed (77 and 80 points respectively) in order to facilitate a fair comparison.

Table 5.12 and 5.13 list the algorithm parameters which have been used in all test cases regarding the CMA-ES and DS methods. As it was already commented, all the defaults parameters recommended by Hansen [83] for CMA-ES are adopted, except the size of the population and the offspring number. Therefore, according to Eq. (4.24) and Eq. (4.32), the population size μ is related with the number of unknowns N:

$$\mu = \left\lfloor 6 + \frac{3}{2} \lfloor 3 \cdot \ln N \rfloor \right\rfloor . \tag{5.4}$$

Fig. 5.6 plots the relation between μ and N for all the test problems. The number of unknowns depends on the number of kernels n, the number of dependent variables m and the problem dimensionality d according to Eq. (4.24). Concretely speaking, the experimental criterion used to select the number of kernels was the following: 4 kernels for each dependent variable in LODEs, NLODEs and SODEs, and 8 in PDEs. Thus, for LODEs and NLODEs, the number of kernels was always 4. On the other hand, if the number of equations (dependent

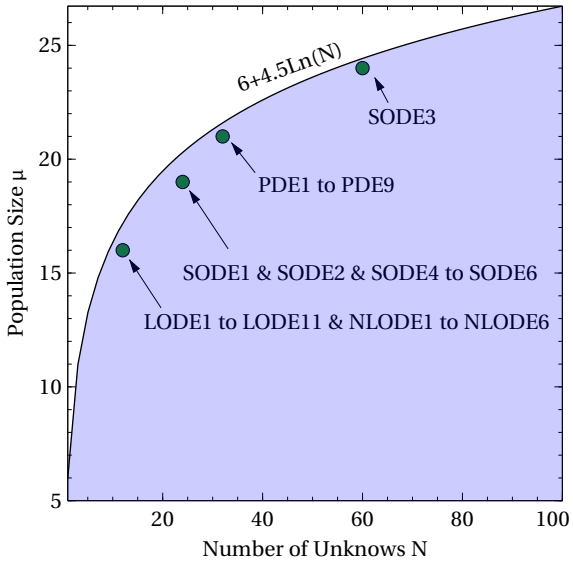

Figure 5.6: Population size μ of the DESCMA-ES algorithm according to the number of unknowns N. Dots mark the specific population used to solved the test problems.

variables) in a SODE is m, the number of kernels used was $4m$. Regarding PDEs, we can say that all of them have 2 independent variables, which make the problem harder to be solved compared with ODEs. Therefore, the number of kernels for PDEs has been doubled respect to ODEs. As we can see in Fig. 5.6, SODE3 has more unknowns than the other SODEs because 5 dependent variable must be solved ($m = 5$) instead of only two ($m = 2$) for the rest of SODEs. The centers are initialized randomly in an extended range controlled by a user parameter β according to Eq. (4.31).

Some parameters affect to both CMA-ES and DS algorithms: the boundary condition penalty φ is set to 300, the inner weighting factor κ used is 30, and the maximum number of fitness evaluations allowed is 10^6.

The results are given at Table 5.11 where data are presented grouped in two categories: those obtained at the end of the CMA-ES phase, and the final ones at the end of the DS method. Data are presented giving the average values and the standard deviations. By

Case	Centers n (see Eq. (4.24))	Fitness (CMA-ES)	Fitness (CMA-ES+DS)	RMSE (CMA-ES)	RMSE (CMA-ES+DS)	Fitness Eval.$\times 10^5$ (CMA-ES)	Fitness Evaluations$\times 10^5$ (CMA-ES+DS)
LODE1	4	$(8.73 \pm 34.5)\,10^{-8}$	$\mathbf{(1.18 \pm 2.87)\,10^{-9}}$	$(1.41 \pm 3.77)\,10^{-5}$	$\mathbf{(2.35 \pm 2.52)\,10^{-6}}$	1.08 ± 0.481	1.92 ± 0.473
LODE2	4	$(7.32 \pm 47.7)\,10^{-7}$	$\mathbf{(6.82 \pm 47.7)\,10^{-7}}$	$(2.70 \pm 14.4)\,10^{-5}$	$\mathbf{(2.14 \pm 14.4)\,10^{-5}}$	0.958 ± 0.749	1.77 ± 0.802
LODE3	4	$(1.42 \pm 8.87)\,10^{-1}$	$\mathbf{(3.47 \pm 17.5)\,10^{-4}}$	$(2.51 \pm 16.1)\,10^{-1}$	$\mathbf{(2.13 \pm 7.53)\,10^{-3}}$	0.691 ± 0.215	1.71 ± 0.312
LODE4	4	$(4.31 \pm 20.2)\,10^{-5}$	$\mathbf{(1.81 \pm 11.6)\,10^{-5}}$	$(2.08 \pm 9.50)\,10^{-4}$	$\mathbf{(1.03 \pm 6.20)\,10^{-4}}$	0.928 ± 0.656	1.64 ± 0.678
LODE5	4	$(4.45 \pm 0.222)\,10^{-7}$	$\mathbf{(7.32 \pm 0.373)\,10^{-10}}$	$(5.64 \pm 26.2)\,10^{-5}$	$\mathbf{(3.01 \pm 8.13)\,10^{-6}}$	1.04 ± 0.570	1.71 ± 0.723
LODE6	4	$(2.71 \pm 18.0)\,10^{-8}$	$\mathbf{(9.87 \pm 42.2)\,10^{-10}}$	$(1.07 \pm 5.75)\,10^{-6}$	$\mathbf{(1.72 \pm 2.85)\,10^{-7}}$	1.14 ± 0.786	1.78 ± 0.856
LODE7	4	$(4.42 \pm 17.9)\,10^{-6}$	$\mathbf{(2.39 \pm 9.47)\,10^{-6}}$	$(1.40 \pm 5.30)\,10^{-5}$	$\mathbf{(6.71 \pm 2.33)\,10^{-6}}$	2.77 ± 1.73	3.41 ± 1.65
LODE8	4	$(1.20 \pm 2.34)\,10^{-7}$	$\mathbf{(2.96 \pm 8.54)\,10^{-8}}$	$(9.07 \pm 8.41)\,10^{-5}$	$\mathbf{(4.87 \pm 5.41)\,10^{-5}}$	0.439 ± 0.161	1.43 ± 0.235
LODE9	4	$(1.01 \pm 0.403)\,10^{-1}$	$\mathbf{(9.70 \pm 4.26)\,10^{-2}}$	$(2.18 \pm 0.770)\,10^{-1}$	$\mathbf{(2.16 \pm 0.773)\,10^{-1}}$	0.825 ± 0.281	1.76 ± 0.405
LODE10	4	$(1.22 \pm 1.14)\,10^{3}$	$\mathbf{(9.28 \pm 10.2)\,10^{2}}$	$\mathbf{1.88 \pm 1.33}$	1.90 ± 1.41	0.522 ± 0.234	1.41 ± 0.502
LODE11	4	$(4.32 \pm 20.4)\,10^{-4}$	$\mathbf{(2.24 \pm 14.3)\,10^{-4}}$	$(1.40 \pm 5.41)\,10^{-3}$	$\mathbf{(9.16 \pm 39.4)\,10^{-4}}$	2.32 ± 1.44	2.78 ± 1.44
NLODE1	4	$(1.05 \pm 2.26)\,10^{-7}$	$\mathbf{(6.61 \pm 18.2)\,10^{-9}}$	$(6.40 \pm 8.70)\,10^{-5}$	$\mathbf{(1.61 \pm 2.12)\,10^{-5}}$	0.914 ± 0.446	1.88 ± 0.453
NLODE2	4	$(2.79 \pm 19.5)\,10^{-4}$	$\mathbf{(2.79 \pm 19.45)\,10^{-4}}$	$(4.84 \pm 33.8)\,10^{-4}$	$\mathbf{(4.83 \pm 33.8)\,10^{-4}}$	1.10 ± 0.589	1.77 ± 0.686
NLODE3	4	$(1.34 \pm 7.1)\,10^{-2}$	$\mathbf{(1.19 \pm 5.02)\,10^{-7}}$	$(4.83 \pm 24.2)\,10^{-3}$	$\mathbf{(2.05 \pm 5.04)\,10^{-6}}$	1.31 ± 0.542	2.26 ± 0.475
NLODE4	4	$(6.85 \pm 16.2)\,10^{-3}$	$\mathbf{(1.05 \pm 4.71)\,10^{-6}}$	$(4.78 \pm 10.4)\,10^{-2}$	$\mathbf{(8.10 \pm 39.9)\,10^{-3}}$	0.717 ± 0.392	1.78 ± 0.360
NLODE5	4	$(3.76 \pm 21.8)\,10^{-4}$	$\mathbf{(5.213 \pm 17.9)\,10^{-5}}$	$(2.24 \pm 7.48)\,10^{-4}$	$\mathbf{(8.51 \pm 12.1)\,10^{-5}}$	2.76 ± 2.13	3.42 ± 1.96
NLODE6	4	$(3.79 \pm 2.07)\,10^{-3}$	$\mathbf{(3.35 \pm 1.66)\,10^{-3}}$	$(5.30 \pm 3.53)\,10^{-1}$	$\mathbf{(5.32 \pm 3.57)\,10^{-1}}$	1.21 ± 0.649	2.10 ± 0.904
SODE1	4	$(1.41 \pm 7.66)\,10^{-9}$	$\mathbf{(1.16 \pm 6.74)\,10^{-9}}$	$(2.23 \pm 4.42)\,10^{-6}$	$\mathbf{(1.86 \pm 3.95)\,10^{-6}}$	3.66 ± 1.39	5.04 ± 1.41
SODE2	4	$(3.18 \pm 4.90)\,10^{-10}$	$\mathbf{(2.47 \pm 4.16)\,10^{-10}}$	$(1.61 \pm 1.23)\,10^{-6}$	$\mathbf{(1.40 \pm 1.12)\,10^{-6}}$	3.75 ± 1.41	4.98 ± 1.37
SODE3	4	$(9.21 \pm 30.0)\,10^{-9}$	$\mathbf{(9.13 \pm 30.0)\,10^{-9}}$	$(8.28 \pm 9.54)\,10^{-6}$	$\mathbf{(8.19 \pm 9.46)\,10^{-6}}$	9.54 ± 0.833	9.95 ± 0.175
SODE4	4	$(1.45 \pm 4.35)\,10^{-1}$	$\mathbf{(1.17 \pm 3.97)\,10^{-1}}$	$\mathbf{6.67 \pm 26.4}$	8.37 ± 31.7	4.49 ± 2.36	6.00 ± 2.12
SODE5	4	$(1.21 \pm 3.28)\,10^{-2}$	$\mathbf{(1.21 \pm 3.27)\,10^{-2}}$	$(2.88 \pm 7.79)\,10^{-2}$	$\mathbf{(2.88 \pm 7.78)\,10^{-2}}$	4.58 ± 2.67	5.89 ± 2.36
SODE6	4	$(2.07 \pm 1.39)\,10^{-2}$	$\mathbf{(1.34 \pm 1.14)\,10^{-2}}$	$(2.83 \pm 1.46)\,10^{-2}$	$\mathbf{(1.92 \pm 1.27)\,10^{-2}}$	1.47 ± 0.465	3.21 ± 0.734
PDE1	8	$(2.82 \pm 5.50)\,10^{-5}$	$\mathbf{(2.02 \pm 2.82)\,10^{-5}}$	$(1.98 \pm 1.60)\,10^{-4}$	$\mathbf{(1.75 \pm 1.14)\,10^{-4}}$	5.14 ± 1.63	7.10 ± 1.52
PDE2	8	$(1.02 \pm 4.36)\,10^{-5}$	$\mathbf{(8.77 \pm 36.1)\,10^{-6}}$	$(9.88 \pm 12.2)\,10^{-5}$	$\mathbf{(9.48 \pm 11.3)\,10^{-5}}$	6.85 ± 2.59	8.13 ± 1.80
PDE3	8	$(1.98 \pm 2.95)\,10^{-7}$	$\mathbf{(1.72 \pm 2.48)\,10^{-7}}$	$(1.19 \pm 0.963)\,10^{-5}$	$\mathbf{(1.09 \pm 0.846)\,10^{-5}}$	6.41 ± 1.78	8.03 ± 1.40
PDE4	8	$(2.44 \pm 4.90)\,10^{-4}$	$\mathbf{(2.03 \pm 3.53)\,10^{-4}}$	$(7.31 \pm 5.80)\,10^{-4}$	$\mathbf{(7.02 \pm 5.28)\,10^{-4}}$	5.16 ± 1.96	7.02 ± 1.72
PDE5	8	$(7.56 \pm 19.8)\,10^{-4}$	$\mathbf{(6.26 \pm 15.4)\,10^{-4}}$	$(8.07 \pm 7.97)\,10^{-4}$	$\mathbf{(7.53 \pm 6.39)\,10^{-4}}$	7.79 ± 2.29	8.94 ± 1.63
PDE6	8	$(3.12 \pm 8.20)\,10^{-4}$	$\mathbf{(1.96 \pm 5.01)\,10^{-4}}$	$(5.76 \pm 7.37)\,10^{-4}$	$\mathbf{(4.75 \pm 5.26)\,10^{-4}}$	4.71 ± 1.41	6.70 ± 1.40
PDE7	8	$(2.25 \pm 4.15)\,10^{-3}$	$\mathbf{(2.07 \pm 4.11)\,10^{-3}}$	$(1.60 \pm 1.39)\,10^{-2}$	$\mathbf{(1.55 \pm 1.37)\,10^{-2}}$	3.97 ± 1.96	5.80 ± 1.86
PDE8	8	$(6.44 \pm 12.7)\,10^{-1}$	$\mathbf{(4.36 \pm 7.63)\,10^{-1}}$	$(5.77 \pm 6.11)\,10^{-2}$	$\mathbf{(5.15 \pm 5.29)\,10^{-2}}$	3.78 ± 2.10	5.62 ± 1.88
PDE9	8	$(1.22 \pm 1.53)\,10^{-5}$	$\mathbf{(7.69 \pm 8.18)\,10^{-6}}$	$(3.77 \pm 2.64)\,10^{-4}$	$\mathbf{(3.47 \pm 2.45)\,10^{-4}}$	4.01 ± 1.07	6.01 ± 1.07

Table 5.11: Experimental results of DESCMA-ES algorithm. Each case was run 50 times using different seeds for the random number generator. In columns "Fitness" and "RMSE", the best values obtained are highlighted in bold letter.

Parameter	Values
Initial weights	Randomly in $w_i \in [-0.01, 0.01]$
Initial γ_i	Randomly in $\gamma_i \in (0, 1]$
Initial centers c_{ik}	Randomly using Eq. (4.31) and $\beta = 2$
Initial mutation step σ	0.01
Offspring number	$\lambda = 3 \cdot \lambda_{default}$ (see Eq. (4.32))
Population size	$\mu = \lfloor \lambda/2 \rfloor$
Stop criteria	Default (see [83])

Table 5.12: CMA-ES parameter values.

Parameter	Values
Number of restarts	10
Increment for first simplex	$\triangle = 10^{-2}$. Eq. (4.33).
Stop criterion	Fitness evaluations $= 2 \cdot 10^4$
Parameter convergence criterion	10^{-20}
Target convergence criterion	10^{-20}

Table 5.13: DS parameter values.

construction of the local search algorithm (see Section 4.4.3), its results (column marked as "Fitness CMA-ES+DS") always outperform the fitness values of the CMA-ES algorithm (column marked as "Fitness CMA-ES"). In some cases, the DS phase improves significantly the fitness value by several orders of magnitude as in LODE3, LODE5, LODE6, NLODE3, NLODE4 or SODE4 problems. In other cases, the solution obtained by the CMA-ES algorithm is good enough and the improvements obtained with the DS phase are less important. In relation to RMSE values, the behavior is similar. Only in a few cases that value slightly gets worse when the DS phase is applied. However, note that the value RMSE is never used by the evolutionary algorithm or by local search method in the optimization process. Therefore, from both points of view (fitness and RMSE), we can say that the use of the DS algorithm allows us to improve (or maintain) the results obtained by the evolutionary algorithm for all the problems, without modifying the control parameters, and with a limited cost, as we can see comparing the number of fitness evaluation before and after the DS phase (two last columns of the mentioned table).

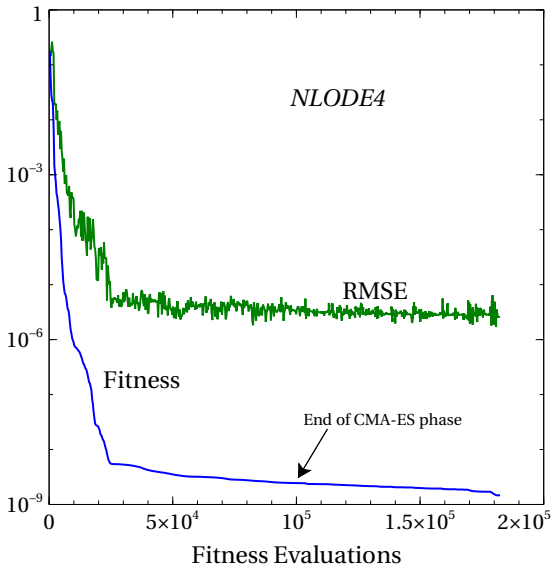

Figure 5.7: A typical run of DESCMA-ES algorithm for NLODE4 showing the fitness value and RMSE evolution with the number of fitness evaluations.

As an example, Fig. 5.7 to Fig. 5.11 show how DESCMA-ES algorithm behaves when it is applied to two particular problems. The first four figures refers to NLODE4 equation and the fifth is from PDE8. Thus, in Fig. 5.7 a typical run for NLODE4 problem showing the evolution of fitness function and the RMSE versus the number of fitness evaluations is provided. Fig. 5.8 shows the evolution of the Gaussian kernels for the same run of NLODE4. Note how a big variation is observed at the first generations, where the CMA-ES explores the solution space. Afterwards the rate of change decreases because CMA-ES and DS perform an exploitation of the best solutions. Fig. 5.9 shows the approximated solution of NLODE4 as an addition of the four Gaussian kernels that form the solution. Note that in this particular case the four centers c_i are outside the independent variable range, and that one kernel has a negative γ value. Fig. 5.10 shows a comparison between the exact and the approximated solutions of NLODE4. Finally, Fig. 5.11 shows a comparison between the approximated solution found by the proposed algorithm versus the exact one for the *wave* equation (PDE8). The solution represents the vibration of a string in his fundamental harmonic scale. Only

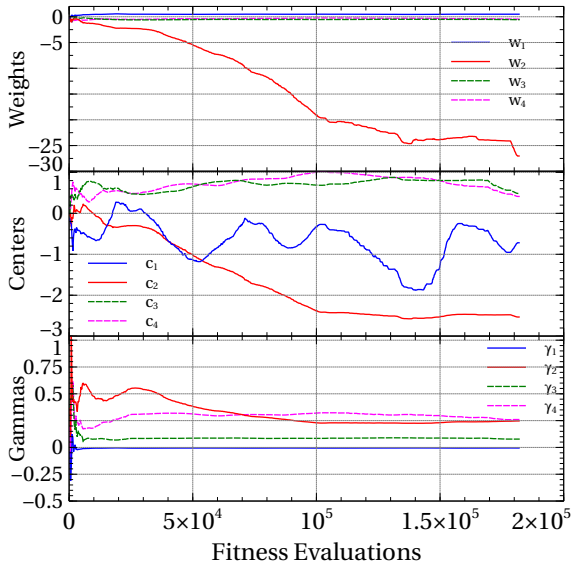

Figure 5.8: A typical run of NLODE4 using DESCMA-ES algorithm showing the evolution of the Gaussian kernels: w_i, c_i and γ_i.

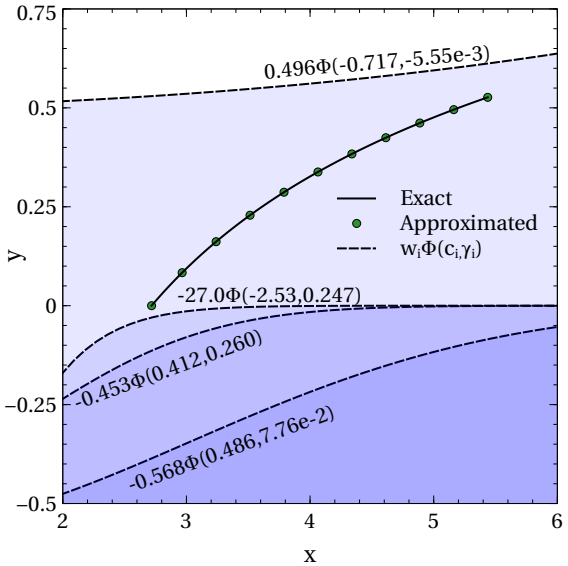

Figure 5.9: Final evolved solution obtained in a typical run of DESCMA-ES algorithm for NLODE4 and constrained to range $[e, 2e]$. It is shown as well the 4 Gaussian kernels which form the approximated solution.

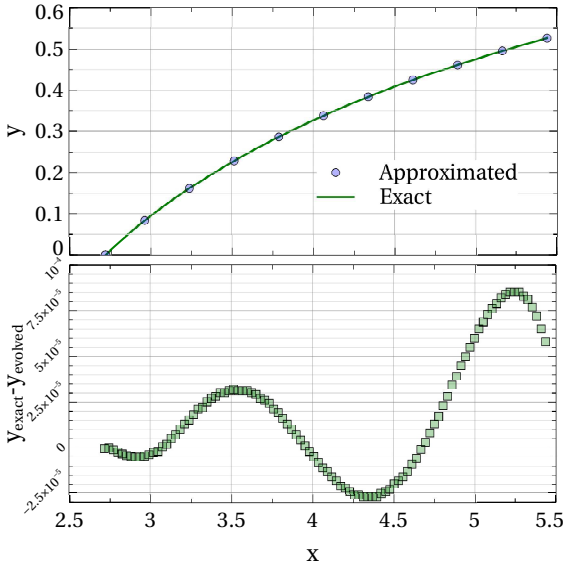

Figure 5.10: Comparison between the final solution obtained in a typical run of DESCMA-ES algorithm for NLODE4 with the exact one (up) and error between the evolved solution and the exact one (down). For the sake of clarity, in the top figure only 11 points are plot for the evolved solution, although 100 collocation points were taken into account. Note the order of magnitude of the error in second figure.

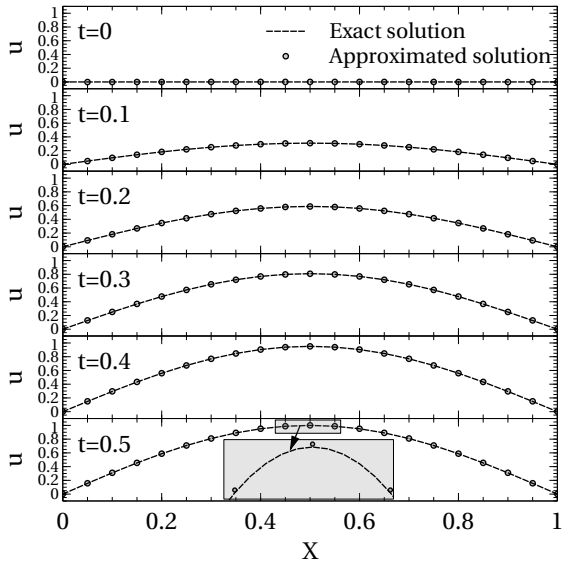

Figure 5.11: Comparison of a typical solution obtained by the DESCMA-ES algorithm with the exact one for PDE8 (*Wave* equation).

5 time instants are plotted for t from 0 to 0.5, although the equation is solved in the range $t \in [0, 1]$. As we can observe in the close up view at the bottom figure, the solution evolved by the algorithm is in good agreement with the exact one.

Sensitivity to the number of kernels and the inner weighting factor (κ)

The solutions of the majority of the test problems present fitness values with orders of magnitude below of 10^{-3} and, in some cases, achieving values below of 10^{-9} as we can see in Table 5.11. In this section we want to study the effect of increasing the number of kernels in the solution quality. For that, we select, for instance, all the problems whose order of magnitude associated to the fitness value is higher than 10^{-3}. We will vary the number of kernels but the rest of the control parameters described in Tables 5.12 and 5.13 are maintained.

As we see in Table 5.14, better values of fitness are obtained when the number of kernels is increased for all the cases, except for SODE4 problem, where all the fitness values in the table are very similar (same order of magnitude). However, for this particular problem, increasing the number of kernels from 4 (Table 5.11) to 6 (Table 5.14) is enough to obtain fitness values with order of magnitude from 10^{-1} to 10^{-9}. On the other hand, the value of RMSE also improves or is approximately maintained when the number of kernels grows. Here, it is important newly to mention that the RMSE value is not used in the evolutionary algorithm or the local search method. Therefore, from the evidence here presented, we can say that when the solution evolved is not as accurate as desired we should increase the number of kernels to improve such accuracy. Finally, in relation to the average number of fitness evaluation, obviously it increases as the size of the problem does with the number of kernels.

As it was already commented, the inner parameter κ was set to 30 for all the cases. It is not observed a high sensitivity in the results to the inner weighting factor κ except for NLODE6 case. As we see in Fig. 5.12, this equation is hard to solve because has a strong gradient in the origin. For example, when $\kappa = 0$, the algorithm is not capable to locate the appropriate transition from -1 to 1 at the origin. When an appropriate value of κ is set ($\kappa = 30$), we can observe how the quality of the solution only depends on the number of

Case	**Centers** n	**Unknowns N** Eq. (4.24)	**Population** μ Eq. (5.4)	**Fitness (CMA-ES+DS)**	**RMSE (CMA-ES+DS)**	**Fit. Evaluations (CMA-ES+DS)**
LODE9	6	18	18	$(4.14 \pm 5.46)\,10^{-2}$	$(1.04 \pm 1.08)\,10^{-1}$	$(3.36 \pm 0.843)\,10^{5}$
	10	30	21	$(2.48 \pm 16.3)\,10^{-3}$	$(1.08 \pm 3.48)\,10^{-2}$	$(5.90 \pm 1.05)\,10^{5}$
	14	42	22	$\mathbf{(7.44 \pm 43.7)\,10^{-6}}$	$\mathbf{(4.49 \pm 20.4)\,10^{-4}}$	$(9.44 \pm 0.80)\,10^{5}$
LODE10	6	18	18	$(1.18 \pm 3.66)\,10^{2}$	$(2.26 \pm 6.14)\,10^{-1}$	$(3.03 \pm 1.50)\,10^{5}$
	10	30	21	$(4.72 \pm 23.1)\,10^{-3}$	$(5.18 \pm 9.21)\,10^{-4}$	$(6.15 \pm 1.40)\,10^{5}$
	14	42	22	$\mathbf{(1.32 \pm 5.04)\,10^{-3}}$	$\mathbf{(4.54 \pm 6.74)\,10^{-4}}$	$(9.27 \pm 1.06)\,10^{5}$
NLODE6	6	18	18	$(1.87 \pm 1.43)\,10^{-3}$	$(4.12 \pm 3.54)\,10^{-1}$	$(3.88 \pm 1.36)\,10^{5}$
	10	30	21	$(1.30 \pm 1.36)\,10^{-3}$	$(3.05 \pm 2.99)\,10^{-1}$	$(6.19 \pm 1.14)\,10^{5}$
	14	42	22	$\mathbf{(6.26 \pm 8.50)\,10^{-4}}$	$\mathbf{(2.02 \pm 2.74)\,10^{-1}}$	$(8.89 \pm 1.15)\,10^{5}$
SODE4	6	36	21	$\mathbf{(2.18 \pm 5.89)\,10^{-9}}$	$\mathbf{(2.76 \pm 3.23)\,10^{-6}}$	$(8.15 \pm 2.01)\,10^{5}$
	10	60	24	$(3.58 \pm 5.56)\,10^{-9}$	$(4.47 \pm 3.13)\,10^{-6}$	$(9.55 \pm 1.15)\,10^{5}$
	14	84	25	$(6.57 \pm 8.52)\,10^{-9}$	$(6.38 \pm 4.45)\,10^{-6}$	$(9.49 \pm 1.09)\,10^{5}$
SODE5	6	36	21	$(2.63 \pm 12.9)\,10^{-9}$	$(3.53 \pm 3.98)\,10^{-6}$	$(7.86 \pm 1.80)\,10^{5}$
	10	60	24	$(6.26 \pm 11.7)\,10^{-10}$	$\mathbf{(2.28 \pm 1.95)\,10^{-6}}$	$(8.83 \pm 1.40)\,10^{5}$
	14	84	25	$\mathbf{(6.22 \pm 5.37)\,10^{-10}}$	$(2.38 \pm 1.24)\,10^{-6}$	$(8.76 \pm 1.51)\,10^{5}$
SODE6	6	36	21	$(1.43 \pm 0.208)\,10^{-2}$	$(2.11 \pm 0.288)\,10^{-2}$	$(4.29 \pm 0.77)\,10^{5}$
	10	60	24	$(1.02 \pm 0.303)\,10^{-2}$	$(1.64 \pm 0.374)\,10^{-2}$	$(7.30 \pm 1.91)\,10^{5}$
	14	84	25	$\mathbf{(8.70 \pm 4.88)\,10^{-3}}$	$\mathbf{(1.38 \pm 0.556)\,10^{-2}}$	$(8.92 \pm 1.34)\,10^{5}$
PDE7	6	24	19	$(7.95 \pm 11.7)\,10^{-2}$	$(2.98 \pm 2.52)\,10^{-2}$	$(3.87 \pm 2.15)\,10^{5}$
	10	40	22	$(7.80 \pm 10.2)\,10^{-4}$	$(9.88 \pm 10.0)\,10^{-3}$	$(7.73 \pm 1.95)\,10^{5}$
	14	56	24	$\mathbf{(2.78 \pm 4.02)\,10^{-4}}$	$\mathbf{(6.19 \pm 6.41)\,10^{-3}}$	$(9.56 \pm 1.04)\,10^{5}$
PDE8	6	24	19	1.20 ± 1.90	$(1.00 \pm 0.729)\,10^{-1}$	$(3.38 \pm 1.64)\,10^{5}$
	10	40	22	$(1.97 \pm 6.39)\,10^{-2}$	$(7.00 \pm 14.3)\,10^{-3}$	$(8.75 \pm 1.60)\,10^{5}$
	14	56	24	$\mathbf{(8.23 \pm 14.8)\,10^{-4}}$	$\mathbf{(1.50 \pm 1.90)\,10^{-3}}$	$(9.95 \pm 0.215)\,10^{5}$

Table 5.14: Effect of increasing the number of centers in DESCMA-ES algorithm. In columns "Fitness" and "RMSE" the best values obtained are highlighted in bold letter.

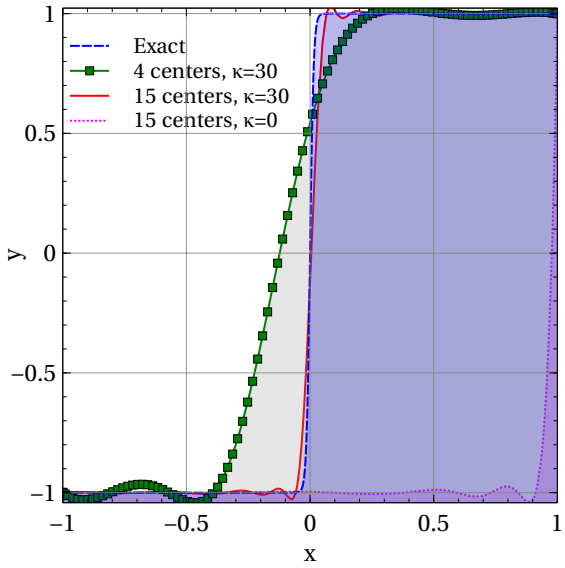

Figure 5.12: Comparison of evolved solutions by DESCMA-ES algorithm with the exact one for NLODE6 equation.

centers, increasing the first when the second does.

5.3 Comparisons with other Methods

For the sake of clarity, this section is only devoted to compare the two best algorithms presented (DESES and DESCMA-ES) with other algorithms, such as other heuristic approaches or more traditional numerical methods. A comparison between both algorithms is also provided. Nevertheless, in Section 5.2.1 where the results of DESGE algorithm were provided, a brief comparison and discussion of this algorithm with works [1, 2] were also commented.

5.3.1 Comparing of DESES algorithm with Numerical Methods

Maybe the greater advantage of the proposed DESES approach is that it is a generic framework, i. e., it does not depend on the type of differential equation. On the other hand, numerical methods are specific for each equation type. For instance, classical Runge-Kutta methods are used for initial value problems due to their higher accuracy features compared with more efficient algorithms as Euler's method. Classical Runge-Kutta methods are ex-

plicit, and are unsuitable for stiff systems because of their small region of stability. On the contrary, implicit Runge-Kutta methods have a large region of absolute stability [7]. Boundary value problems requires different algorithms, such as shooting method for one-dimensional problems or the finite element method for more general domains. Even for the same equation, in some problems depending on the boundary conditions the numerical scheme must be changed due to stability reasons [104]. Furthermore, the implementation of a new numerical method could turn difficult because it is necessary to take into account several issues as the discretization order, the algorithm stability, the convergence speed, how to fulfill the boundary conditions, etc. In the methods described in this thesis, the original problem is transformed into an optimization one according to Eq. (4.1), so the problem of choosing the most appropriate numerical method disappears.

For making a quantitative comparison with a numerical method, two non-linear ordinary differential equations from the test case suite (Tables 5.1 and 5.2) have been solved with traditional numerical methods. The first problem, NLODE2 is a first order non-liner differential equation. The domain interval $x \in [1, 2]$ is discretized into N equidistant nodes. This system can be solved in a very efficient way using a fourth order Runge-Kutta (RK4) algorithm [102]. Once the solution is obtained, the RMSE is computed according to Eq. (5.1). The RMSE can be computed on different grids using a liner interpolation obtaining the values of the dependant variable y_i on each grid node x_i. In table 5.15 the results obtained for two different grid sizes (10^2 and 10^3 nodes) for the numerical method are compared with the evolutionary approach (using 10^2 collocation points). The RMSE are computed from the obtained solutions in each case on three different grid sizes (10^2, 10^3 and 10^4 nodes).

We can see that the RMSE of evolutionary solution is not dependant on the grid size. The RK4 solutions have a high accuracy in the grid used in the computation, but this accuracy decreases when other grid is used due to the linear interpolation. The RK4 algorithm with the same grid size as the evolutionary solution has less accuracy in the finer grids (10^3 and 10^4 nodes). As expected, if the number of nodes is increased, the accuracy achieved is higher and better than the evolutionary approach. It is worth to point out that the solution

Method	**Grid size**	$RMSE_{10^2}$	$RMSE_{10^3}$	$RMSE_{10^4}$
RK4	10^2	$2.23 \cdot 10^{-12}$	$8.91 \cdot 10^{-6}$	$8.91 \cdot 10^{-6}$
RK4	10^3	$8.71 \cdot 10^{-8}$	$3.80 \cdot 10^{-15}$	$8.75 \cdot 10^{-8}$
DESES	10^2	$5.98 \cdot 10^{-6}$	$5.99 \cdot 10^{-6}$	$5.99 \cdot 10^{-6}$

Table 5.15: Comparison of the numerical method solution with DESES algorithm for NODE2 case. Two grid sizes of 10^2 and 10^3 have been used in the numerical method. The RMSE are computed on three grid sizes using a linear interpolation for the numerical solution and the equation (4.6) for the evolutionary one.

representation of the evolutionary approach only needs to store 11 numbers (the harmonic coefficients), whereas the RK4 solution requires to store the values of x and y in all the grid nodes, i. e. the RK4 solution in the finer grid is around 200 times bigger than the evolutionary solution.

For the next comparative NLODE5 has been selected. It is as well a non linear differential equation, but the boundary conditions are given in different points. This type of cases is called two point boundary problems. The crucial distinction between initial value problems and two point boundary value problems is that in the former case we are able to start an acceptable solution at its beginning, while in the present case, the boundary conditions at the starting point do not determine a unique solution to start with. For this reason, two point boundary value problems require considerably more effort to solve than do initial value problems. The *shooting method* [102] has been used to solve this problem. In this method the original problem is transformed into a root finding problem. We choose values for all of the dependent variables at one boundary. We then integrate the ODE by initial value methods, arriving at the other boundary. We find discrepancies from the desired boundary values there, so the initial boundary condition is readjusted. The iteration process is stopped when no improve is detected in the error of the boundary condition at the left side of the domain interval. Each iteration is solved using a RK4 algorithm transforming the initial two order equation into the following equivalent first order system:

Method	**Grid size**	$RMSE_{10^2}$	$RMSE_{10^3}$	$RMSE_{10^4}$
Shooting	10^2	$3.92 \cdot 10^{-7}$	$5.67 \cdot 10^{-5}$	$5.68 \cdot 10^{-5}$
Shooting	10^3	$5.54 \cdot 10^{-7}$	$3.89 \cdot 10^{-10}$	$5.60 \cdot 10^{-7}$
DESES	10^2	$3.12 \cdot 10^{-6}$	$3.13 \cdot 10^{-6}$	$3.14 \cdot 10^{-6}$

Table 5.16: Comparison numerical method solution with DESES algorithm for NODE5 case. Two grid sizes of 10^2 and 10^3 have been used in the numerical method. The RMSE is obtained using a linear interpolation for the numerical solution and the equation (4.6) for the evolutionary one in three different grids of 10^2, 10^3 and 10^4 nodes.

$$\left.\begin{aligned} y' &= z \\ z' &= \frac{yz}{x \sin x^2} - 4x^2 \sin x^2 \end{aligned}\right\}, \tag{5.5}$$

Table 5.16 shows a comparative of the solutions obtained using two grid sizes. As before, the RMSE values are computed in three different grids using a linear interpolation in the numerical solutions, and equation (4.6) for the evolutionary one.

We see that as in the previous case a finer grid is needed for achieving a better RMSE value than in the evolutionary approach. When the RMSE is computed in a different grid than the one used in the *Shooting* algorithm, the errors increase. On the other hand, the RMSE values of the Evolutionary solutions are not affected by the grid size. Note that the numerical integration is performed using a fourth order Runge-Kutta algorithm, which is a high order method, i. e. the errors depends on the grid size rise to the power of four. The memory requirements of the *Shooting* solution using 10^3 nodes is around 200 times bigger than the evolutionary solution (arrays of x and y values must be stored, meanwhile only the harmonic coefficients must be kept in evolutionary solution).

With these two examples, we can say that in some cases the evolutionary approach can achieve a more accurate solution using less number of nodes. The solutions obtained are coded in a more compact way requiring significantly less amount of memory. Nevertheless, a major drawback is the CPU time consuming. In this comparison, the Evolutionary approach consumes around 5000 more time than the numerical approach. This number could be

decreased in other problems where efficient numerical methods as RK4 and *Shooting* can not be applied, as for instance in PDEs.

5.3.2 Comparison of DESES algorithm with other Evolutionary Computing approaches

It is difficult to make a quantitative comparative study with other reported approaches. Although the same problems described in [2] have been used with the same collocation points, the comparative is not straightforward because, as it was already commented previously, the fitness values are high dependent on the solver parameters and on how differential equations are provided to expression (4.1). A correct comparison of the solution quality should be done with the RMSE values, but these quantities are not reported in previous contributions. However, Tsoulos et al. [2] obtained an average fitness values between 10^{-5} and 10^{-9}, meanwhile in the present work the fitness values are in the range of 10^{-2} and 10^{-8}. Therefore it seems that neural networks can approximate the solution functions with better accuracy. Moreover, no problems have been reported for ODE3 case in [2, 1]. It is important to notice that in the present contribution all the test cases have been run with the same solver parameters in order to present a systematic method. However, better results could be obtained if these solver parameters were adjusted by trial and error for each problem.

In contribution [17] a PDE using complex numbers on a triangle shape domain is reported. The RMSE is computed approximating the exact solution by a numerical method solution. The RMSE obtained is around 10^{-4}, which is better than those obtained by DESES. Nevertheless, the number of unknowns that must be tuned in [17] is much higher than in DESES algorithm. A total of 132 neurons are needed, so considering the weights and bias, this implies a number of unknowns around 260 against 100 of the present contribution. In a similar way, good accuracy has been obtained for ODE9 using 30 harmonics, whereas in [18] more than 70 neurons are needed for obtaining a similar solution.

It can be remarked that in some other previous contributions [12, 2, 1] PDE's results are of the same quality than ODEs. In DESES algorithm, the PDE's RMSE is around two orders

of magnitude bigger compared with ODEs and SODEs. This could be explained noting that the number of unknown coefficients needed by a neural network for solving a PDE problem scale linearly with the dependent variable dimension. On the other hand, using an harmonic approach, the number of harmonics increases quadratically for 2D problems, as it was shown in expression (4.15).

Because local search is used in other approaches [12, 14, 2], but not in DESES algorithm, it is difficult as well to make a quantitative comparison of the required computational power. It should be compared not the generation number, but the number of fitness evaluations. But these quantities have not been reported in the approaches mentioned. Furthermore, in DESES, the fitness evaluation number could be reduced taking into account that is cheaper to evaluate the fitness function when the number of *active* harmonics is lower in the first *ES steps* than in the last *ES steps*.

From a qualitative point of view, some advantages of DESES algorithm can be enumerated. Firstly, in our algorithm is straightforward to compute the derivatives because all the solutions are only expressed as sum of cosine functions. In works based on GP [27, 3, 1], an automatic differentiation engine must be used. In neural networks approaches [12, 17, 2, 14, 18], the activation function must be differentiated, which could be hard depending on the chosen function. Furthermore, if more than one hidden layer is used, the symbolic derivatives implementation could turn very complex.

Secondly, in DESES, several steps of a classical ES are used, guiding the search process in a more efficient way. Other approaches [12, 14, 2] use local search, which makes the implementation and analysis more difficult. Nevertheless, the proposed approach can deal naturally with local search phases, but we have wanted to test the skills of our method in the simplest way possible. In any case, the addition of this kind of search could accelerate the convergence velocity.

Thirdly, low dispersion in the results has been observed, so this finding provides evidence about the robustness of our method. Works based on GP reported a higher dispersion.

And finally, DESES algorithm can be applied to different kind of differential equations. Some authors [12, 3, 36] use some particular methods for dealing with the boundary condi-

tions, facilitating the optimization process eliminating the constraints. Nevertheless, these methods are problem dependant and can not be applied to all problems. DESES does not assume any particular structures in the boundary conditions. That is, it is straightforward to assign a fitness value to each individual in the population transforming the original problem into an optimization one according to Eq. (4.1), even in those problems with complex geometries and boundary conditions, such as PDE6 (see Table 5.2) where the definition domain is more complex than a simple 2-dimensional interval.

5.3.3 Comparison of DESCMA-ES with Numerical Methods

In this subsection a comparison of the proposed DESCMA-ES algorithm with numerical methods is presented. As in Section 5.3.1, our intention here is not to give an exhaustive comparison with this kind of methods. Numerical methods are much more mature than evolutionary methods, can cope with a great variety of difficult problems and are faster than evolutionary approaches. Therefore, we focus the comparison from the point of view that numerical methods are usually specific for each type of equation while an evolutionary algorithm could cope with different types of differential equations if they are transformed into an optimization problem. An example of this is analyzed here. For that, two partial differential equations are chosen: PDE1 (*Poisson* equation) and PDE8 (*Wave* equation). Although both equations are of second order, the former is *elliptic* and the later *hyperbolic*. This fact causes that different numerical methods should be employed to solve them. As we will show in the next paragraphs, the proposed evolutionary method can cope with the two problems without any change in the algorithm or in the user parameters.

A brief description of two simple finite difference numerical methods used in the comparison is provided. The first method is an explicit four order Runge-Kutta (RK4) [102]. Being the equation $\Psi_{xx} + \Psi_{yy} = f(x, y)$, after a discretization on the grid nodes and using central differences, the residual at point (i, j) of a candidate solution field Ψ is computed as

$$R_{i,j}(\Psi) = \frac{\Psi_{i-1,j} - 2\Psi_{ij} + \Psi_{i+1,j}}{\triangle x^2} + \frac{\Psi_{i,j-1} - 2\Psi_{ij} + \Psi_{i,j+1}}{\triangle y^2} - f(x_i, y_j), \tag{5.6}$$

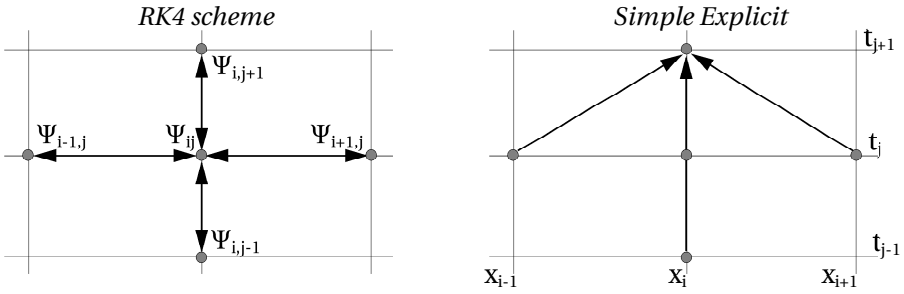

Figure 5.13: Numerical schemes for comparison. Arrows indicate that one node is affecting to another. Note that at Runge-Kutta method, point (i, j) affects to all his neighbours. On the contrary, at Explicit method, all nodes at time instant t_{j+1} does not affect to previous time instants t_j and t_{j-1}.

where constant grid sizes $\triangle x$ and $\triangle y$ are assumed. At the boundary condition points, the residual is set to 0. Defining a pseudo time parameter τ which controls the rate of convergence and the stability of the method, the RK4 consist on the following sequence:

$$\left.\begin{aligned} \Psi_0 &= \Psi \\ \Psi_1 &= \Psi + \tau R\left(\Psi_0\right)/4 \\ \Psi_2 &= \Psi + \tau R\left(\Psi_1\right)/3 \\ \Psi_3 &= \Psi + \tau R\left(\Psi_2\right)/2 \\ \Psi &= \Psi_3 \end{aligned}\right\} \tag{5.7}$$

The above sequence is repeated after some convergence criterion is fulfilled, typically a maximum number of iterations or when the norm of the residual array R is below a predefined tolerance. We have used 2000 iterations and $\tau = 10^{-3}$ for PDE1. At Fig. 5.13a we can see a schematic of the numerical scheme.

Some modifications must be done to the numerical scheme to solve *Wave* equation. Thus the coefficient of the discretization of the second derivative u_{tt} in Eq. (5.6) changes the sign. Boundary conditions must be applied in a different way because not only they are set on u, but as well on its first derivative u_t. However, this method does not work for Wave equation because the algorithm is not stable, even for very low values of τ.

To solve the PDE8, a different approach must be followed. A simple explicit scheme can be used. Being the equation $u_{tt} = u_{xx}$, and using as well central differences, the field can be computed in a straightforward way in just a single iteration:

$$u_{i,j+1} = -u_{i,j-1} + 2\left(1-\alpha^2\right)u_{ij} + \alpha^2\left(u_{i+1,j} + u_{i-1,j}\right), \tag{5.8}$$

where $\alpha = \triangle t/\triangle x$. Note that the first and second time instant ($u_{i,0}$ and $u_{i,1}$) must be obtained from the boundary conditions. At Fig. 5.13b the numerical scheme is outlined. In the same way than with RK4 algorithm, some changes must be applied on Eq. (5.8) and in the boundary condition to solve the Poisson equation.

Table 5.17 shows the results of solving PDE1 and PDE8 with the evolutionary algorithm and with both numerical methods. As we can observe, the numerical methods are specific for only one type of equation and can not cope with different problems. On the other hand, evolutionary algorithm works properly both for elliptic and parabolic equations. When the numerical schemes converge, the RMSE values are on the same order of magnitude. However, the elapsed time required by the numerical methods is several orders of magnitude lower.

Other advantage of DESCMA-ES algorithm, like DESES, is that it has lower memory requirements: using 8 centers, a total of 32 unknowns must be stored. Numerical methods must be stored as many values as collocation points, 100 in these examples. Therefore the solution requires 3 times less memory. And what is more important, the solution is symbolically stored, so new solution values different from the collocation points can be obtained without performing any interpolation.

It is important to highlight that this comparison has to be taken as an example. It is probable that a more complex numerical method, for instance an implicit one, could solve

Case	Method	RMSE	Iterations/ Fitness Eval.	Absolute Elapsed time	Relative Elapsed time
PDE1	Evolutionary	$8.3 \cdot 10^{-6}$	$1.4 \cdot 10^{5}$	$68s$	$8.1 \cdot 10^{-2}$
	RK4	$6.5 \cdot 10^{-5}$	$8.0 \cdot 10^{3}$	$0.2s$	$2.4 \cdot 10^{-4}$
	Simple Explicit	1.1	1	$0.003s$	$3.6 \cdot 10^{-6}$
PDE8	Evolutionary	$7.8 \cdot 10^{-3}$	$4.2 \cdot 10^{5}$	$14\,minutes$	1
	RK4	∞	∞	∞	∞
	Simple Explicit	$4.7 \cdot 10^{-3}$	1	$0.004s$	$4.7 \cdot 10^{-6}$

Table 5.17: Comparison of different algorithms for PDE1 and PDE8.

both equations as the evolutionary approach does. Here we only want to give some experimental feedback about how an evolutionary approach could be more flexible and with a more straightforward setup than a numerical method.

5.3.4 Comparison of DESCMA-ES with DESES and other Evolutionary Algorithms

As it was commented in Section 5.3.2, it is difficult to make a quantitative comparative study with other reported approaches because the fitness values are high dependent on the solver parameters and on how differential equations are provided to Eq. (4.26). A correct comparison of the solution quality should be done with the RMSE values, but these quantities generally are not reported in previous contributions. As discussed in Section 5.2.3, the direct way to increase the solution accuracy in DESCMA-ES method depends strongly on the number of kernels used. In order to measure the performance of our approximation in relation to other methods, we maintained here the set-up used previously, Tables 5.12 and 5.13, except the number of kernels. That is, when the RMSE value obtained in Table 5.11 was lower than the best result obtained by one of the techniques of the comparison, we increased the number of kernels to investigate the potential of our method. In this regard, we also reused the results obtained in Table 5.14.

Problem	RMSE in [3]	RMSE in DESES	RMSE in [4]	RMSE in [5]	RMSE in DESCMA-ES| centers n used
LODE1	-	$(3.70 \pm 0.166) \cdot 10^{-5}$	-	-	$(\mathbf{2.35 \pm 2.52}) \cdot \mathbf{10^{-6}}$|4
LODE2	-	$(6.59 \pm 0.255) \cdot 10^{-5}$	-	-	$(\mathbf{2.14 \pm 14.4}) \cdot \mathbf{10^{-5}}$|4
LODE3	-	13.16 ± 0.013	-	-	$(\mathbf{2.13 \pm 7.53}) \cdot \mathbf{10^{-3}}$|4
LODE4	-	$(1.65 \pm 0.87) \cdot 10^{-6}$	-	-	$(\mathbf{4.17 \pm 9.16}) \cdot \mathbf{10^{-7}}$|6
LODE5	-	$(9.90 \pm 0.38) \cdot 10^{-5}$	-	-	$(\mathbf{3.01 \pm 8.13}) \cdot \mathbf{10^{-6}}$|4
LODE6	-	$(5.44 \pm 3.64) \cdot 10^{-6}$	-	-	$(\mathbf{1.72 \pm 2.85}) \cdot \mathbf{10^{-7}}$|4
LODE7	-	$(1.25 \pm 0.26) \cdot 10^{-5}$	-	-	$(\mathbf{6.71 \pm 2.33}) \cdot \mathbf{10^{-6}}$|4
LODE8	-	$(1.22 \pm 0.001) \cdot 10^{-2}$	-	-	$(\mathbf{4.87 \pm 5.41}) \cdot \mathbf{10^{-5}}$|4
LODE9	-	$(1.51 \pm 0.001) \cdot 10^{-2}$	-	-	$(\mathbf{1.08 \pm 3.48}) \cdot \mathbf{10^{-2}}$|10
LODE10	-	$(3.15 \pm 0.0519) \cdot 10^{-2}$	-	-	$(\mathbf{5.18 \pm 9.21}) \cdot \mathbf{10^{-4}}$|10
LODE11	-	-	$4.65 \cdot 10^{-3}$	-	$(\mathbf{9.16 \pm 39.4}) \cdot \mathbf{10^{-4}}$|4
NLODE1	-	$(7.42 \pm 0.968) \cdot 10^{-5}$	-	-	$(\mathbf{1.61 \pm 2.12}) \cdot \mathbf{10^{-5}}$|4
NLODE2	-	$(5.90 \pm 0.549) \cdot 10^{-6}$	-	-	$(\mathbf{3.79 \pm 3.31}) \cdot \mathbf{10^{-7}}$|6
NLODE3	-	$(3.64 \pm 0.49) \cdot 10^{-5}$	-	-	$(\mathbf{2.05 \pm 5.04}) \cdot \mathbf{10^{-6}}$|4
NLODE4	-	$(8.32 \pm 0.79) \cdot 10^{-5}$	-	-	$(\mathbf{8.21 \pm 33.0}) \cdot \mathbf{10^{-7}}$|6
NLODE5	-	$(3.19 \pm 0.59) \cdot 10^{-6}$	-	-	$(\mathbf{1.59 \pm 1.74}) \cdot \mathbf{10^{-6}}$|8
NLODE6		$(3.03 \pm 0.0009) \cdot 10^{-1}$			$(\mathbf{2.02 \pm 2.74}) \cdot \mathbf{10^{-1}}$|14
SODE1	-	$(7.465 \pm 1.66) \cdot 10^{-5}$	-	-	$(\mathbf{1.86 \pm 3.95}) \cdot \mathbf{10^{-6}}$|4
SODE2	-	$(3.90 \pm 0.37) \cdot 10^{-5}$	-	-	$(\mathbf{1.40 \pm 1.12}) \cdot \mathbf{10^{-6}}$|4
SODE3	-	$(8.51 \pm 4.02) \cdot 10^{-5}$	-	-	$(\mathbf{8.19 \pm 9.46}) \cdot \mathbf{10^{-6}}$|4
SODE4	-	$(4.72 \pm 0.261) \cdot 10^{-5}$	-	-	$(\mathbf{2.76 \pm 3.23}) \cdot \mathbf{10^{-6}}$|6
SODE5	-	$(3.19 \pm 3.18) \cdot 10^{-6}$	-	-	$(\mathbf{2.28 \pm 1.95}) \cdot \mathbf{10^{-6}}$|10
SODE6	-	-	$1.78 \cdot 10^{-2}$	-	$(\mathbf{1.64 \pm 0.374}) \cdot \mathbf{10^{-2}}$|10
PDE1	$(6.9 \pm 8.3) \cdot 10^{-4}$	$(6.37 \pm 0.73) \cdot 10^{-3}$	-	$7.25 \cdot 10^{-4}$	$(6.20 \pm 3.36) \cdot 10^{-5}$|14
PDE2	-	$(1.16 \pm 0.21) \cdot 10^{-3}$	-	$2.45 \cdot 10^{-4}$	$(\mathbf{9.48 \pm 11}) \cdot \mathbf{10^{-5}}$|8
PDE3	-	$(5.90 \pm 0.79) \cdot 10^{-3}$	-	$9.48 \cdot 10^{-6}$	$(\mathbf{5.02 \pm 2.19}) \cdot \mathbf{10^{-6}}$|10
PDE4	-	$(1.23 \pm 0.03) \cdot 10^{-3}$	-	$6.60 \cdot 10^{-3}$	$(\mathbf{7.02 \pm 5.2}) \cdot \mathbf{10^{-4}}$|8
PDE5	$(1.4 \pm 2.7) \cdot 10^{-2}$	$(9.06 \pm 1.29) \cdot 10^{-4}$	-	$3.72 \cdot 10^{-2}$	$(\mathbf{1.17 \pm 0.73}) \cdot \mathbf{10^{-4}}$|14
PDE6	$(2.0 \pm 2.1) \cdot 10^{-2}$	$(1.79 \pm 0.03) \cdot 10^{-2}$	-	$3.82 \cdot 10^{-1}$	$(\mathbf{1.80 \pm 1.15}) \cdot \mathbf{10^{-4}}$|10

Table 5.18: Comparison of the obtained errors using DESCMA-ES algorithm, respect to exact solution, (RMSE) considering only those works where that information is reported [3, 4, 5]. Standard deviations are not always provided in the referenced works. The best results are marked in bold letter. The final number of centers, n, (see Eq. (4.24)), used by DESCMA-ES for this comparison, is also provided in the last column.

We begin this comparison focusing in those works where RMSE values (or at least some other measure of the solution accuracy) are managed [3, 4, 5], and obviously including DESES and DESCMA-ES algorithms. The results of the comparison are showed in Table 5.18. The DESCMA-ES approach clearly outperforms the results of DESES in terms of lower values of RMSE.

As an example, Fig. 5.14 shows a typical run of DESCMA-ES algorithm against the DESES method for NLODE1 problem. Regarding the set-up for this comparison, the same parameters already commented in Tables 5.12 and 5.13 have been employed for the DESCMA-ES algorithm, i. e., 4 kernels are adjusted (12 unknowns) with a population of $(\mu, \lambda) = (16, 33)$. The DESES algorithm uses 10 harmonics or unknowns and a population of $(\mu, \lambda) = (10, 400)$. Note how the CMA-ES algorithm needs a lower λ value, which turns in a lower number of fitness evaluations. In any case, due to the stochastic nature of the algorithms, when average values are compared after running both algorithms 50 times, DESCMA-ES algorithm obtains a RMSE of $(1.61 \pm 2.12) \cdot 10^{-5}$ needing $(1.88 \pm 0.453) \cdot 10^5$ fitness evaluations, meanwhile in DESES approach, the RMSE obtained was $(7.42 \pm 0.968) \cdot 10^{-5}$ in $(4.87 \pm 1.67) \cdot 10^5$. In DESCMA-ES algorithm all the problems solved by DESES approach have as well successfully solved, meanwhile in the harmonic evolutionary solver LODE3 was not correctly handled. As it was already commented, in DESES the number of unknowns increases exponentially with the space dimension. This drawback does not exist DESCMA-ES. Thus, for example, for a PDE with two dependent variables, the election of 10 centers in DESCMA-ES algorithm turns into 40 unknowns, meanwhile 10 harmonics in DESES implies the tuning of 100 unknowns, more than the double. Thanks to the good performance of CMA-ES, all the unknowns can be adjusted simultaneously. However, in DESES algorithm, successive ES steps should be done to adjust, in each, a harmonic coefficient. Therefore our DESCMA-ES algorithm greatly simplifies the procedure used to tune the variables (unknowns).

Another approach using Fourier series can be seen in Babaei's work [4], where PSO techniques are employed to tune the fitness coefficients. There, none PDE was presented.

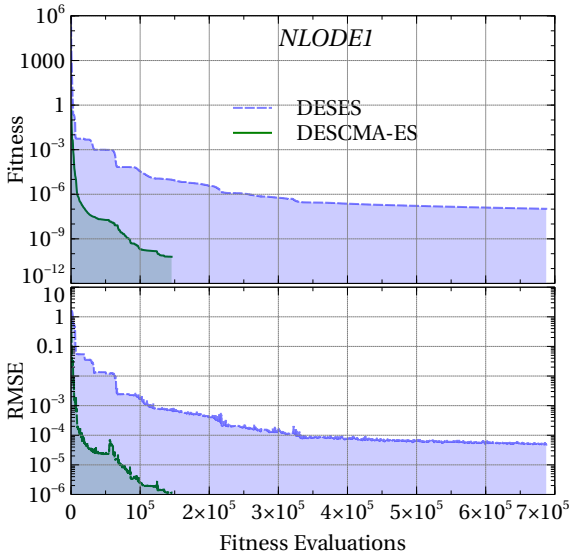

Figure 5.14: Comparison of a typical run of DESCMA-ES with DESES algorithm for problem NLODE1.

In that work, some tuned Fourier coefficients are given, so it is possible to calculate and compare the RMSE values obtained. Thus, for the *brachistrone* problem (SODE6) the best RMSE obtained was $1.78 \cdot 10^{-2}$. In DESCMA-ES algorithm $4.44 \cdot 10^{-3}$ is achieved as best value using 4 centers, which gives the same number of unknowns, 12, than used in [4]. In a similar way, the best RMSE value reported in [4] for LODE11 is $4.65 \cdot 10^{-3}$, meanwhile in DESCMA-ES the best RMSE obtained is $1.74 \cdot 10^{-6}$.

Another work where RMSE values are reported can be consulted in Panagant and Bureerat [5]. There, only PDEs are solved and the solutions are approximated by polynomial functions where their coefficients are tuned using a method based on Differential Evolution. However the number of executions for each problem, the dispersion of the results, and the number of unknowns used are not provided. According to Tables 5.8 and 5.18 is easy to see that DESCMA-ES outperforms Panagant and Bureerat approach [5], using only 8 centers and considering that the number of fitness evaluations is of the same order in both cases (10^5).

Sobester et al. [3] presents a Genetic Programming approach for solving differential equations where the solutions are split into two terms in order to fulfill the boundary conditions

by construction. This particular approach can be adopted only in some specific geometries. In that paper, as a measure of accuracy the MSE (mean square error) is used, although it is straightforward to obtain the RMSE values ($RMSE^2 = MSE$). As we can see in Table 5.18, the accuracy of our approach is better than the results presented in [3].

Solution accuracy were partially reported in Lagaris et al. [12] for PDE1 achieving a maximum error value around 10^{-7}. Mean and dispersion measures are not provided, so we assume that the reported values correspond to the best solution found. A drawback of [12] method is that boundary conditions should be treated in different ways for each problem, and the method only works for second order elliptic differential equations. DESCMA-ES does not have such limitations. Besides, the RMSE can be reduced until 10^{-5} increasing the number of centers (see Table 5.18).

As said above, Table 5.18 summarizes the comparison with the mentioned works in terms of RMSE values. Note that only the available values for each work and equation are given, and very few works report standard deviations.

An important issue to compare stochastic algorithms is the dispersion in the results. A good algorithm should give similar results in all the executions. However, very few works in the literature give dispersion rates. Tsoulos and Lagaris [1] present a differential equation solver using Genetic Programming. Although RMSE values are not provided, as a measure of dispersion the authors give minimum, maximum and average of the number of generations needed to find a correct solution. Knowing the population size used, it is possible to perform a comparison with our proposed approach in terms of the number of fitness evaluations. Thus, the average order of magnitude calculated was 10^5, i.e., the same as that obtained by our method, and all the problems solved there were also successfully solved here.

In contribution by Shirvany et al. [17] a PDE using complex numbers on a triangle shape domain is reported. The RMSE is computed approximating the exact solution by a numerical method solution. The RMSE obtained is around 10^{-4}, which is of the same order than the average obtained for all the PDEs in DESCMA-ES algorithm. However, the number of unknowns that must be tuned in [17] is much higher than in the present work. A total of 132 neurons are needed, against 32 unknowns and RMSE order of magnitude varying from

10^{-2} to 10^{-5} (see Table 5.8) in DESCMA-ES. In a similar way, in Yazdi and Pourreza [36] more than 70 neurons are needed for obtaining a solution in LODE9. In our case, good accuracy (RMSE order of magnitude equal to 10^{-4}) has been obtained with 42 unknowns (see Table 5.14).

From a qualitative point of view, some advantages of DESCMA-ES algorithm can be enumerated. Firstly, in our algorithm is straightforward to compute the derivatives because all the solutions are only expressed as sum of Gaussian kernels. This feature is also true in DESES method. In works based on GP [27, 3, 1], an automatic differentiation engine must be used. In neural networks approaches [12, 17, 2, 14, 18], the activation function must be differentiated, which could be hard depending on the chosen function. Furthermore, if more than one hidden layer is used, the symbolic derivatives implementation could turn very complex.

Secondly, like in DESES, low dispersion in the results has been observed, so this finding provides evidence about the robustness of our method. Works based on GP reported a higher dispersion.

And finally, the proposed approach can be applied to different types of differential equations. Some authors [12, 3, 36] use some particular methods for dealing with the boundary conditions, facilitating the optimization process eliminating the constraints. Nevertheless, these methods are problem dependent and can not be applied to all problems. DESES and DESCMA-ES alorithms do not assumed any particular structures in the boundary conditions making the methods suitable for all problems handled in this work.

Chapter 6

Future Research

The motivation of this chapter is to sketch some possible future works and research. From the three approaches presented to solve differential equations with evolutionary algorithms, DESCMA-ES method has obtained the best results in all of the benchmarking problems in terms of accuracy and convergence. Therefore, in this chapter several ideas to improve DESCMA-ES are discussed.

First, it is sketched the effects of changing the kernel functions in the capabilities of the algorithm to approximate complex solutions with strong gradients. Then, other challenging equations are solved with DESCMA-ES. These problems are stiff equations, which have some difficulties even for numerical methods.

6.1 Analysis of other kernels in DESCMA-ES

To see the influence of the kernels in DESCMA-ES algorithm, LODE4 and NLODE6 equations have been solved using several kernels. The best results have been obtained with arctan kernel, Eq. (4.34), improving even the good results obtained with Gaussian kernels.

Thus LODE4 has been correctly solved using 4 arctan kernels, achieving a fitness value of $6.9 \cdot 10^{-12}$ and a RMSE of $1.5 \cdot 10^{-8}$, which are better than those using Gaussian kernels (fitness of $2.7 \cdot 10^{-9}$ and RMSE of $2.6 \cdot 10^{-6}$). NLODE6 equation has been solved using only 1 and 4 new kernels. The control parameter inner penalty κ has little influence in the results,

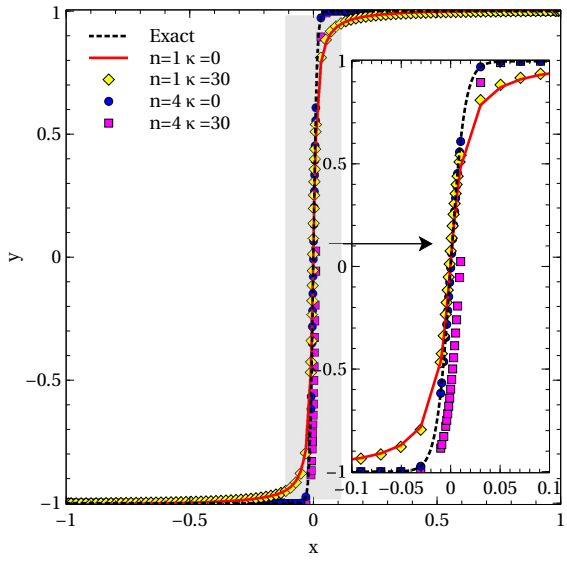

Figure 6.1: Comparison of the exact solution with several evolved ones for NLODE6 problem using DESCMA-ES and arctan kernel.

Kernels	κ	Fitness	RMSE	Fitness Evaluations
1	0	$3.52e-3$	$4.61e-2$	6559
1	30	$3.40e-4$	$4.48e-2$	7239
4	0	$3.30e-10$	$2.40e-3$	97115
4	30	$4.98e-9$	$2.13e-1$	289310

Table 6.1: Inner penalty κ sensitivity analysis in NLODE6 using DESCMA-ES algorithm with arctan kernel.

even more, it is better not to use it in order to weight more the errors in the origin, where is located the strongest gradients (Table 6.1). The solution is compared with the exact one in Fig. 6.1. We can observe that the agreement is pretty good with just one kernel, and with 4 kernels the matching is almost perfect.

6.2 More complex problems: Stiff equations

This final section is devoted to test the performance of the best algorithm so far, the DESCMA-ES, on more challenging equations. To do that, three stiff equations are studied. Our intention in this section is not to do an exhaustive study, but to motivate the analysis of this new equations and to show the potential of DESCMA-ES algorithm.

In mathematics, a stiff equation is a differential equation for which certain numerical methods for solving the equation are numerically unstable, unless the step size is taken to be extremely small. It has been proven difficult to formulate a precise definition of stiffness, but the main idea is that the equation includes some terms that can lead to rapid variation in the solution.

When integrating a differential equation numerically, one would expect the requisite step size to be relatively small in a region where the solution curve displays much variation and to be relatively large where the solution curve straightens out to approach a line with slope nearly zero. For some problems this is not the case. Sometimes the step size is forced down to an unacceptably small level in a region where the solution curve is very smooth. The phenomenon being exhibited here is known as stiffness. In some cases we may have two different problems with the same solution, yet problem one is not stiff and problem two is stiff. Clearly the phenomenon cannot be a property of the exact solution, since this is the same for both problems, and must be a property of the differential system itself. It is thus appropriate to speak of stiff systems.

Stiff Example 1: Simple Stiff Equation

In this first equation, a motivation example has been solved with DESCMA-ES algorithm. The equation is

$$y'(t) = -15y(t)$$

in the range $t \in [0, 1]$ with the boundary condition $y(0) = 1$. The exact solution is $y(t) = e^{-15t}$. Fig. 6.2 illustrates the numerical issues for various numerical integrators applied on

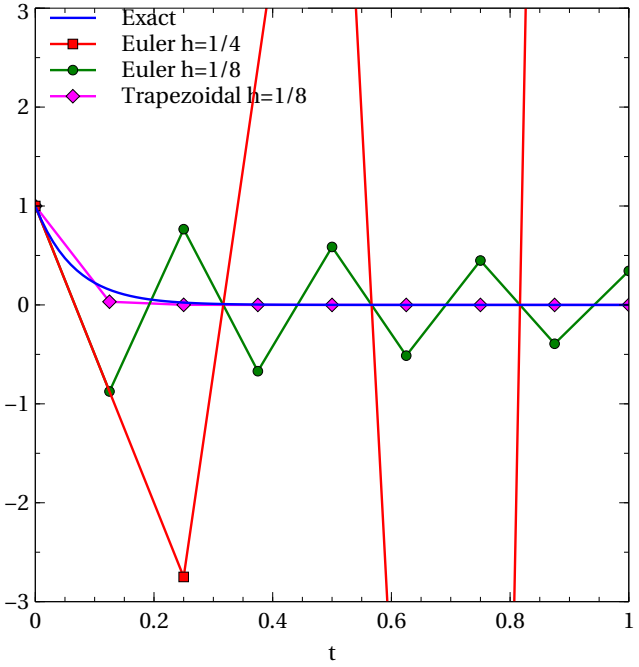

Figure 6.2: Explicit numerical methods exhibiting instability when integrating a stiff ordinary differential equation.

the equation. Euler's method with a step size of $h = 1/4$ oscillates wildly and quickly exits the range of the graph. Euler's method for this case can be expressed as

$$y\left(t_{n+1}\right) = \left(1 - 15h\right) \cdot y\left(t_n\right),$$

being h the step size in time, that is, $t_{n+1} = t_n + h$. Euler's method with half the step size, $h = 1/2$, produces a solution within the graph boundaries, but oscillates about zero. The trapezoidal method (i.e., the two-stage Adams-Moulton method) gives a much better result. This method consist of

$$y\left(t_{n+1}\right) = y\left(t_n\right) - 15h\left[y\left(t_{n+1}\right) + y\left(t_n\right)\right],$$

so finally

$$y\left(t_{n+1}\right) = \frac{2 - 15h}{2 + 15h} y\left(t_n\right).$$

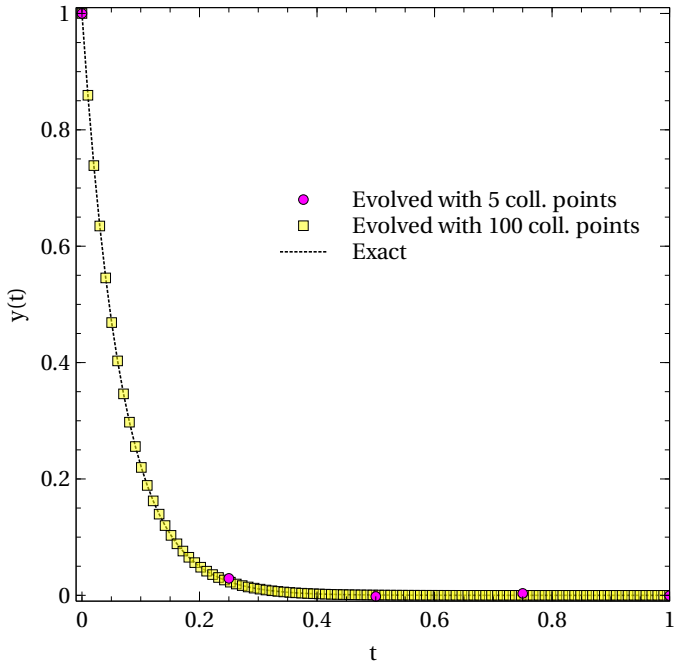

Figure 6.3: Comparison of exact and evolutionary solutions for stiff 1 equation.

Fig. 6.3 compares the exact solution with two evolved solutions using DESCMA-ES. The same control parameters given in Tables 5.12 and 5.13 are employed using 4 centers. In the first solution, only 4 collocation points are used (which gives a step size of $h = 1/4$) and 100 collocation points in the second solution. As we can observe, good agreement is obtained no matter how many collocation points are used, on the contrary than numerical methods.

Stiff Example 2: Van der Pol oscillator

In dynamics, the Van der Pol oscillator is a non-conservative oscillator with non-linear damping. It evolves in time according to the second order differential equation:

$$y'' - \mu \left(1 - y^2\right) y' + y = 0$$

Lets solve this equation with $\mu = 5$ and boundary conditions $y(0) = 2$ and $y'(0) = 0$ on a range $0 \leq t \leq 15$. Fig. 6.4 shows the exact solutions. As we see, the solution is periodic with a period around 11.

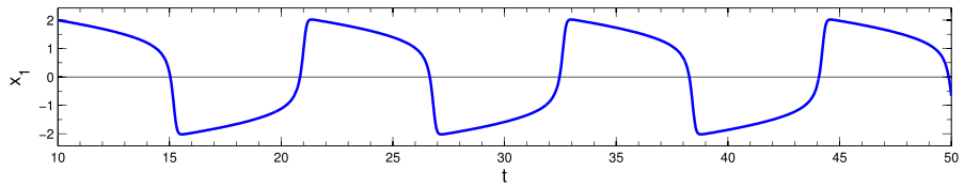

Figure 6.4: Exact solution of Pol oscillator with $\mu = 5$.

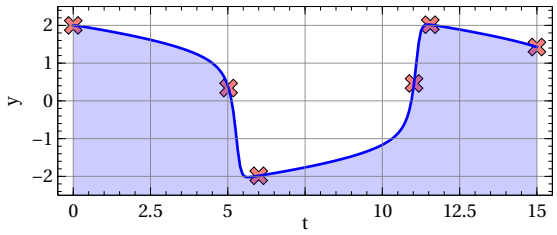

Figure 6.5: Van der Pol oscillator solution obtained using DESCMA-ES algorithm with arctan kernel and splitting the time domain into 6 sub-domains, marked in the plot with crosses.

Taking advantage of the type of Van der Pol oscillator equation (initial value problem), the time domain has been split into 6 sub-domains, as we can see in Table 6.2. The boundary conditions of each domain is taken from the last point of the previous domain imposing continuity in the function and in its first derivative. In Fig. 6.5 the evolved solution is shown.

Stiff Example 3: Robertson equation

This problem is a stiff system of 3 non-linear ordinary differential equations. It was proposed by H. H. Robertson in 1966:

$$\left.\begin{aligned} y_1' &= -0.04y_1 + 10^4 y_2 y_3 \\ y_2' &= 0.04y_1 - 10^4 y_2 y_3 - 3 \cdot 10^7 y_2^2 \\ y_3' &= 3 \cdot 10^7 y_2^2 \end{aligned}\right\}$$

Sub-domain	Range $t \in [a, b]$	$y(a)$	$y'(a)$	Kernels	Fitness
1	$[0, 4.5]$	2	0	8	$3.97602e-09$
2	$[4.5, 5.025]$	1.00079	-0.62987	8	$2.77432e-08$
3	$[5.025, 6]$	0.339938	-2.72308	8	$1.56187e-04$
4	$[6, 11.025]$	-1.98307	0.133722	8	$1.26075e-05$
5	$[11.025, 11.55]$	0.461843	6.49762	10	$1.27254e-03$
6	$[11.55, 15]$	2.01693	-0.0985352	8	$3.53942e-03$

Table 6.2: Time splitting for solving Van der Pol oscillator using DESCMA-ES algorithm with arctan kernel.

with the boundary conditions $y_1(0) = 1$ and $y_2(0) = y_3(0) = 0$. The Robertson problem describes the kinetics of an autocatalytic reaction. It is very popular in numerical studies and it is often used as a test problem in the stiff integrator comparisons. The large difference among the reaction rate constants is the reason for stiffness. As is typical for problems arising in chemical kinetics this special system has a small very quick initial transient. This phase is followed by a very smooth variation of the components where a large step size would be appropriate for a numerical method. Originally the problem was posed on the interval $0 \leq t \leq 40$, but it is convenient to integrate it on much longer intervals. As a matter of fact, Hindmarsh discovered that many codes fail if t becomes very large [105]. In this case if y_2 accidentally becomes negative, it then tends to $-\infty$, causing overflow.

Robertson equation has been solved using DESCMA-ES solver with arctan. To obtain a good result, the 3 equations are transformed into a system of two equations. For that, the third equation is used to substitute the second dependant variable ($y_2 = \sqrt{y_3'/3 \cdot 10^7}$). Thus, the original problem is transformed into:

$$\left.\begin{array}{l} y_1' = -0.04y_1 + 10^4 y_3 \sqrt{\frac{y_3'}{3 \cdot 10^7}} \\ \frac{y_3''}{\sqrt{12 \cdot 10^7 y_3'}} = 0.04y_1 - y_3 \sqrt{\frac{y_3'}{0.3}} - y_3' \end{array}\right\}$$

with the boundary conditions $y_1(0) = 1$ and $y_3(0) = y_3'(0) = 0$. The system has been solved in the time range $t \in [0, 10^{11}]$. Due to the behavior of the system, the collocation points are distributed in a logarithm scale. A total of 10 kernels are used and the original time range is solved in only one run without splitting it. Fig. 6.6 compares the solutions obtained with the exact one. We can appreciate a good matching obtaining a fitness value of $3.31e-4$. Other runs with different number of kernels are also plot. Note that the shape of y_2 is better using less number of kernels. However, the steady state of y_3 at the end of the time range is not 1 because of the errors at the first time instants. It has been tested as well to split the original range into two intervals, marked as 6+6 kernels in the plot.

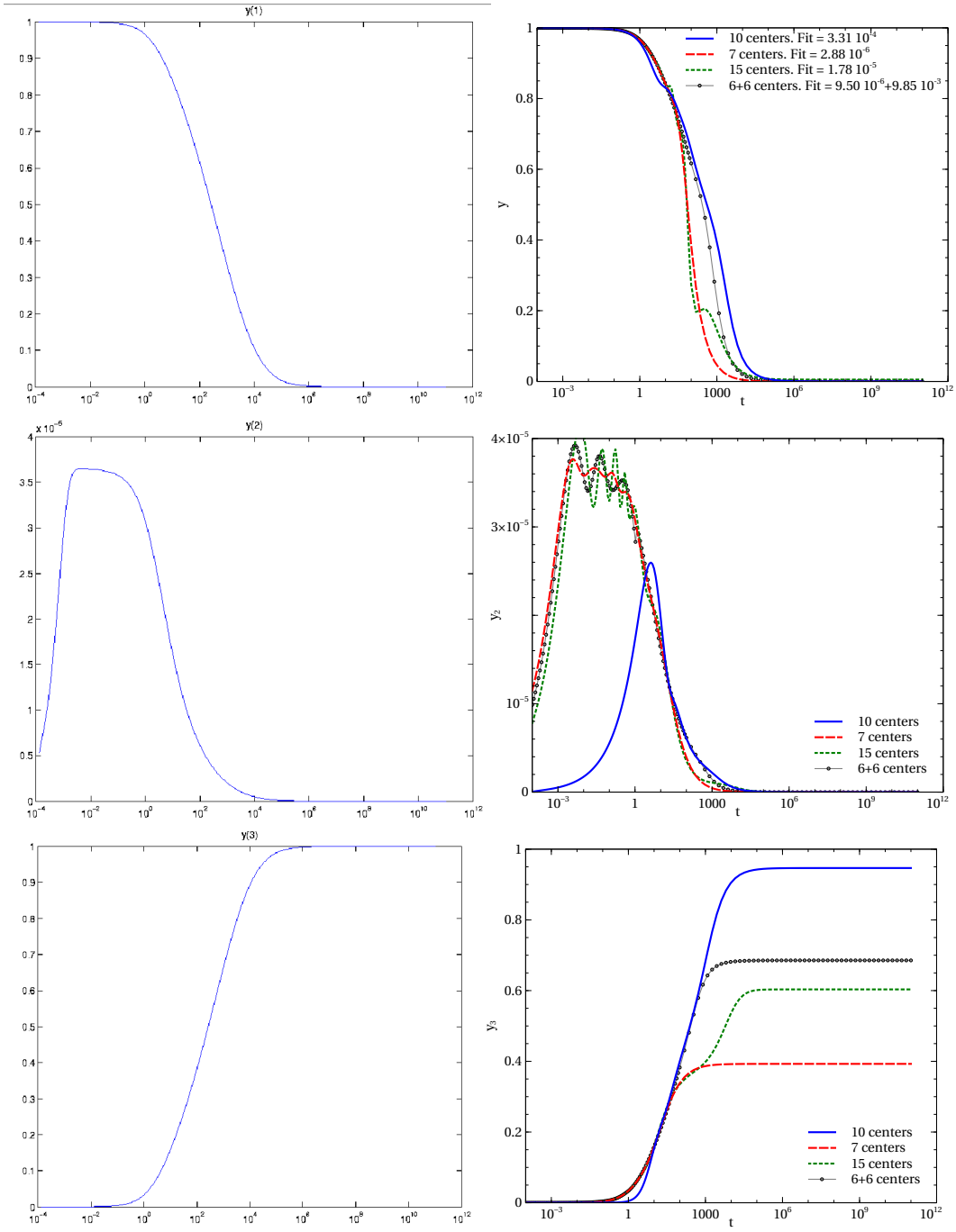

Figure 6.6: Comparison of exact (left column) and evolutionary solutions (right column) for Robertson equation using arctan kernels.

Chapter 7

Conclusions

In this final chapter, the main conclusions and contributions are summarize. As the title of the thesis suggests, the main target of this work is to demonstrate that is reliable to solve differential equations (DEs) with evolutionary algorithms. First, a survey of the related works has been performed. Non-classical methods to solve DEs are relative recent, with the first papers appearing at the last decade of 20th century. So it is a young field with around 20 years of history. Interest in the research community is observed, appearing new works every year. This interest can be partially explained thanks to the great advances in Evolutionary Computing techniques and increment of the computation power of the new hardware.

Three novel methods to solve DEs based on evolutionary algorithms are proposed. The first one, called DESGE (Differential Equation Solver based on Grammatical Evolution), is based on Grammatical Evolution and expresses candidate solutions as generic mathematical expressions. The second one uses a Fourier series to express candidate solutions and an ad-hoc Evolutionary Strategy to seek the optimum harmonics. We call this algorithm DESES (Differential Equation Solver based on Evolution Strategies). Finally, a third approach based on Gaussian kernels and Covariance Matrix Adaptation Evolutionary Strategies has been implemented. This method is called DESCMA-ES (Differential Equation Solver based on Covariance Matrix Adaptation Evolutionary Strategies).

A total of 32 problems have been defined to test the algorithms. The first approach,

DESGE algorithm has been tested in some of those problems. No good results have been obtained with the base algorithm because of the inefficient method to seek numeric constants. Better results were obtained with the so called enhanced DESGE, where 4 modifications to the baseline algorithm are proposed. However, in some problems, it was not possible to decrease the fitness function from a certain level. Besides, some theoretical limitation of these approach has been reported, such as difficulties to approximate solutions without an exact representation or when the functions and terminals in the grammar do not contain the necessary elements to represent the exact solution.

A more systematic study has been performed with the second approach, DESES. Good results have been performed in the majority of the equation, outperforming other heuristic approaches in the literature. However, some convergence problems have been reported in one of the benchmarking equation, and bad scalability is observed in partial differential equations.

Finally, using the third approach, DESCMA-ES algorithm, has been demonstrated good behavior in all the 32 benchmarking problems. The comparison with other approaches have confirmed the good features of the algorithm and good scalability is observed in all the solved problems.

After the finalization of the present thesis, we can answer the following research questions stated in the introduction chapter.

Is it more efficient to express candidate solutions using function basis, parametric kernels or generic mathematical expressions?

To obtain the answer to this question, three different evolutionary algorithms have been implemented. Regarding how the candidate solutions are expressed, the algorithms are very different. Thus, DESGE uses generic mathematical expressions, DESES algorithm employs Fourier series and DESCMA-ES Gaussian kernels.

DESGE approach could turn inefficient in some problems. Although it is possible to obtain the exact solution (if it exists and if the appropriate functions and terminals are included in the grammar) dispersion in the results is observed, and some problems are not correctly solved.

Using a function based approach (concretely speaking, Fourier series have been employed in this thesis) improves significantly the results. Almost all the problems are correctly approximated (with this approach the exact solutions are not obtained any more). Nevertheless, the efficiency of the method decreases when the dimensionality of the solution domain increases. The scalability of the algorithm is not good, increasing the number of unknowns with the power of the number of independent variables.

The best results have been obtained using kernels to express the candidate solutions. The kernels used are built using Gaussian functions. As in the previous approach, exact solutions never will be obtained. Better scalability to partial differential equations is observed. Opposite to the previous approach, a direct evolutionary algorithm has been used to find the best solution, not needing different sub-steps to gradually seek the parameters. Several kernels have been tested. To solve more complex differential equations, such as stiff equations, even better behavior have been observed using arctangent kernels instead of Gaussian functions.

Therefore, we conclude that the most efficient method to express candidate solutions to solve differential equations is using kernel functions.

Does the heuristic algorithms need to deal with non-separable problems in the context of solving DEs?

A problem is called *separable* [106], when the decision variables involved are independent of each other. Therefore, the problem can be easily solved by decomposing it into a number of sub-problems, each of which involves only one decision variable while treating all others as constants. Mathematically speaking, a function $f(x_1, \cdots, x_N)$ is separable if

$$\begin{gathered} \arg\min_{(x_1,\cdots,x_N)} f(x_1, \cdots, x_N) = \\ \\ (\arg\min_{x_1} f(x_1, \cdots), \cdots, \arg\min_{x_N} f(\cdots, x_N)) \end{gathered} , \tag{7.1}$$

where the dots $(\cdots)$ in the right part of the expression means that the rest of variables can be set to any constant value for doing the partial minimization. Some separable functions

fulfill a stronger property called *additively* separable condition when

$$f(x_1, \cdots, x_N) = \sum_{i=1}^{N} f_i(x_i). \tag{7.2}$$

Therefore, all additively separable functions are as well separable, but not all separable functions are additively separable. An easy way of obtaining *non-separable* functions consist in applying a reflection or rotation to the search space before using a separable function.

When the original DE problem is transformed into an optimization one, the question is if the new problem is non-separable. According with the results reported, we justify the better performance of DESCMA-ES respect DESES algorithm in the fact that the new optimization problems obtained transforming the original DE problems are non-separable. The ES used in DESES approach could deal with non-separable problems if correlated mutation operator had been used. However, according to Rudolph [107], the convergence of ES on non-separable problems is not satisfactory. The main reason might be found in the fact that the step size adaptation process affects the angle adaptation process in a disruptive way. For that reason, in order to obtain acceptable convergences, a sophisticate policy of running several sub-steps introducing harmonics one by one has been adopted. If all the harmonics are tuned simultaneusly, the evolutionary algorithm is normally trapped in a local optimum.

On the other hand, in DESCMA-ES algorithm, the evolutionary method adopted can handle non-separable problems in a natural way. Therefore, good convergence is reported although all the variables are handle simultaneously. This fact make DESCMA-ES algorithm much more efficient, because any guided search by means of different sub-steps is needed.

Although numerical methods are by far the most efficient approach to solve DEs, could heuristics outperform them in some aspects? If any, which ones?

Maybe the greater advantage of the proposed approaches (DESES and DESCMA-ES) is that they provide a generic framework, i. e., they does not depend on the type of differential equation. On the other hand, numerical methods are specific for each equation type. For instance, classical Runge-Kutta methods are used for initial value problems due to their

higher accuracy features compared with more efficient algorithms as Euler's method. Classical Runge-Kutta methods are explicit, and are unsuitable for stiff systems because of their small region of stability. On the contrary, implicit Runge-Kutta methods have a large region of absolute stability [7]. Boundary value problems requires different algorithms, such as shooting method for one-dimensional problems or the finite element method for more general domains. Even for the same equation, in some problems depending on the boundary conditions the numerical scheme must be changed due to stability reasons [104]. Furthermore, the implementation of a new numerical method could turn difficult because it is necessary to take into account several issues as the discretization order, the algorithm stability, the convergence rate, how to fulfill the boundary conditions, etc.

In the methods described in this thesis, the original problem is transformed into an optimization one, so the problem of choosing the most appropriate numerical method disappears. In some cases the evolutionary approach can achieve a more accurate solution using less number of nodes. The solutions obtained are coded in a more compact way requiring significantly less amount of memory. Other advantage of evolutionary algorithm is that in some cases it has lower memory requirements. Numerical methods must store as many values as collocation points. And what is more important, the solution is symbolically stored, so new solution values different from the collocation points can be obtained without performing any interpolation.

However, the main drawbacks of evolutionary methods can be considered from two different point of views. In the one hand, the stochastic nature of the evolutionary algorithms cannot guarantee the convergence to the solution in the 100% of the executions. On the other hand, it is possible to obtain a low convergence speed because a population of candidate solutions has to be handled.

Because derivatives must be obtained from each solution proposed by the evolutionary algorithm, it is possible to implement efficient solvers without using complex symbolic engine libraries?

The answer to this question is affirmative if a function basis or a family of kernels are used. That is, if the symbolic derivatives of the involved functions can be known a priori, the symbolic engine only is needed for compute algebraic expressions. In this thesis, a simple symbolic engine based in binary trees has been implemented, making the computation much more efficient than if a commercial or open-source engine is employed.

When kernel approach is used, depending of the complexity of computing their derivatives, in some occasions a mathematical engine can be used. But this process can be done offline and store the derivatives. Thus, when the evolutionary algorithm is run, the derivatives of the kernel are known, so the mathematical engine is not needed.

Although evolutionary algorithms are stochastic methods, it is possible to guarantee a good convergence to the solution?

Because of the stochastic characteristic of evolutionary algorithms, the answer to this question is done from an experimental point of view and taking into account only the set of benchmarking problems studied in this thesis. Good behavior in all the 32 benchmarking problems are reported achieving equal or better accuracy respect to other methods. Low dispersion has been observed in the results, an the same control parameters have been maintained in all the experiments.

Although a not exhaustive study has been done on three more complex problems (stiff equations), the DESCMA-ES algorithm seems to behave properly. However, the solution quality depends on the type of kernel used and how a unique fitness function is built using the errors of the DE itself and the boundary conditions. As future works, the behavior of the algorithm in these more complex equations should be studied in a more systematic way.

Appendix A

Release Control Version

```
GODEHYDA: Genetic prOgramming Derivative Equation HYbrid DynAmic solver

22/03/2010 v0.0 First release. Basic data structure.

16/05/2010 v0.1 New features: local search for ephemeral constants. Fitness
                evaluations counter. Addition crossover. Gaussians. Gaussian
                local search. Inertia in fitness. Reinitialize population.

26/06/2010 v0.2 Improve ephemeral local search: random delta for initialization
                and relaxation of 0.8 in Newton-Raphson. Probability for
                local serach. Convergence History with
                fitness evalautions counter. Change elitism and best/worst
                chromosome policy.

31/07/2010 v1.0 Radical change of ephemeral constant coding acording to paper
                "Introducing Based Extensions for Grammatical Evolution":

<real> = <int>.<int> | <int>.0 | (-<int>.<int>) | (-<int>.0) |
   <GECodonValue_0_9>.<int>e+<GECodonValue_0_9><GECodonValue_0_9><GECodonValue_0_9> |
   <GECodonValue_0_9>.<int>e-<GECodonValue_0_9><GECodonValue_0_9><GECodonValue_0_9> |
 (-<GECodonValue_0_9>.<int>e+<GECodonValue_0_9><GECodonValue_0_9><GECodonValue_0_9>) |
 (-<GECodonValue_0_9>.<int>e-<GECodonValue_0_9><GECodonValue_0_9><GECodonValue_0_9>)
<int>  = <int><GECodonValue_0_9>|<GECodonValue_0_9>

                Still there are several "ad hoc" values in the code:
                  - Precission for Real numbers: 8
                  - Maximum delta for real local search: 1e-3
                  - Relaxation factor for Newton-Rapson: 0.8
                  - Tolerance for stop criteria in Newton-Rapson: 1e-7
                  - Tolerance for fitness inertia counter: 1e-20
                  - Initial real values for gaussian local search: 0.01

29/09/2010 v0.3 Using v0.2, improving of grammar for ephemeral constants:
```

```
<GECV> = <GECodonValue_0_9>
<eph> = <GECV>.<GECV>e+<GECV><GECV><GECV> |
        <GECV>.<GECV>e-<GECV><GECV><GECV> |
      (-<GECV>.<GECV>e+<GECV><GECV><GECV>) |
      (-<GECV>.<GECV>e-<GECV><GECV><GECV>)

                - Maximum delta for real local search: 1e-1

29/10/2010 v0.4 Using v0.3 several crossovers operators can be chosen. A new one
                is implemented: LHSE (LHS enhanced).
                Symbolic Newton-Raphson for Local Search.

03/12/2010 v2.0 Radical change. Use Fourier series and Evolutionary Estrategies.

03/12/2010 v2.0.0 Dynamic Fitness functions, trying avoid local optima.

24/12/2010 v2.1 Use Even extension of Fourier Series without c coefficients.

25/12/2010 v2.2 Use Taylor series.

08/01/2011 v2.3 As version 2.1 using Niching.

18/01/2011 v2.4 Some minor improvements. Cache for the collocation points.

21/01/2011 v3.0 Not linking with GiNaC and CLN for increasing performance.
                Parser for expression using same idea than Gene Expression
                Programming.

12/03/2011 v3.1 Remove sine extension and niching features. Only multiple
                mutation steps.

01/04/2011 v3.2 Remove descendant Order list. Adding last step for fine tuning.

27/08/2011 v4.0 Change Evolutionary Estrategy by Dual Genetic Algorithm.

09/09/2011 v4.0.1 Include a substep in the dual GA.

15/10/2011 v3.3 From 3.2, do Separation of variables reducing the number of
                unkows in PDEs.

17/10/2011 v3.2.1 From 3.2, do separation of variables only when a new harmonic
                  is taking into account.
                  Some test for improving ODE3 performances: bc penalty dynamic
                  with the generation number, (mu+lambda) posibility.

17/08/2013 v5.0 Use v3.2 as base line, change base functions from sine to
                exponential, using support vector machine ideas.

31/08/2013 v5.1 As 5.0, but in three phases: the first one the centers are
                not evolved, only the weights and the betas. Then the centers,
                and finally all the variables.
```

```
01/09/2013 v5.2 As v5.0, but the centers are sorted for a better cross
                operation. Add new parameter: FeasibleFactor.

03/09/2013 v5.3 Usign v5.2 as baseline, add correlated mutation.
                operation.

09/09/2013 v5.4 Usign v5.2 as baseline, subtitute the fine tuning phase by
                DownHill Simplex method.

12/09/2013 v5.5 Make only one phase dealing with all the centers. Do a local
                search in the best individual with a DownHill simplex method.

06/02/2014 v5.4 Use a random sign and increment in Jacobian icrement for
                Downhill Simplex. Maximum increment = 1e-3.
                Reduce max. iterations of Simplex to 200000 and tarConvCrit
                and parConvCrit to parConvCrit to 1e-20.
                Things to do: Impose a maximum generations of 3000 in Simplex.
                              Compleate statistic history in Simplex.
                              Store phase in statistics.

14/02/2014 v5.6 Taken 5.4, change Gaussian sigma by its inverse, called gamma.
                It was already done in previous code!
                Substitute ES by DownHill Simplex.

17/02/2014 v5.7 DownHill Simplex whith niching.

24/02/2014 v6.0 Use CMA-ES in the first phase, and DownHill Simplex in the
                second one.

04/03/2014 v6.1 Absolute value in gammas at evalutaor-> poor performance!

04/03/2014 v6.2 Not allow negative gammas, resampling population if detected.
                -> poor performance!

04/03/2014 v6.3 First phase as v6.0 (no restriction in gamma values). Change
                second phase removing restarts: Apply DownHill Simplex method
                on the best 10 individuals of the last generation.

13/03/2014 v6.4 As 6.0, but it is possible to define weights to compute fitness.

16/03/2014 v6.4 Implement two new functions: allen() and si().

24/03/2014 v6.5 As 6.0, but including allen() and si() functions. Satistics
                with values before and after the local search.
                New brachistochrone function: brachis
                For this problem, when an exact solution is a constant, the
                errors are computed taking the absolute value of the aproximated
                solution.

08/04/2014 v6.6 Including new parameter(OutlierPenalty) to try to fix NLODE6.
                Warning: it should be done more efficiently!
```

```
12/04/2014 v6.6 Change OutlierPenalty by innerPenalty considering the distances
                to the boundary condition points.

19/06/2014 v6.6.1 Dump in the .sol file all the derivatives in order to allow
                  split the domain into several subdomains and facilitate the
                  convergence of stiff equations.

22/06/2014 v6.7 Change Gaussian kernels by atan kernel.

13/07/2014 v6.8 Change atan kernels by polinomials.

09/09/2014 v6.6.2 As 6.6.1, but 50 repetitions in multisolver.
```

Appendix B

A C++ Program to Test GiNaC Library

In order to test all the needed features to develop a differential equation solver based on Grammatical Evolution (DESGE), the following C++ program has been created:

```
#include <iostream>
#include <sstream>
#include <ginac/ginac.h>
using namespace std;
using namespace GiNaC;

int main()
{
  symbol x("x"), y("y");
  lst symbols(x,y);
  string expression;

  cout << "Type a symbolic expression in x and y:\n";
  cin >> expression;
  cout << "Results:\n\n";
  try
  {
    //Create symbolic expression
    ex e(expression, symbols);

    //Transform into string
    ostringstream s;
    s << e;
    cout << "e in std::string format: " << s.str() << "\n";
```

```
    //Some operations: derivatives, substitutions and evaluations
    cout << "e(x,y) = " << e << "\n"
      << "e(x,y).diff(x) = " << e.diff(x) << "\n"
      << "e(x,y).diff(y) = " << e.diff(y) << "\n";
    e = e.subs(x==2*y);
    e = e.subs(y==sin(y));
    cout << "e(2*y,sin(y)) = " << e << "\n"
      << "e(2*x,sin(y)).diff(y) = " << e.diff(y) << "\n";
    e = e.subs(y==1);
    cout << "e(1,0) = " << e << "\n"
      << "evalf(e(1,0)) = " << evalf(e) << "\n";
  }
  catch(exception &error)
  {
    cout << "Exception: " << error.what() << "\n";
    exit(1);
  }
  catch(...)
  {
    cout << "Unkown exception!\n";
    exit(1);
  }
}
```

A correct output of the program is shown:

```
jose@linux-xpm6:~/GPDE/src/GINAC_TEST> TestGiNaC.lx
Type a symbolic expression in x and y:
x^2+x^x-sin(atan(y))
Results:

e in std::string format: x^2+x^x-y*(1+y^2)^(-1/2)
e(x,y) = x^2+x^x-y*(1+y^2)^(-1/2)
e(x,y).diff(x) = 2*x+x^x*(1+log(x))
e(x,y).diff(y) = y^2*(1+y^2)^(-3/2)-(1+y^2)^(-1/2)
e(2*y,sin(y)) = 4*sin(y)^2-(1+sin(y)^2)^(-1/2)*sin(y)+(2*sin(y))^(2*sin(y))
e(2*y,sin(y)).diff(y) = 2*(log(2*sin(y))*cos(y)+cos(y))*(2*sin(y))^(2*sin(y))-
  (1+sin(y)^2)^(-1/2)*cos(y)+8*sin(y)*cos(y)+(1+sin(y)^2)^(-3/2)*sin(y)^2*cos(y)
e(1,0) = -(1+sin(1)^2)^(-1/2)*sin(1)+(2*sin(1))^(2*sin(1))+4*sin(1)^2
evalf(e(1,0)) = 4.589823896395813796
```

Before integrating GiNaC library in our DE solver, all the numerical evaluations of math functions have been modified. GiNaC always tries to compute whatever demanded operation. With high big numbers, the amount of memory needed could be very high making the computation un-affordable. For instance, an expression such as $\exp(\exp(10)) \simeq 10^{9565}$ can be generated during the GP convergence. GiNaC can computed accurately this expression but is a waste of time because the searched solution should be in a more manageable range.

Therefore GiNaC library has been modified. Only one source file (`numeric.cpp`) has been changed using the `DBL_MAX` constant of the standard library as it is shown in the following code:

```
/** Exponential function.
 *
 *  @return  arbitrary precision numerical exp(x). */
const numeric exp(const numeric &x)
{
  if(abs(x)>=DBL_MAX) throw std::range_error("Numeric overflow (>DBL_MAX)");
return numeric(cln::exp(x.to_cl_N()));
}
```

As we can see, an exception is thrown if the input parameter is higher than `DBL_MAX`. In a standard 64 bits machine, `DBL_MAX`=$1.79769 \cdot 10^{308}$. This technique has been used in the following GiNaC functions: $exp(x)$, $log(x)$, $sin(x)$, $cos(x)$, $tan(x)$, $asin(x)$, $acos(x)$, $atan(x)$, $atan(y, x)$, $sinh(x)$, $cosh(x)$, $tanh(x)$, $asinh(x)$, $acosh(x)$ and $atanh(x)$.

During the develop of the thesis, some numerical problems have been detected in the library. Thus, trigonometric functions give the same value for big numbers: $\sin(C) = 0 \quad \forall C > 10^{39}$. And due to limit of floating point representation, when two numbers are added with very different range, the result could be constant: $x + C = C$ if $x \in [0, 1]$ and $C > 10^{18}$.

Appendix C

Configuration File Examples

An example of a configuration file for DESGE of an ordinary differential equation problem is given in this appendix:

```
GODEHYDA_VERSION 0.0

######################
# PROBLEM PARAMETERS #
######################
INDEPENDENT_VARIABLES x
DEPENDENT_VARIABLES y
CONSTANTS C0 0 C1 1 C2 2 C3 3 C4 4 C5 5 C6 6 C7 7 C8 8 C9 9
#EPHEMERAL_CONSTANS 3  0.000000000000000e+00  1.000000000000000e+01
OPERATORS + - * /
FUNCTIONS sin cos exp log
DERIVATIVES DyDx y x 1
DIFFERENTIAL_EQUATIONS DyDx-(1-y*cos(x))/sin(x)
BEGIN_COLLOCATION_POINTS
#                     x
  0.1
  0.15
  0.2
  0.25
  0.3
  0.35
  0.4
  0.45
  0.5
  0.55
  0.6
  0.65
  0.7
```

```
  0.75
  0.8
  0.85
  0.9
  0.95
  1.0
END_COLLOCATION_POINTS
BEGIN_BOUNDARY_CONDITIONS
#                      x   expression
                     0.1   y-2.1/sin(0.1)
END_BOUNDARY_CONDITIONS

#####################
# SOLVER PARAMETERS #
#####################
#Possible actions: SOLVE CHECK_INPUTS POSTPROCESS MULTI_SOLVE
ACTION SOLVE
CASE_NAME ODE2
MAXIMUN_WRAPPING 1
#SEED 1272065064
BC_PENALTY 100
POPULATION 1000
GENERATIONS 2000
MAX_BEST_FITNESS 1e-10
MINIMUM_CHR_SIZE_INITIALIZATION 20
MAXIMUM_CHR_SIZE_INITIALIZATION 50
MAXIMUM_CHR_SIZE 100
TOURNAMENT_SIZE 3
P_RECOMBINATION 0.9
P_MUTATION_INTRA_CODON 0.2
P_MUTATION_EXTRA_CODON 0.1
ELITISM 2
#USE_SOLUTION
```

There are two sections. In the first one, the problem to solve is defined. It is possible to specify systems of equations defining several dependent variables. The derivatives are defined with couples of dependent and independent variables and their order. The differential equations are specified moving all the terms to one side of the equality mathematical expression.

In the second section of the input file, the parameters for the iterative GP algorithm are specified. It is possible to specify a seed for the random number generator in order to reproduce runs for debugging purposes. If no seed is specified, or 0 value is used, a new seed using the machine clock is generated.

Following another example of a configuration file for DESCMA-ES algorithm where a

partial differential equation is specified is shown:

```
GODEHYDA_VERSION 6.0

######################
# PROBLEM PARAMETERS #
######################
INDEPENDENT_VARIABLES x y
DEPENDENT_VARIABLES Z
DERIVATIVES Zxx Z 2 0
DERIVATIVES Zyy Z 0 2
DIFFERENTIAL_EQUATIONS  - + Zxx Zyy * exp * -1 x + ^ y 3 + x - * 6 y 2
SOLUTIONS * exp * -1 x + x ^ y 3

BEGIN_COLLOCATION_POINTS
#   x  y
   0.  0.
   0.  0.1
   0.  0.2
   0.  0.3
   0.  0.4
   0.  0.5
   0.  0.6
   0.  0.7
   0.  0.8
   0.  0.9
   0.  1.0

   0.1 0.
   0.1 0.1
   0.1 0.2
   0.1 0.3
   0.1 0.4
   0.1 0.5
   0.1 0.6
   0.1 0.7
   0.1 0.8
   0.1 0.9
   0.1 1.0

   0.2 0.
   0.2 0.1
   0.2 0.2
   0.2 0.3
   0.2 0.4
   0.2 0.5
   0.2 0.6
   0.2 0.7
   0.2 0.8
   0.2 0.9
   0.2 1.0
```

```
0.3 0.
0.3 0.1
0.3 0.2
0.3 0.3
0.3 0.4
0.3 0.5
0.3 0.6
0.3 0.7
0.3 0.8
0.3 0.9
0.3 1.0

0.4 0.
0.4 0.1
0.4 0.2
0.4 0.3
0.4 0.4
0.4 0.5
0.4 0.6
0.4 0.7
0.4 0.8
0.4 0.9
0.4 1.0

0.5 0.
0.5 0.1
0.5 0.2
0.5 0.3
0.5 0.4
0.5 0.5
0.5 0.6
0.5 0.7
0.5 0.8
0.5 0.9
0.5 1.0

0.6 0.
0.6 0.1
0.6 0.2
0.6 0.3
0.6 0.4
0.6 0.5
0.6 0.6
0.6 0.7
0.6 0.8
0.6 0.9
0.6 1.0

0.7 0.
0.7 0.1
0.7 0.2
```

```
   0.7 0.3
   0.7 0.4
   0.7 0.5
   0.7 0.6
   0.7 0.7
   0.7 0.8
   0.7 0.9
   0.7 1.0

   0.8 0.
   0.8 0.1
   0.8 0.2
   0.8 0.3
   0.8 0.4
   0.8 0.5
   0.8 0.6
   0.8 0.7
   0.8 0.8
   0.8 0.9
   0.8 1.0

   0.9 0.
   0.9 0.1
   0.9 0.2
   0.9 0.3
   0.9 0.4
   0.9 0.5
   0.9 0.6
   0.9 0.7
   0.9 0.8
   0.9 0.9
   0.9 1.0

   1.  0.
   1.  0.1
   1.  0.2
   1.  0.3
   1.  0.4
   1.  0.5
   1.  0.6
   1.  0.7
   1.  0.8
   1.  0.9
   1.  1.0
END_COLLOCATION_POINTS
BEGIN_BOUNDARY_CONDITIONS
#   x   y      expression
    0.  0.     - Z 0
    0.  0.1    - Z 0.001
    0.  0.2    - Z 0.008
    0.  0.3    - Z 0.027
    0.  0.4    - Z 0.064
```

```
    0.  0.5    - Z 0.125
    0.  0.6    - Z 0.216
    0.  0.7    - Z 0.343
    0.  0.8    - Z 0.512
    0.  0.9    - Z 0.729
    0.  1.0    - Z 1

    1.  0.     - Z 0.367879
    1.  0.1    - Z 0.368247
    1.  0.2    - Z 0.370822
    1.  0.3    - Z 0.377812
    1.  0.4    - Z 0.391424
    1.  0.5    - Z 0.413864
    1.  0.6    - Z 0.447341
    1.  0.7    - Z 0.494062
    1.  0.8    - Z 0.556234
    1.  0.9    - Z 0.636064
    1.  1.0    - Z 0.735759

    0.   0.    - Z 0
    0.1  0.    - Z 0.0904837
    0.2  0.    - Z 0.163746
    0.3  0.    - Z 0.222245
    0.4  0.    - Z 0.268128
    0.5  0.    - Z 0.303265
    0.6  0.    - Z 0.329287
    0.7  0.    - Z 0.34761
    0.8  0.    - Z 0.359463
    0.9  0.    - Z 0.365913
    1.0  0.    - Z 0.367879

    0.   1.    - Z 1
    0.1  1.    - Z 0.995321
    0.2  1.    - Z 0.982477
    0.3  1.    - Z 0.963064
    0.4  1.    - Z 0.938448
    0.5  1.    - Z 0.909796
    0.6  1.    - Z 0.878099
    0.7  1.    - Z 0.844195
    0.8  1.    - Z 0.808792
    0.9  1.    - Z 0.772482
    1.0  1.    - Z 0.735759
END_BOUNDARY_CONDITIONS

######################
# SOLVER PARAMETERS #
######################
#Possible actions: SOLVE CHECK_INPUTS POSTPROCESS MULTI_SOLVE PLOT TEST_NORMAL
ACTION MULTI_SOLVE
CASE_NAME PDE1
#SEED 1296682309
BC_PENALTY 300
```

```
INITIALIZATIONS -0.01 0.01 1 0.1
LAMBDA_MULT 3
EVALUATIONS 1e6
FEASIBLE_FACTOR 2
#RESTART
NCENTERS 8
```

Note how the expressions are given in a such a way that the in-house symbolic interpreter can built a binary tree.

Appendix D

Conclusiones (Conclusions in Spanish)

Las principales conclusiones y contribuciones se exponen a continuación. Como sugiere el título de esta tesis, el objetivo principal de este trabajo es demostrar que es posible resolver ecuaciones diferenciales (EDs) mediante algoritmos evolutivos. En primer lugar, se ha realizado un estudio bibliográfico de trabajos relacionados con la resolución de EDs mediante metodos heurísticos. Estos métodos, al contrario con los métodos tradicionales, son relativamente recientes, ya que los primeros artículos datan de la última década del siglo XX. Por ello, consideramos que se trata de un campo de conocimiento joven con apenas 20 años de historia. La comunidad científica ha mostrado interés en este campo, por lo que podemos encontrar nuevos trabajos cada año. Este interés puede explicarse parcialmente debido al gran avance experimentado en las técnicas de Computación Evolutiva y a un incremento de la capacidad de cálculo del *hardware*.

Se ha desarrollado tres novedos métodos para resolver EDs basados en algoritmos evolutivos. El primero de ellos, llamado DESGE (*Differential Equation Solver based on Grammatical Evolution*), está basado en *Grammatical Evolution* y construye las soluciones candidatas mediante expresiones matemáticas genéricas. El segundo método propuesto usa series de Fourier para representar las soluciones candidatas y una estrategia evolutiva ad hoc para sin-

tonizar los harmónicos. Hemos llamado a este algorimo DESES (*Differential Equation Solver based on Evolution Strategies*). Finalmente, se ha implementado una tercera aproximación basada en *kernels* gaussianos y estrategias evolutivas de matrices de covarianzas adaptativos (CAM-ES). A este método lo denominamos DESCMA-ES (*Differential Equation Solver based on Covariance Matrix Adapatation Evolution Strategies*).

Se han definido un total de 32 problemas para comprobar los algoritmos propuestos. La primera aproximación, el agoritmo DESGE, sólo se ha comprobado en algunos problemas. No se ha obtenido buenos resultados con el algoritmo básico porque el método es ineficiente para ajustar las posibles constantes numéricas que aparecen en la solución simbólica candidata. Se han obtenido mejores resultados con el denominado algoritmo DESGE mejorado, en el cual se han introducido 4 modificaciones respecto al algoritmo inicial. De todas formas, en algunas problemas no ha sido posible alcanzar la precisión deseada. Además, se han encontrado ciertas limitaciones teóricas del método, tales como la dificultad inherente de aproximar soluciones que no cuenten con una representación simbólica exacta o el problema de no disponer en la gramática de las funciones y terminales necesarios para representar las soluciones candidatas.

Se ha realizado un estudio más sistemático con la se segunda aproximación, algoritmo DESES. Se han obtenido buenos resultados en la mayoría de las ecuaciones, mejorando incluso a otras aproximaciones heurísticas de la literatura. Sin embargo, se han detectado ciertos problemas de convergencia en una de las ecuaciones del conjunto de prueba, y también problemas de escabilidad en las ecuaciones en derivadas parciales.

Finalmente, el tercer algoritmo, DESCMA-ES, ha demostrado un muy buen comportamiento en todo el conjunto de las 32 ecuaciones de prueba. Su buen comportamiento ha quedado también patente en una comparación realizada con otras aproximaciones publicadas en la literatura. Así mismo se ha observado una buena escalabilidad en todos los problemas resueltos.

Tras la finalización de esta tesis, estamos en disposición de respondes a las siguiente preguntas de investigación enunciadas en el capítulo introductorio.

¿Es más eficiente expresar las soluciones candidatas mediante una base funcional, o mediante *kernels* paramétricos o usando expresiones matemáticas genéricas?

Para obtener la respuesta a esta pregunta, se han desarrollado tres nuevos algoritmos evolutivos. Respecto a cómo representan las soluciones candidatas, los algoritmos son muy diferentes. Así, DESGE utiliza expresiones matemáticas genéricas, DESES emplea series de Fourier y DESCMA-ES *kernels* gaussianos.

El algoritmo DESGE puede resultar poco eficiente. Aunque es posible obtener la solución exacta (si existe y si la gramática cuenta con las funciones y terminales apropiados), se ha observado bastante dispersión en los resultados. Además, ciertos problemas no se han resuelto con el nivel de precisión deseado.

Los resultados han mejorado significativamente usando una base de funciones, concretamente se han empleado en esta tesis series de Fourier. Aunque todos los problemas se han resuelto satisfactoriamente (con esta aproximación ya no se puede obtener la solución exacta, si es que existe). Sin embargo, la eficiencia del método se reduce drásticamente cuando aumenta la dimensión del problema. La escalabilidad del algoritmo no es buena, ya que el número de incógnitas aumenta exponencialmente con el número de variables independientes de la ecuación diferencial.

Los mejores resultados se han obtenido con el algoritmo DESCMA-ES, el cual utiliza *kernels* para construir las soluciones candidatas. Estos *kernels* se han implementado mediante funciones gaussianas. Al igual que en el algoritmo DESES, no es posible obtener las soluciones exactas. No obstante, la precisión obtenida en los resultados es satisfactoria, así como se observa una mejor escalabilidad en las ecuaciones en derivadas parciales. Al contrario que en el método anterior, ha sido posible el uso de un sólo algoritmo evolutivo directo para encontrar las mejores soluciones, es decir, no ha sido necesario el uso de pasos intermedios de ajuste de los parámetros de forma gradual. Se han analizado varios *kernels*. Para resolver problemas más complejos, como las ecuaciones tipo *stiff*, se ha observado un mejor comportamiento empleando el *kernel* arco-tangente en lugar del gaussiano.

Por todo lo expuesto, podemos concluir que la manera más eficiente de expresar las

soluciones candidatas para resolver ecuaciones diferenciales es mediante el uso de *kernels*.

¿Necesitan los algoritmos heurísticos resolver problemas no separables dentro del contexto de resolución de ecuaciones diferenciales?

Un problema se denomina separable [106] cuandolas variables de decisión onvolucradas son independientes unas de otras. Por tanto, el problema puede resolverse fácilmente descomponiéndolo en varios sub-problemas en los cuales sólo interviene una variable de decisón y el resto se consideran constantes. En notación formal, una función $f(x_1, \cdots, x_N)$ es separable si

$$\arg\min_{(x_1,\cdots,x_N)} f(x_1, \cdots, x_N) = \\ (\arg\min_{x_1} f(x_1, \cdots), \cdots, \arg\min_{x_N} f(\cdots, x_N))$$

donde los puntos $(\cdots)$ de las expresiones a la derecha de la igualdad significan que el resto de las variables se pueden considerar constantes a la hora de reolver los problemas parciales de minimización. Algunas funciones separables pueden cumplir una propiedad más fuerte llamada condición separable aditiva cuando

$$f(x_1, \cdots, x_N) = \sum_{i=1}^{N} f_i(x_i).$$

Por tanto, toda función separable aditiva es también separable, pero no toda función separable es aditivamente separable. Una forma sencilla de obtener una función no separable consiste en aplicar una reflexión o una rotación al espacio de definición a una función separable.

Cuando el problema original de resolver una ecuación diferencial se transforma en un problema de optimización, la pregunta que se nos plantea es si el nuevo problema es no separable. De acuerdo con los resultados obtenidos justificamos las mejores prestaciones del algoritmo DESCMA-ES respecto de DESES en que el problema de optimización obtenido

resulta se no separable. La estrategia evolutiva usada en el método DESES podría tratar problemas no separables si hiciera uso del operador de mutación correlacionada. Sin embargo, de acuerdo con Rudolph [107], la convergencia de las estrategias evolutivas en problemas no separables no es satisfactoria. El motivo principal radica en que la adapatación del paso de mutación afecta al proceso de adaptación de los ángulos de la matriz de covarianzas de una forma muy disruptiva. Por ello, de cara a obtener una convergencia aceptable, se ha tenido que adoptar una estrategia bastante sofisticada ajustando de forma sucesiva y gradual los diferentes armónicos. Si todos los armónicos se ajustan simultáneamente, el algoritmo evolutivo puede quedar atrapado en mínimos locales.

Por otro lado, el algoritmo evolutivo empleado por el método DESCMA-ES puede manejar problemas non separables de una forma natural. Por tanto, se ha observado una buena convergencia ajustando simultáneamente todas los parámetros. Este hecho hace que el algoritmo DESCMA-ES sea mucho más eficiente, ya que no es necesario realizar búsquedas guiadas mediante diferentes subetapas de optimización.

Aunque los métodos numéricos son las aproximaciones más eficientes pare resolver EDs, ¿pueden los métodos heurísticos tener algunas ventajas? ¿Cuáles?

Probablemente la mayor ventaja de los métodos propuestos (DESES y DESCMA-ES) es que son métodos genéricos, es decir, el algoritmo no depende del tipo de ecuación diferencial a resolver. Por el contrario, los métodos numéricos son específicos para cada ecuación o familia de ecuaciones. Poe ejemplo, los métodos clásicos de tipo Runge-Kutta se uilizan para problemas de valor inicial debido a sus buenas características de precisión comparados con otros métodos más eficientes como los de tipo Euler. Los métodos clásicos Runge-Kutta son explícitos, y por tanto no son aptos para sistemas tipo *stiff* debido a su reducida región de estabilidad. Por el contrario, los métodos Runge-Kutta implícitos cuentan con una región de estabilidad absoluta mayor [7]. Los problemas de valors en la frontera requieren de algoritmos numéricos diferentes, tales como el método *shooting* para problemas uni-dimensionales o el método de elementos finitos para domios más generales. Incluso para la misma ecuación, algunos problemas deben usar esquemas numéricos diferentes dependiendo de kas condiciones

de contorno debido a razones de estabilidad [104]. Además, la implementación de un nuevo método numérico puede resultar complejo porque es necesario tener en cuenta diferentes aspectos tales como el orden de la discretización, la estabilidad del algoritmo, la velocidad de convergencia, las condiciones de contorno, etc.

En los tres métodos descritos en esta tesis, el problema original se transforma en un problema de optimización, por lo que la dificultad de elegir el algoritmo numérico más apropiado desaparece. En algunios casos el algoritmo evolutivo es capaz de obtener una solución más precisa usando un menor número de puntos de colocación. Además, la solución se codifica de una forma más compacta necesitando una menor cantidad de memoria. Otra ventaja de los algoritmos evolutivos es que en algunas ocasiones requieren un menor consumo de memoria. Los métodos numéricos deben almacenar tantos valores como puntos de colocación. Y lo que es más importante, la solución en el caso de los algoritmos evolutivos es simbólica, por lo que es posible obtener valores numéricos en puntos diferentes a los usado para la resolución del problema sin necesidad de realizar ninguna interpolación.

Sin embargo, las principales desventajas de los métodos evolutivos puden ser consideradas desde dos puntos de vista. Por un lado, la naturaleza estocástica de los algoritmos evolutivos hace que no se pueda garantizar la vonvergencia a la solución en el 100% de los casos. Por otro lado, el coste computacional puede llegar a ser elevado debido a que se utiliza una población de soluciones.

Dado que de cada solución propuesta por el método evolutivo deben calcularse las derivadas, ¿es posible implementar algoritmos eficientes sin la necesidad de usar complejos motores de cálculo simbólico?

La respuesta a esta cuestión es afirmativa si se utiliza una base funcional o una familia de kernels. Es decir, si las derivadas simbólicas pueden conocerse a priori, el motor simbólico sólo es necesario para el cálculo de expresiones algebraicas. En esta tesis, se ha implementado un motor de cálculo simbólico de forma simple basado en árboles binarios, lo cual hace que el algoritmo sea mucho más eficiente que si se hubiera empleado un motor de uso general.

Cuando se utilice una aproaximación basada en kernels, dependiendo de la complejidad

del cálculo de las derivadas de los propios kernels, en algunas ocasiones puede ser necesario el uso de herramientas de cálculo simbólico de uso general. Pero este proceso puede realizarse *offline* guardando las derivadas simbólicas para su uso posterior. Durante la ejecución del algoritmo evolutivo, no se necesita el motor simbólico, pues las derivadas simbólicas se han precalculado en una etapa anterior.

Aunque los algoritmo evolutivos son métodos estocásticos, ¿es posible garantizar una buena convergencia a la solución exacta?

Dada la naturaleza estocástica de los algoritmos evolutivos, la respuestas a esta pregunta se puede realizar en base a un punto de vista experimental y tomando en consideración únicamente el conjunto de problemas de validación estudiados en esta tesis. Se ha observado un buen comportamiento en los 32 problemas de validación alcanzando una precisión igual o mayor respecto a otros métodos de resolución de ecuaciones diferenciales. Así mismo, se ha observado una baja dispersión en los resultados obtenidos.

Aunque no se ha realizado un estudio exhaustivo en tres equaiones más complejas presentadas en este trabajo (ecuaciones *stiff*), el algoritmo DESCMA-ES parece comportarse mejor que los otros dos métodos. Sin embargo, la calidd de la solución depende del tipo de kernel usado y de cómo la función fitness se construye con los errores de la ecuación misma y con las condiciones de contorno. Como trabajos futuros, el comportamiento del algoritmo es estas ecuaciones más complejas debe ser analizado.

Appendix E

Publications

Solving Differential Equations with Evolutionary Algorithms

- 

J.M. Chaquet and E.J. Carmona, ***"Solving Differential Equations with Fourier Series and Evolution Strategies"***, Applied Soft Computing, Volume 12, Issue 9, Pages 3051-3062, September 2012. Impact factor 2.14 (Q1)

- 

J.M. Chaquet and E.J. Carmona, ***"Using Covariance Matrix Adaptation Evolution Strategies for Solving Different Types of Differential Equations"***, Soft Computing, **Submitted**.

- INSTICC J.M. Chaquet and E.J. Carmona, ***"A Grid-based Genetic Algorithm for Multimodal Real Function Optimization"***, ECTA 2012: 4th International Conference on Evolutionary Computation, Theory and Applications, June 2012. Indexed in SCITEPRESS Digital Library. Rank C (ERA).

Solving Differential Equations with Numerical Methods

- J.M. Chaquet, R. Corral, G. Pastor, J. Pueblas and D.D. Coren, ***"Validation of a Coupled Fluid/Solid Heat Transfer Method"***, 56th ASME Turbo Expo: Power for Land, Sea and Air, GT2011-45951, June 2011. Indexed in the American Society of Mechanical Engineers (ASME) Digital Collection.

- J.M. Chaquet, Z. Wang, R. Corral and G. Pastor, ***"Loosely Coupled Fluid/Solid Heat Transfer Analysis Using a Dynamic HTC Approach"***, Paper 223, European Turbomachinery Conference 2013, April 2013.

- J.M. Chaquet, A. Altuna, R. Corral, F. Gisbert and G. Pastor, ***"Application of a Fast Loosely Coupled Fluid/Solid Heat Transfer Method to the Transient Analysis of Low-Pressure-Turbine Disk Cavities"***, GT2013-95426, ASME Turbo Expo, June 2013. Indexed in the American Society of Mechanical Engineers (ASME) Digital Collection.

- J.M. Chaquet, Z. Wang, R. Corral and G. Pastor, ***"Analysis and Improvement of Loosely Coupled Fluid-Solid Heat Transfer Method"***, GT2013-94332, ASME Turbo Expo, June 2013. Indexed in the American Society of Mechanical Engineers (ASME) Digital Collection.

- J.M. Chaquet, R. Corral, F. Gisbert and G. Pastor, ***"A Loosely Coupled Fluid/Solid Heat Transfer Method For Disc Cavities Including Mixing Planes and a Combination of 2D and 3D Cavities"***, GT2015-42269, ASME Turbo Expo, June 2015. Indexed in the American Society of Mechanical Engineers (ASME) Digital Collection.

Evolutionary Algorithms

-  J.M. Chaquet, E.J. Carmona and R. Corral, ***"Optimizing the Number of Airfoils in Turbine Design using Genetic Algorithms"***. IEA/AIE'10 Proceedings of the 23rd international conference on Industrial engineering and other applications of applied intelligent systems, Part III, Pages 169-178, June 2010. Indexed in ACM Digital Library. Rank C (ERA).

- 

J.M. Chaquet, E.J. Carmona and R. Corral, ***"Using Genetic Algorithms to Improve the Thermodynamic Efficiency of Gas Turbines Designed by Traditional Methods"***, Applied Soft Computing, Volume 12, Issue 11, Pages 3627–3635, November 2012. Impact factor 2.14 (Q1)

Bibliography

[1] Tsoulos, I. G. and Lagaris, I. E., "Solving differential equations with genetic programming," *Genetic Programming and Evolvable Machines*, Vol. 7, 2006, pp. 33–54.

[2] Tsoulos, I. G., Gavrilis, D., and Glavas, E., "Solving differential equations with constructed neural networks," *Neurocomputing*, Vol. 72, 2009, pp. 2385–2391.

[3] Sobester, A., Nair, P. B., and Keane, A. J., "Genetic Programming Approaches for Solving Elliptic Partial Differential Equations," *IEEE Transactions on Evolutionary Computation*, Vol. 12, 2008, pp. 469–478.

[4] Babaei, M., "A general approach to approximate solutions of nonlinear differential equations using particle swarm optimization," *Applied Soft Computing*, Vol. 13, No. 7, 2013, pp. 3354 – 3365.

[5] Panagant, N. and Bureerat, S., "Solving Partial Differential Equations Using a New Differential Evolution Algorithm," *Mathematical Problems in Engineering*, Vol. 2014, No. 747490, 2014, pp. 10.

[6] Spivak, M., *Calculus*, Publish or Perish, 4th ed., 1980.

[7] Suli, E. and Mayers, D. F., *An Introduction to Numerical Analysis*, Cambridge University Press, 2003.

[8] Ozisik, M. N., *Finite Difference Methods in Heat Transfer*, CRC Press, Inc., 1994.

[9] Leveque, R. J., *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, 2002.

[10] Liu, G. R., *Meshfree Methods. Moving Beyond the Finite Element Method*, CRC Press, Inc., 2010.

[11] Puffer, F., Tetzlaff, R., and Wolf, D., "A learning algorithm for cellular neural networks (CNN) solving nonlinear partial differential equations," *Proceedings International Symposium on Signals, Systems, and Electronics, ISSSE '95 URSI*, Vol. 1, 1995, pp. 501–504.

[12] Lagaris, I. E., Likas, A., and Fotiadis, D. I., "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations," *IEEE Transactions on Neural Networks*, Vol. 5, 1998, pp. 987–1000.

[13] He, S., Reif, K., and Unbehauen, R., "Multilayer neural networks for solving a class of partial differential equations," *Neural Networks*, Vol. 13, 2000, pp. 385–396.

[14] Parisi, D. R., Mariani, M. C., and Laborde, M. A., "Solving differential equations with unsupervised neural networks," *Chemical Engineering and Processing*, Vol. 42, 2003, pp. 715–721.

[15] Sun, M., Yan, X., and Sclabassi, R. J., "Solving Partial Differential Equations in Real-Time using Artificial Neural Network Signal Processing as an Alternative to Finite Element Analysis," *IEEE Int. Conf. Neural Networks & Signal Processing*, Vol. 1, 2003, pp. 381–384.

[16] Choi, B. and Lee, J., "Comparison of generalization ability on solving differential equations using backpropagation and reformulated radial basis function networks," *Neurocomputing*, Vol. 73, 2009, pp. 115–118.

[17] Shirvany, Y., Hayati, M., and Moradian, R., "Multilayer perceptron neural networks with novel unsupervised trining method for numerical solution of the partial differential equations," *Applied Soft Computing*, Vol. 9, 2009, pp. 20–29.

[18] Chen, H., Kong, L., and Leng, W.-J., "Numerical solution of PDEs via integrated radial basis function networks with adaptive training algorithm," *Applied Soft Computing*, Vol. 11, 2011, pp. 855–860.

[19] Kumar, M. and Yadav, N., "Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations. A survey," *Computers and Mathematics with Applications*, Vol. 62, 2011, pp. 3796–3811.

[20] Yazdi, H. S., Pakdaman, M., and Modaghegh, H., "Unsupervised kernel least mean square algorithm for solving ordinary differential equations," *Neurocomputing*, Vol. 74, 2011, pp. 2062–2071.

[21] Mosleh, M., "Fuzzy neural network for solving a system of fuzzy differential equations," *Applied Soft Computing*, Vol. 13, No. 8, 2013, pp. 3597 – 3607.

[22] Rudd, K. and Ferrari, S., "A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks," *Neurocomputing*, Vol. 155, 2015, pp. 277–285.

[23] MacNeil, P., "A technique for generating approximate solutions and it's application to Coulomb interactions," *Southeastcon, 2012 Proceedings of IEEE*, March 2012, pp. 1–5.

[24] Howard, D. and Roberts, S. C., "Genetic programming solution of the convection-diffusion equation," *Proceedings of Genetic Evolutionary Computation Conference (GECCO-2001)*, 2001, pp. 34–41.

[25] Kirstukas, S. J., Bryden, K. M., and Ashlock, D. A., "A hybrid genetic programming approach for the analytical solution of differential equations," *International Journal of General Systems*, Vol. 34, 2005, pp. 279–299.

[26] Bryden, K. M., Ashlock, D. A., Corns, S., and Willson, S. J., "Graph-Based Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 5, 2006, pp. 550–567.

[27] Balasubramaniam, P. and Kumar, A. V. A., "Solution of matrix Riccati differential equation for nonlinear singular system using genetic programming," *Genetic Programming and Evolvable Machines*, Vol. 10, 2009, pp. 71–89.

[28] Seaton, T., Brown, G., and Miller, J. F., "Analytic Solutions to Differential Equations under Graph-Based Genetic Programming," *EuroGP LNCS*, Vol. 6021, 2010, pp. 232–243.

[29] Khan, J., Zahoor, R., and Qureshi, I., "Swarm Intelligence for the Solution of Problems in Differential Equations," *Environmental and Computer Science, 2009. ICECS '09. Second International Conference on*, Dec 2009, pp. 141–147.

[30] Mehrkanoon, S. and Suykens, J. A. K., "Learning solutions to partial differential equations using LS-SVM," *Neurocomputing*, Vol. 159, 2015, pp. 105–116.

[31] He, J., Xu, J., and Yao, X., "Solving Equations by Hybrid Evolutionary Computation Techniques," *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 3, 2000, pp. 295–304.

[32] Ramuhalli, P., Udpa, L., and Udpa, S. S., "Finite-Element Neural Networks for Solving Differential Equations," *IEEE Transactions on Neural Networks*, Vol. 16, No. 6, 2005, pp. 1381–1392.

[33] El-Emam, N. N. and Al-Rabeh., R. H., "An itelligent computing technique for fluid flow problems using hybrid adaptive neural network and genetic algorithm," *Applied Soft Computing*, Vol. 11, 2011, pp. 3283–3296.

[34] Veer, L. P. A. . P. V. D., "Neural Network Method for Solving Partial Differential Equations," *Neural Processing Letters*, Vol. 14, 2001, pp. 261–271.

[35] MacNeil, P. E., "Genetic Algorithms and Solutions of an Interesting Differential Equation," *Proceeding of the 10th Annual Genetic and Evoutionary Computation Conference (GECCO)*, July 2008, pp. 1711–1712.

[36] Yazdi, H. S. and Pourreza, R., "Unsupervised adaptive neural-fuzzy inference system for solving differential equations," *Applied Soft Computing*, Vol. 10, 2010, pp. 267–275.

[37] Raja, M. A. Z., Khan, J. A., and Qureshi, I. M., "Evolutionary Computational Intelligence in Solving the Fractional Differential Equations," *Lecture Notes in Computer Science*, Vol. 5990, 2010, pp. 231–240.

[38] MacNeil, P. E. and Schultz, S. R., "A Genetic Algorithm Approach to the Solution of a Differential Equation," *Proceeding of the 2010 IEEE Southeastern Conference*, March 2010, pp. 448–450.

[39] Raja, M. A. Z., Khan, J. A., and Qureshi, I. M., "A new stochastic approach for solution of Riccati differential equation of fractional order," *Annals of Mathematics and Artificial Intelligence*, 2011.

[40] Zjavka, L. and Abraham, A., "Failure and Power Utilization System Models of Differential Equations by Polynomial Neural Networks," *International Conference on Hybrid Intelligent Systems (HIS)*, 2013, pp. 273–278.

[41] Zjavka, L. and Snasel, V., "Constructing Ordinary Sum Differential Equations Using Polynomial Networks," *Information Sciences*, Vol. 281, 2014, pp. 462–477.

[42] Raja, M. A. Z., "Numerical treatment for boundary value problems of Pantograph functional differential equation using computational intelligence algorithms," *Applied Soft Computing*, Vol. 24, 2014, pp. 806–821.

[43] Monteagudo, A. and Santos, J., "Studying the capability of different cancer hallmarks to initiate tumor growth using a cellular automaton simulation. Application in a cancer stem cell context," *Biosystems*, Vol. 115, 2014, pp. 46–58.

[44] Santos, J. and Monteagudo, A., "Analysis of behaviour transitions in tumour growth using a cellular automaton simulation, , DOI: 10.1049/iet-syb.2014.0015," *IET Systems Biology*, 2014.

[45] Storn, R. and Price, K., "Differential Evolution: A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Global Optimization*, Vol. 11, 1997, pp. 341–359.

[46] O'Neill, M. and Ryan, C., "Grammatical Evolution," *IEEE Transactions on Evolutionary Computation*, Vol. 5, 2001, pp. 349–358.

[47] Tsoulos, I., Gavrilis, D., and Glavas, E., "Neural network construction and training using grammatical evolution," *Neurocomputing*, Vol. 72, 2008, pp. 269–277.

[48] Kennedy, J. and Eberhart, R., "Particle Swarm Optimization," *Proceedings of IEEE International Conference on Neural Networks IV*, 1995, pp. 1942–1948.

[49] Shi, Y. and Eberhart, R., "A modified particle swarm optimizer," *Proceedings of IEEE International Conference on Evolutionary Computation*, 1998, pp. 69–73.

[50] Kennedy, J. and Eberhart, R., *Swarm Intelligence*, Morgan Kaufmann, isbn 1-55860-595-9 ed., 2001.

[51] Poli, R., "Analysis of the publications on the applications of particle swarm optimisation," *Artificial Evolution and Applications*, 2008, pp. 1–10.

[52] Vapnik, V. and Lerner, A., "Pattern recognition using generalized portrait method," *Automation and Remote Control*, Vol. 24, 1963, pp. 774–780.

[53] Vapnik, V. and Chervonenkis, A., "A note on one class of perceptrons," *Automation and Remote Control*, Vol. 25, 1964.

[54] Boser, B. E., Guyon, I. M., and Vapnik, V. N., "A training algorithm for optimal margin classifiers," *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*, 1992.

[55] Cortes, C. and Vapnik, V., "Support-vector networks," *Machine Learning*, Vol. 20, No. 3, 1995, pp. 273–297.

[56] Ryan, C., Collins, J., and O'Neill, M., *Genetic Programming: First European Workshop, EuroGP'98*, chap. Grammatical evolution: Evolving programs for an arbitrary language, Springer Berlin Heidelberg, April 1998, pp. 83–96.

[57] O'Neill, M. and Ryan, C., "Crossover in Grammatical Evolution," *Genetic Programming and Evolvable Machines*, Vol. 4, 2003, pp. 67–93.

[58] O'Neill, M., Brabazon, A., Nicolau, M., Garraghy, S. M., and Keenan, P., "pi-Grammatical Evolution," *GECCO*, 2004, pp. 617–629.

[59] Harper, R. and Blair, A., "A Structure Preserving Crossover In Grammatical Evolution," *IEEE Evolutionary Computation*, Vol. 3, 2005, pp. 2537–2544.

[60] Harper, R. and Blair, A., "A Self-Selecting Crossover Operator," *IEEE Congress on Evolutionary Computation*, July 2006, pp. 1420–1427.

[61] Harper, R. and Blair, A., "Dynamically Defined Functions In Grammatical Evolution," *IEEE Congress on Evolutionary Computation*, Vol. 1, 2006, pp. 2638 – 2645.

[62] Nicolau, M. and Dempsey, I., "Introducing Grammar Based Extensions for Grammatical Evolution," *IEEE Congress on Evolutionary Computation*, 2006, pp. 648–655.

[63] Ortega, A., de la Cruz, M., and Alfonseca, M., "Christiansen Grammar Evolution: Grammatical Evolution With Semantics," *IEEE Transactions on Evolutionary Computation*, Vol. 11, 2007, pp. 77–90.

[64] Mingo, J. M. and Aler, R., "The Role Of The Lamarck Hypothesis In The Grammatical Evolution Guided By Reinforcement," *Latin America Transactions*, Vol. 6, 2008, pp. 500–504.

[65] Byrne, J., O'Neil, M., Hemberg, E., and Brabazon, A., "Analysis of Constant Creation Techniques on the Binomial-3 Problem with Grammatical Evolution," *IEEE Congress on Evolutionary Computation*, Vol. 1, 2009, pp. 568–573.

[66] Langdon, W. B. and Gustafson, S. M., "Genetic Programming and Evolvable Machines: ten years of reviews," *Genetic Programming and Evolvable Machines*, Vol. 11, September 2010, pp. 321–338.

[67] Alfonseca, M. and Ortega, A., "Book Review: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language," *Genetic Programming and Evolvable Machines*, Vol. 5, No. 4, 2004, pp. 393–393.

[68] Swafford, J., *Analyzing the Discovery and Use of Modules in Grammatical Evolution*, Master's thesis, University College, Dublin, 2013.

[69] Fagan, D., *An analysis of genotype-phenotype mapping in Grammatical Evolution*, Master's thesis, University College, Dublin, 2013.

[70] Murphy, E., *An Exploration of Tree-Adjoining Grammars for Grammatical Evolution*, Master's thesis, University College Dublin, 2013.

[71] Beyer, H. G. and Schwefel, H. P., "Evolution strategies - A comprehensive introduction." *Natural Computing*, Vol. 1, 2002, pp. 3–52.

[72] Hansen, N., Arnold, D. V., and Auger, A., "Evolution Strategies," Unpublished.

[73] Ohkura, K., Matsumura, Y., and Ueda, K., "Robust Evolution Strategies," *Applied Intelligence*, Vol. 15, 2001, pp. 153–169.

[74] Leung, K.-S. and Liang, Y., "Evolution Strategies with a Fourier Series Auxiliary Function for Difficult Function Optimization," *Lecture Notes in Computer Science*, Vol. 2690, 2003, pp. 303–312.

[75] Shir, O. M. and Back, T., "Dynamic Niching in Evolution Strategies with Covariance Matrix Adaptation," *Congress on Evolutionary Computation CEC-2005*, Vol. 1, 2005, pp. 2584–2591.

[76] Shir, O. M. and Back, T., "Niching in Evolution Strategies," *Conference on Genetic and evolutionary computation, GECCO*, 2005.

[77] Kramer, O. and Schwefel, H.-P., "On three new approaches to handle constraints within evolution strategies," *Natural Computing*, Vol. 5, 2006, pp. 363–385.

[78] Debski, R., Drezewski, R., and Kisiel-Dorohinicki, M., "Maintaining Population Diversity in Evolution Strategy for Engineering Problems," *IEA/AIE*, Vol. 1, 2008, pp. 379–387.

[79] Wierstra, D., Schaul, T., Peters, J., and Schmidhuber, J., "Natural Evolution Strategies," *Proceedings of IEEE Congress on Evolutionary Computation (CEC-2008, Hongkong)*, 2008.

[80] Glasmachers, T., Schaul, T., and Schmidhuber, J., "A Natural Evolution Strategy for Multi-objective Optimization," *Lecture Notes in Computer Science*, Vol. 6238, 2011, pp. 627–636.

[81] Eiben, A. E. and Smith, J. E., *Introduction to Evolutionary Computing*, Springer-Verlag Berlin Heidelberg, 2nd ed., 2007.

[82] Hansen, N., "The CMA Evolution Strategy: A Comparing Review," *Towards a New Evolutionary Computation*, edited by J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, Vol. 192 of *Studies in Fuzziness and Soft Computing*, Springer Berlin Heidelberg, 2006, pp. 75–102.

[83] Hansen, N., "The CMA Evolution Strategy: A Tutorial," https://www.lri.fr/~hansen/cmatutorial.pdf, June 2011.

[84] Hohm, T. and Zitzler, E., "Multicellular pattern formation," *Engineering in Medicine and Biology Magazine, IEEE*, Vol. 28, No. 4, July 2009, pp. 52–57.

[85] Santamaria, J., O. Cordon, O., Damas, S., and Ibanez, O., "3D-2D image registration for craniofacial superimposition in forensic medicine using covariance matrix adaptation evolution strategy," *Information Technology and Applications in Biomedicine, 2009. ITAB 2009. 9th International Conference on*, 2009.

[86] Ibanez, O., Cordon, O., Damas, S., and Santamaria, J., "Modeling the Skull-Face Overlay Uncertainty Using Fuzzy Sets," *Fuzzy Systems, IEEE Transactions on*, Vol. 19, No. 5, Oct 2011, pp. 946–959.

[87] Colutto, S., Fruhauf, F., Fuchs, M., and Scherzer, O., "The CMA-ES on Riemannian Manifolds to Reconstruct Shapes in 3-D Voxel Images," *IEEE Trans. Evol. Comput.*, Vol. 14, No. 2, april 2010, pp. 227–245.

[88] Fang, X. S., Chow, C. K., Leung, K.-W., and Lim, E. H., "New Single-/Dual-Mode Design Formulas of the Rectangular Dielectric Resonator Antenna Using Covariance Matrix Adaptation Evolutionary Strategy," *Antennas and Wireless Propagation Letters, IEEE*, Vol. 10, 2011, pp. 734–737.

[89] Gong, R. H., Stewart, J., and Abolmaesumi, P., "Multiple-Object 2-D-3-D Registration for Noninvasive Pose Identification of Fracture Fragments," *Biomedical Engineering, IEEE Transactions on*, Vol. 58, No. 6, June 2011, pp. 1592–1601.

[90] Gregory, M., Namin, F., and Werner, D., "Exploiting Rotational Symmetry for the Design of Ultra-Wideband Planar Phased Array Layouts," *Antennas and Propagation, IEEE Transactions on*, Vol. 61, No. 1, Jan 2013, pp. 176–184.

[91] Nelder, J. A. and Mead, R., "A Simplex Method for Function Minimization," *The Computer Journal*, Vol. 7, No. 4, 1965, pp. 308–313.

[92] Robin, F., Orzati, A., Moreno, E., Homan, O., and Bachtold, W., "Simulation and evolutionary optimization of electron-beam lithography with genetic and simplex-downhill algorithms," *Evolutionary Computation, IEEE Transactions on*, Vol. 7, No. 1, Feb 2003, pp. 69–82.

[93] Floridia, C. and de Moraes, J., "Fast on-line OSNR measurements based on polarisation-nulling method with downhill simplex algorithm," *Electronics Letters*, Vol. 44, No. 15, July 2008, pp. 926–927.

[94] Hansen, N., "Benchmarking the nelder-mead downhill simplex algorithm with many local restarts," *GECCO '09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, 2009.

[95] Anet Neto, L., Erasme, D., Genay, N., Chanclou, P., Deniel, Q., Traore, F., Anfray, T., Hmadou, R., and Aupetit-Berthelemot, C., "Simple Estimation of Fiber Dispersion and Laser Chirp Parameters Using the Downhill Simplex Fitting Algorithm," *Lightwave Technology, Journal of*, Vol. 31, No. 2, Jan 2013, pp. 334–342.

[96] Bauer, C., Frink, A., and Kreckel, R., "Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language," *Symbolic Computation*, Vol. 33, 2002, pp. 1–12.

[97] CLN, "Class Library for Numbers," http://www.ginac.de/CLN/.

[98] Rao, K. R. and Yip, P., *Discrete Cosine Transform. Algorithms, Advantages and Applications*, Academic Press, Inc, 1990.

[99] Ghodadra, B. L., "Order of magnitude of multiple Fourier coefficients of functions of bounded p-variation," *Acta Mathematica Hungarica*, Vol. 22, No. 3, 2010, pp. 187–198.

[100] Hangelbroek, T. and Ron, A., "Nonlinear approximation using Gaussian kernels," *Journal of Functional Analysis*, Vol. 259, No. 1, 2010, pp. 203 – 219.

[101] Hansen, N. and Kern, S., "Evaluating the CMA Evolution Strategy on Multimodal Test Functions," *Eighth International Conference on Parallel Problem Solving from Nature PPSN VIII*, Springer, 2004, pp. 282–291.

[102] Press, W. H., Vetterling, W. T., Teukolsky, S. A., and Flannery, B. P., *Numerical Recipes in C++: the art of scientific computing*, Cambridge University Press, New York, NY, USA, 2nd ed., 2002.

[103] Powell, M. J. D., "A Tolerant Algorithm for Linearly Constrained Optimization Calculations," *Mathematical Programming*, Vol. 45, 1989, pp. 547–566.

[104] Zhao, X., "Stream Function Solution of Transonic Flow Along S2 Streamsurface of Axial Turbomachines," *Journal of Engineering for Gas Turbines and Power*, Vol. 108, No. 1, 1986, pp. 138–143.

[105] Hairer, E. and Wanner, G., *Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems*, Springer-Verlag, second revised edition ed., 1996.

[106] Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-P., Auger, A., and Tiwari, S., "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Tech. rep., Nanyang Technological Univ., Singapore, 2007.

[107] Rudolph, G., "On Correlated Mutations in Evolutionary Strategies," *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature*, edited by R. Manner and B. Manderick, Elsevier, 1992, p. 105.114.