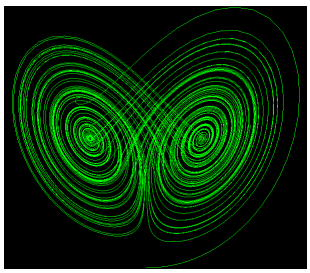
Departments of Mathematics, Physics and DACSO

MASTER THESIS

Modelling for Science and Engineering

Center Problem for 3-Monomial Systems Using Parallelization

Oscar Saleta Reig



Setembre 2016

UAB
Universitat Autònoma de Barcelona

Abstract

The study of singular points of planar systems of differential equations in order to distinguish weak foci from centers often requires the computation of Lyapunov constants, which is a complex problem, proven hard to tackle even for powerful modern computers.

In this work, we will introduce a parallelization tool and a new set of operations that act on polynomials as if they were vectors. These tools, allow us to take a first step towards solving the problem of characterising the centers of polynomial systems with three monomial nonlinearities. In the last chapter, we provide preliminary results in this direction, studying the homogeneous and non homogeneous cases of degree 3 in which the first monomial is non resonant.

Acknowledgements

En primer lloc, agraeixo de tot cor a Joan Torregrosa la seva dedicació durant mesos, des del primer dia en què em va explicar què és una constant de Lyapunov fins l'entrega d'aquest treball, i més enllà. A més, dono les gràcies a Marina per tot, als meus pares i germà pel seu suport, i al lector pel seu temps.

Oscar Saleta Reig, September 1, 2016.

Contents

Chapter 1

Introduction

1.1 Motivation of this work

In this Masters Thesis we aim to study the theoretical framework around the center-focus problem, and to develop a set of computational tools in order to produce a solid ground from where to solve problems that require the symbolic computation of Lyapunov constants. In particular, we aspire to take a first step towards the characterization of all the possible centers for a family of planar differential polynomial equations with three monomial nonlinearities.

The layout of this work is the following. In this first chapter, we will introduce the reader to the center-focus problem, the mathematical notation that we are going to use, and several basic definitions such as *first integral* and *Lyapunov constant*. Then, in the second chapter we will explain how to systematically compute the Lyapunov constants of a given complex differential equation, with some considerations about non *well posed* systems.

In the third chapter we will discuss many improvements to the algorithm previously introduced. In this chapter we will explain the development of a set of operations that transform polynomial algebra in Computer Algebra Systems to vector operations, which increases the speed of computation. We will also introduce our parallelization interface PBala, which is a piece of software that allows users to seamlessly distribute executions in a cluster and will be a key factor for being able to simulate many systems of differential equations at once. Here we will also introduce some algebraic concepts that will be used later on, such as the *Gröbner basis* of an ideal in a polynomial ring.

The fourth chapter will serve as a test for all the tools we developed. We will try to recover some results from previous research and also use the performance of our computational software to check a conjecture for higher degree systems than what the

original authors could compute.

In the fifth chapter, we will take the first step towards the classification and exhaustive characterization of centers for planal differential polynomial equations that are expressed with three monomial nonlinearities in complex variable. Here a first theorem of characterization of centers will be given, and we will discuss the future possibility of computing the conditions in which the theorem gives an exhaustive classification.

Finally, in Chapter 6 we will talk about the future work that will follow the content of this Masters Thesis.

1.2 Center-focus problem

Consider a system of planar differential equations with a singular point at the origin $(x, y) = (0, 0)$, under a suitable coordinates transformation such that the coefficients matrix for the linear part is in canonical form (Hirsch, Smale, & Devaney, 2004):

$$\begin{cases} \dot{x} &= \lambda x - y + P(x, y), \\ \dot{y} &= x + \lambda y + Q(x, y). \end{cases} \tag{1.1}$$

Here, P and Q are real analytic functions without constant nor linear terms. From Hartman-Grobman, we know that if $\lambda \neq 0$, the singular point is hyperbolic and it is either a stable strong focus ($\lambda < 0$) or an unstable one ($\lambda > 0$) (Hartman, 1960). If we consider $\lambda = 0$, we cannot use the linear components to distinguish between a center and a focus, and our attention has to turn to the higher degree terms.

This problem is known as the *center-focus problem*, and has motivated many studies and the development of several methods that aim to find necessary and sufficient conditions on P and Q for the origin to be a center (Dumortier, Llibre, & Artés, 2006).

From Poincaré, we know that if the origin is a center then there exists an analytic first integral (Ilyashenko & Yakovenko, 2006), so if we could construct such an object by imposing the existence of this analytic function and we found an impassable barrier we would be able to affirm that the singular point is not a center, but a focus. Recently, many practical methods for this construction arise from the use of computer algebra systems and the computation capabilities of modern computers. However, these methods can only tackle a small portion of the problem, such as restricted families of systems (e.g. when P and Q are homogeneous, real polynomials of degree 2, Zoladek, 1994 and when they are homogeneous, real polynomials of degree 3, Dukarić, Giné, & Llibre, 2016). Another studied subfamily of systems is that where P and Q satisfy that $R = P + iQ$ only has two monomials in complex coordinates (Gasull, Giné, & Torregrosa, 2016). The last chapter of this work is based in the last paper.

1.3 Complex coordinates

In order to simplify the notation, we will use complex coordinates to treat this problem. If we apply the change $z = x + iy$ the system (1.1) with $\lambda = 0$ can be written as

$$\dot{z} = iz + R(z, \overline{z}), \qquad \dot{\overline{z}} = -i\overline{z} + \overline{R}(z, \overline{z}). \tag{1.2}$$

Here R is an analytic function with complex coefficients defined as $R = P + iQ$.

Lemma 1.1. *Let $P(x, y) \in \mathbb{R}[x, y]$ a polynomial with real coefficients, and $R(z, \overline{z})$ its transformation to complex variable. Then, the coefficient $r_{m,n}$ corresponding to the term $z^m\overline{z}^n$ satisfies that $r_{m,n} = \overline{r_{n,m}}$ for any m, n.*

Proof. We can firstly consider a single monomial $p_{k,\ell}x^k y^\ell$. Notice that, since $p_{k,\ell} \in \mathbb{R}$, we can consider $p_{k,\ell} = 1$ without loss of generality. Thus, we consider $x^k y^\ell$. Written in complex variable:

$$x = \frac{z + \overline{z}}{2}, \qquad y = \frac{z - \overline{z}}{2i}, \tag{1.3}$$

$$x^k y^\ell = \left(\frac{z + \overline{z}}{2}\right)^k \left(\frac{z - \overline{z}}{2i}\right)^\ell := f(z, \overline{z}). \tag{1.4}$$

We can expand f using the binomial expansion:

$$\begin{aligned} f(z, \overline{z}) &= \frac{1}{2^{k+\ell} i^\ell} \sum_{n=0}^{k} \binom{k}{n} z^{k-n}\overline{z}^n \sum_{m=0}^{\ell} (-1)^m z^{\ell-m}\overline{z}^m \\ &= \frac{1}{2^{k+\ell} i^l} \left(\binom{k}{0} z^k + \binom{k}{1} z^{k-1}\overline{z} + \cdots + \binom{k}{k-1} z\overline{z}^{k-1} + \binom{k}{k}\overline{z}^k \right) \\ &\quad \left(\binom{\ell}{0} z^\ell - \binom{\ell}{1} z^{\ell-1}\overline{z} + \cdots + \binom{\ell}{\ell-1}(-1)^{\ell-1} z\overline{z}^{\ell-1} + \binom{\ell}{\ell}(-1)^\ell \overline{z}^\ell \right). \end{aligned}$$

Let us compute $\overline{f(\overline{z}, z)}$:

$$\begin{aligned} \overline{f(\overline{z}, z)} &= \frac{1}{2^{k+\ell}(-1)^\ell i^\ell} \left(\binom{k}{0}\overline{z}^k + \binom{k}{1}\overline{z}^{k-1} z + \cdots + \binom{k}{k-1} z^{k-1}\overline{z} + \binom{k}{k} z^k \right) \\ &\quad \left(\binom{\ell}{0}\overline{z}^\ell - \binom{\ell}{1}\overline{z}^{\ell-1} z + \cdots + \binom{\ell}{\ell-1}(-1)^{\ell-1}\overline{z} z^{\ell-1} + \binom{\ell}{\ell}(-1)^\ell z^\ell \right) \\ &= \frac{1}{2^{k+\ell} i^\ell} \left(\binom{k}{k} z^k + \binom{k}{k-1} z^{k-1}\overline{z} + \cdots + \binom{k}{1} z\overline{z}^{k-1} + \binom{k}{0}\overline{z}^k \right) \\ &\quad \left(\binom{\ell}{\ell}(-1)^{2\ell} z^\ell + \binom{\ell}{\ell-1}(-1)^{2\ell-1} + \cdots + \binom{\ell}{1}(-1)^{\ell+1} z\overline{z}^{\ell-1} + \binom{l}{0}(-1)^\ell \overline{z}^\ell \right) \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2^{k+\ell} i^{\ell}} \left(\binom{k}{k} z^k + \binom{k}{k-1} z^{k-1}\overline{z} + \cdots + \binom{k}{1} z\overline{z}^{k-1} + \binom{k}{0} \overline{z}^k \right) \\
&\left(\binom{\ell}{\ell} z^{\ell} - \binom{\ell}{\ell-1} z^{\ell-1}\overline{z} + \cdots + \binom{\ell}{1} (-1)^{\ell-1} z\overline{z}^{\ell-1} + \binom{\ell}{0} (-1)^{\ell} \overline{z}^{\ell} \right) \\
&= f(z, \overline{z}),
\end{aligned}
$$

because binomial coefficients are symmetric $\binom{k}{n} = \binom{k}{k-n}$.

Therefore, the equality is satisfied for every monomial of the real polynomial, which assures that the final complex polynomial will satisfy the property. □

This lemma shows a property that we will be able to use, because all our complex polynomials will be of the for $R = P + iQ$, where P and Q are real polynomials.

1.4 First integral and Lyapunov constants

Definition 1.2. Given a $\mathbb{C}$-polynomial system such as (1.2), its associated vector field is

$$Z = \dot{z}\frac{\partial}{\partial z} + \dot{\overline{z}}\frac{\partial}{\partial \overline{z}}. \tag{1.5}$$

Then, the system is *integrable* on an open subset U of $\mathbb{C}$ if there exists a nonconstant analytic function $H : U \to \mathbb{C}$, called a *first integral* of the system on U, such that H is constant along all solution curves $(z(t), \overline{z}(t))$ of (1.2). That is, $H(z(t), \overline{z}(t)) = h$ for some constant h, for all t for which the solution is defined inside U (Dumortier et al., 2006).

It is trivial to see that $H(z, \overline{z})$ is a first integral if and only if $ZH = (iz + R)H_z + (-i\overline{z} + \overline{R})H_{\overline{z}} = 0$ on U, because the solution curves $(z(t), \overline{z}(t))$ follow Z.

Our system (1.2) has a singular point at the origin. If we only consider the linear part, the singular point is a center, and $H = z\overline{z}$ is a first integral:

$$H' = H_z\dot{z} + H_{\overline{z}}\dot{\overline{z}} = \overline{z}(iz) + z(-i\overline{z}) = 0.$$

Therefore, we can consider a first integral for the complete case (if it exists) of the form

$$H = z\overline{z} + \sum_{k=3}^{\infty} H_k(z, \overline{z}), \tag{1.6}$$

which is a perturbation of the linear case by an analytic function in a neighbourhood of the origin. H_k are homogeneous complex polynomials in $z, \overline{z}$ of degree k:

$$H_k = \sum_{j=0}^{k} h_{k-j,j} z^{k-j} \overline{z}^{j}. \tag{1.7}$$

We start at $k = 3$ because the first term of the original first integral is already of degree 2. The following lemma will give us a direct method for distinguishing between a center and a focus.

Lemma 1.3 (Songling, 1981)**.** *Given the planar system of differential equations* (1.2)*, there exists a formal power series:*

$$F = z\overline{z} + \sum_{k=3}^{\infty} F_k(z, \overline{z}),$$

such that

$$\frac{\mathrm{d}F}{\mathrm{d}t} = F_z \dot{z} + F_{\overline{z}} \dot{\overline{z}} = \sum_{k=1}^{\infty} -V_k (z\overline{z})^{k+1}$$

The main idea for understanding this lemma will be clear in the following chapter, where we explain the computational method derived from it.

Observation 1.4. In Lemma 1.3, if $V_k = 0$ for all $k > 0$, we have that $F(z, \overline{z}) = H(z, \overline{z})$ is a first integral of the system, and therefore the singular point at the origin is a center.

Definition 1.5. The quantities $V_k \in \mathbb{R}[\{r_{m,n}\}]$, where $\mathbb{R}[\{r_{m,n}\}]$ denotes the multivariate polynomial ring in the coefficients of $R(z, \overline{z})$ of the system (1.2) over the real field, are called *Lyapunov constants* (or *focal values*, or *center conditions*). Also, given a polynomial system, the first nonzero Lyapunov constant is called the *first significant* Lyapunov constant.

Our method for distinguishing between weak foci and centers is based on the existence of this first integral $H(z, \overline{z})$. We will explain the method used for its construction more in depth in the following chapters (see e.g. Songling, 1984 for a similar explanation using real variables).

Observation 1.6. Notice that we only consider *well posed* systems with singular points in the origin, i.e. systems of the form (1.2) with $\lambda = 0$ with the singularity at the origin. If we wanted to analyse systems with singularities outside of the origin, we would have to apply a transformation to the system in order to place the singularity at the origin, which would render a system generally not in the form of (1.2). Christopher (2006), deals with these kind of systems that cannot be forced to have a first integral like (1.6), using a general quadratic term $H_2(x, y) = Ax^2 + Bxy + Cy^2$. The computational method that follows in order to determine if the singular point is a center or a weak focus is very similar to what we will use, and we will comment it a bit in a later chapter although we will not apply these generalizations in this work because they will not be needed.

1.5 Limit cycles and the 16th Hilbert problem

Definition 1.7. For a system of differential equations with unique solution outside of the origin, such as (1.1), a *periodic orbit* of the system is a curve given by a periodic nonconstant solution $(f(t), g(t))$.

Definition 1.8. Given a periodic orbit γ of a system, we call it an *externally (internally) limit cycle* if for an arbitrarily small outer (inner) neighbourhood of γ there do not exist other periodic orbits (Dumortier et al., 2006).

The study of a higher bound for the number of limit cycles that can bifurcate from a center-focus singular point is an interesting and difficult problem in the theory of planar differential equations. See Giné (2007), Llibre, Ramírez, and Sadovskaia (2010) for studies in which a maximum number of limit cycles for certain systems is given. This number is closely related to the number of functionally independent Lyapunov constants that can be computed from a system. For further reference, Giné's conjecture for the maximum number of functionally independent Lyapunov constants (and, under some assumptions, of limit cycles) is

$$N = n^2 - 3n + 7, \tag{1.8}$$

where n is the degree of $R(z, \overline{z})$ in (1.2).

In 1902, David Hilbert stated his famous 23 mathematical problems which influenced greatly the Mathematics of the twentieth century. Problem number 16 (*Problem of the topology of algebraic curves and surfaces*) asks about the relative position of branches of a $n-$th order algebraic curve in the plane, and as a related question brings the problem of finding an upper bound for the limit cycles in planar differential equations of degree n and an investigation of their relative positions.

This is still an open problem in which much research has been performed during both the 20th and 21st centuries. The practical possibility of computing higher order Lyapunov constants for certain systems of differential equations, using both elaborated methods and powerful computers, has provided researches with tools to tackle this problem with a deeper understanding on how the systems behave, but the complete solution for any n seems still a cyclopean task to resolve for the present state of both Mathematics and Computer Science.

Chapter 2

Computation of the Lyapunov constants

2.1 Iterative construction of a formal first integral

Observation 2.1. During this chapter, we will use the following notation: if $H(z,\overline{z})$ is a complex function of z and $\overline{z}$, then H_z and $H_{\overline{z}}$ will denote $\partial H/\partial z$ and $\partial H/\partial \overline{z}$, respectively. On the other hand, if $H(z,\overline{z})$ can be written as a polynomial or a formal power series, expressions such as H_k, where k is an integer, will be used to denote "the terms of $H(z,\overline{z})$ of degree k", with no partial derivative meaning whatsoever. This convention is used to maintain a simple notation throughout this work, even though it might seem confusing at first.

We can try and generate an object like (1.6) step by step, constructing H_3, H_4 and so on[1], always imposing that the intermediate result is a first integral (i.e. $\dot{H} = 0$ over the solution curves for the truncated first integral up to the order of the step). This consists simply in computing the quantities V_i from Lemma 1.3 and forcing them to be zero before continuing with the next step.

The first step is $k = 3$. Let $H = z\overline{z} + H_3$, and we impose the condition of first integral using the following notation:

$$R = \sum_{j=2}^{\infty} R_j(z,\overline{z}) = \sum_{j=2}^{\infty} \sum_{\ell=0}^{j} r_{j-\ell,\ell} z^{j-\ell} \overline{z}^{\ell}, \tag{2.1}$$

$$H_k = \sum_{\ell=0}^{k} h_{k-\ell,\ell} z^{k-\ell} \overline{z}^{\ell}, \tag{2.2}$$

[1] Notice the use of H_3 to denote "the terms of $H(z,\overline{z})$ of degree 3", since H is a formal power series.

Then, the total derivative of H is

$$\begin{aligned}\dot{H} &= H_z\dot{z} + H_{\overline{z}}\dot{\overline{z}} \\ &= (\overline{z} + 3h_{3,0}z^2 + 2h_{2,1}z\overline{z} + h_{1,2}\overline{z})(iz + R) \\ &+ (z + h_{2,1}z + 2h_{1,2}z\overline{z} + 3h_{0,3}\overline{z}^2)(-i\overline{z} + \overline{R}).\end{aligned}$$

We can separate the computation of $\dot{H}$ in each step in two parts: $H_z(iz) + H_{\overline{z}}(-i\overline{z})$ and $H_zR + H_{\overline{z}}\overline{R}$.

Observation 2.2. The first part will contain terms of the highest order of H, because H_z and $H_{\overline{z}}$ have degree one less than H, and we are multiplying by a monomial of degree one (and the total degree must be equal to that of H). This means that, in step k[2], the term $H_z(iz) + H_{\overline{z}}(-i\overline{z})$ will contain the coefficients of H_k, which are unknown. On the other hand, $H_zR + H_{\overline{z}}\overline{R}$ will always involve terms of H of lower degrees (from H_{k-1} to H_3 for step $k > 3$ or none for $k = 3$).

Thus, for $k = 3$ we have

$$\begin{aligned}H_z iz &= (3h_{3,0}z^3 + 2h_{2,1}z^2\overline{z} + h_{1,2}z\overline{z}^2)i, \\ H_{\overline{z}}(-i\overline{z}) &= -(h_{2,1}z^2\overline{z} + 2h_{1,2}z\overline{z}^2 + 3h_{0,3}\overline{z}^3)i, \\ H_z iz - H_{\overline{z}}i\overline{z} &= (3h_{3,0}z^3 + h_{2,1}z^2\overline{z} - h_{1,2}z\overline{z}^2 - 3h_{0,3}\overline{z}^3)i.\end{aligned}$$

Recall that we want $\dot{H} = 0$, so we can write $H_z(iz) + H_{\overline{z}}(-i\overline{z}) = -H_zR - H_{\overline{z}}\overline{R}$. This is a system with $k + 1$ equations and $k + 1$ unknowns, and the matrix of coefficients is given by the left hand side, which we already computed in this case:

$$\begin{pmatrix}3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -3\end{pmatrix}\begin{pmatrix}h_{3,0} \\ h_{2,1} \\ h_{1,2} \\ h_{0,3}\end{pmatrix} = -\frac{H_zR + H_{\overline{z}}\overline{R}}{i}.$$

It is straigthforward to see that the system will always be diagonal if we use complex variables[3]. Thus, the difficulty of this algorithm will reside in the computation and storage of the independent terms, which are polynomials of as many variables as parameters R has (Dumortier et al., 2006) (in general, they would be analytic functions, but in practice we deal with polynomic fields of finite degree because $R(z, \overline{z})$ is considered to be a polynomial with a finite number of monomials).

[2] Assuming the "first" step is $k = 3$.

[3] The idea is that the partial derivative H_z is compensated by the multiplication by iz and the same for $\overline{z}$, so the degree of each monomial always corresponds to the coefficient of that degree in H, thus falling into the diagonal of the matrix.

Once the right hand side has been computed, we obtain the coefficients of H_3 by a simple division. Now we can proceed to the next step, $k = 4$, using $h_{3,0}, h_{2,1}, h_{1,2}$ and $h_{0,3}$ as known parameters of our new system. A simple computation equivalent to what we did above will yield the following system matrix:

$$\begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & -4 \end{pmatrix}$$

This matrix is singular because it has a zero in the diagonal corresponding to the term of $(z\overline{z})^{k/2}$, which means that our system has no solution in general. The only solvable case would be when the corresponding right hand side expression was also zero, so we could obtain an indeterminate solution depending on one parameter.

Observation 2.3. Notice that these are the terms given by Songling $-V_k(z\overline{z})^{k+1}$: the system can only be solved if $V_k = 0$. Also notice that these terms only appear in even steps of the process, because odd steps cannot have a monomial of the form $(z\overline{z})^{k/2}$.

Therefore, in step $k = 4$ we can find $L_1 := V_3$. We will use the notation L_j for the $j-$th Lyapunov constant, starting at $j = 0$, where $L_0 = V_1$ is proportional to λ (see, e.g., Christopher, 2006, Timochouk, 1994), so $L_0 = 0$ because we consider $\lambda = 0$.

Observation 2.4. In our case, where $\lambda = 0$, the focal values are polynomials in $\mathbb{R}[a_1, a_2, b_1, b_2]$ (Dumortier et al., 2006), with an ever increasing number of monomials as new orders of Lyapunov constants are computed, which makes computational treatment of the Lyapunov constants a very troublesome problem to handle even with modern and powerful computers.

Notice that for every even degree we will obtain a singular system matrix, so even degrees of H will yield one Lyapunov constant each. There are, however, infinitely many Lyapunov constants. We cannot construct the *full* first integral H using this iterative process in order to assure that the origin is a center of the system. Nevertheless, the question of how many functionally independent Lyapunov constants can we find in a system of differential equations with a center-focus singularity has been studied by researchers such as Giné (2007), who conjectured in this work that the maximum number of functionally independent Lyapunov constants is $n^2 - 3n + 7$, as we mentioned in the previous chapter, where n is the degree of $R(z, \overline{z})$ from (1.2). This means that if we find as many null Lyapunov constants there is a strong case for considering that the singular point is a center.

2.2 Adaptation to *non-well posed* systems

As we mentioned in the Introduction, Colin Christopher (2006) treats the problem of applying the previous algorithm to systems that are not *well posed*, in the sense that they cannot be written in the simple form of (1.2). If a planar system cannot be written as in (1.2), we cannot expect it to admit a first integral of the form (1.6), because the equality $\dot{H} = 0$ will not hold even for degree 2.

For example, if we consider a system that has a singular point of center-focus type but outside of the origin, and we apply a transformation to bring the singular point to the origin, the system will be written generally as

$$\dot{z} = r_{1,0}z + r_{0,1}\overline{z} + R(z, \overline{z}). \tag{2.3}$$

This means that, instead of considering the formal power series given by Songling in Lemma 1.3, we will consider

$$F(z, \overline{z}) = h_{2,0}z^2 + h_{1,1}z\overline{z} + h_{0,2}\overline{z}^2 + \sum_{k=1}^{\infty} F_k(z, \overline{z}), \tag{2.4}$$

where $h_{j,k} \in \mathbb{C}$. In this case, we also want to force $F(z, \overline{z}) = H(z, \overline{z})$ to be a first integral of the system, but first one has to find $h_{2,0}, h_{1,1}, h_{0,2}$ such that $H = h_{2,0}z^2 + h_{1,1}z\overline{z} + h_{0,2}\overline{z}^2$ is a first integral of the system $\dot{z} = r_{1,0}z + r_{0,1}\overline{z}$ (in the well posed case, $r_{1,0} = i, r_{0,1} = 0$ and $h_{2,0} = h_{0,2} = 0, h_{1,1} = 1$).

Of course, using this method the underlying algorithm is unchanged: we still iterate order by order finding the conditions that must be held for the system of equations in $h_{j,k}$ to be solvable. However, in this case the coefficient matrices are not diagonal, because the terms $h_{2,0}z^2 + h_{0,2}\overline{z}^2$ will generate terms with uncompensated degrees that will fall outside the diagonal.

2.3 Gröbner basis, reduction of constants and ideals

In order to make sure that two Lyapunov constants (or multivariate polynomials in the same polynomial ring, in general) are functionally independent, we need to make use of the concept of Gröbner basis of an ideal in a polynomial ring.

In order to introduce the concept of a Gröbner basis we need some previous definitions, all taken from Becker and Weispfenning (1993) and Greuel and Pfister (2007).

Definition 2.5. A *monomial ordering* is a total (or linear) ordering in the set of monomials $\mathrm{Mon}_n = \{x^\alpha \mid \alpha \in \mathbb{N}^n\}$ in n variables satisfying

$$x^\alpha > x^\beta \Rightarrow x^\gamma x^\alpha > x^\gamma x^\beta,$$

for all $\alpha, \beta, \gamma \in \mathbb{N}^n$. We also say that $>$ is a monomial ordering on $A[x_1, \ldots, x_n]$, for A any ring, meaning that $>$ is a monomial ordering on Mon_n.

The concept of monomial ordering allows us to write a polynomial $f \in K[x]$ in a unique way, which is what we will use in order to define the polynomial-vector translation of Definition 3.3 in the next chapter.

In the following definitions we will consider K a field and $K[x]$ a general polynomial ring over K.

Definition 2.6. Given $>$ a monomial ordering and $f \in K[x]$, $f \neq 0$ written according to this ordering, the *leading monomial* of f is $LM(f) = x^\alpha$, where α is the highest exponent (also called *leading exponent*).

Definition 2.7. For any subset $G \subset K[x]$, define the ideal

$$L(G) := \langle LM(g) \mid g \in G \backslash \{0\} \rangle_{K[x]},$$

called the *leading ideal of* G.

Definition 2.8. Let $R = K[x_1, \ldots, x_n]$ and $I \subset R$ be an ideal. Then,

1. A finite set $G \subset R$ is called a *standard basis* of I if $G \subset I$ and $L(I) = L(G)$ (i.e., G is a standard basis if the leading monomials of the elements of G generate the leading ideal of I).
2. If the ordering on R is global, a standard basis is called a *Gröbner basis*.

Now we will present the last definition needed for this section:

Definition 2.9. Let $G \subset R$ be any subset. Then $f \in R$ is called *completely reduced with respect to* G if no monomial of the power series expansion of f is contained in $L(G)$.

Therefore, if we work in $R = \mathbb{R}[a_1, b_1, a_2, b_2]$ for computing Lyapunov constants, we can add the following procedures to the iterations of the algorithm previously explained:

(i) After computing the 3 first Lyapunov constants, define the ideal $I = \langle L_1, L_2 \rangle \subset R$ and compute its Gröbner basis B.

(ii) Check if $L_3 \in R$ is reduced with respect to B.

- If L_3 is completely reduced with respect to B, it means that it is functionally dependent on L_1 and L_2, so it will have the same zeros and does not contribute with new information to the set of Lyapunov constants. Therefore, we discard this constant as zero.

- If L_3 is not completely reduced with respect to B, we take only the non reducible part (the monomials which do not belong to B) and redefine L_3 as these monomials. Then we add L_3 to the list of computed Lyapunov constants.

(iii) Repeat this process at every step, each time adding the new reduced Lyapunov constants to I if they were not completely reduced with respect to B.

Observation 2.10. This method can take out many superfluous Lyapunov constants that we do not need. However, it is not perfect. If L_j is not completely reduced with respect to the Gröbner basis of I for the previous constants that were added to the list but L_j^k is, then this constant will not add further information to the center conditions previously contained in I although we will add L_j to the list based on the criteria of the algorithm (Gasull & Torregrosa, 2001).

Chapter 3

Improvements to the algorithm

3.1 Scale issues for computation

The procedure for computing Lyapunov constants shown in the previous chapter is well known (Christopher, 2006; Songling, 1981, 1984), but it puts a lot of strain on computers. For instance, CAS (Computer Algebra Systems) have a hard time computing derivatives, Taylor expansions (to pick only terms of a certain degree at each step) and so on. On the other hand, memory requirements for these operations are really high, and even computers with hundreds of GB of RAM can fall short. We have to take into account that, for a general field of degree 3 ($R = r_{2,0}z^2 + r_{1,1}z\overline{z} + r_{0,2}\overline{z}^2 + r_{3,0}z^3 + r_{2,1}z^2\overline{z} + r_{1,2}z\overline{z}^2 + r_{0,3}\overline{z}^3$) the fourth Lyapunov constant is a polynomial in $\mathbb{R}[r_{2,0}, r_{1,1}, r_{0,2}, r_{3,0}, r_{2,1}, r_{1,2}, r_{0,3}, \overline{r_{2,0}}, \overline{r_{1,1}}, \overline{r_{0,2}}, \overline{r_{3,0}}, \overline{r_{2,1}}, \overline{r_{1,2}}, \overline{r_{0,3}}]$ with already 2644 monomials[1], and each step depends on all the variables computed in previous steps, so everything has to be stored during computations.

Table 3.1 shows the number of monomials of the first 5 Lyapunov constants (as computed by the algorithm explained in the previous chapter), along with the memory used during their computation. The fifth constant already takes nearly 5GB of RAM and 12 minutes to compute (and this is using Maple 18, which automatically parallelizes computations when possible). It is clear that it is not feasible to compute higher order Lyapunov constants using this approach.

[1] This means that the other 9 variables $h_{i,j}$ in degree 10 (which is the degree where the 4th Lyapunov constant is computed) are also polynomials of this size.

Table 3.1: Number of monomials, size in memory of the objects used during the computation, and time spent computing, for the first five Lyapunov constants of the system $\dot{z} = iz + r_{2,0}z^2 + r_{1,1}z\overline{z} + r_{0,2}\overline{z}^2 + r_{3,0}z^3 + r_{2,1}z^2\overline{z} + r_{1,2}z\overline{z}^2 + r_{0,3}\overline{z}^3$. Computations for this table were performed using Maple 18 with an Intel Core 2 Duo E8400@3.00GHz and 6GB of RAM. This was not the computer used in the other computations of this work due to lack of computing power.

Constant	Number of monomials	Size (MB)	Time (s)
1	4	5.5	0.068
2	52	14.6	0.148
3	462	115.0	1.432
4	2644	857.1	31.528
5	10885	4846.5	755.212

3.2 Adaptions to the algorithm and computing systems

As we have seen, adaptations must be made in order to make this problem computationally affordable. We will treat each one of these during the current chapter:

- In the first place, as we already stated, we have decided to focus our efforts on systems with only 3 monomial nonlinearities, which reduces the number of parameters to take into account to 4 (if we also apply a change of variables in order that the first monomial has coefficient 1, more on this later).

- Then, we need to free the CAS from the burden of computing symbolic derivatives and series expansion and truncation. With this goal, we developed a computational system of equivalent operations that work with vectors instead of multivariate polynomials.

- Since we do not need specific functions for treating derivatives and Taylor series, because we will develop our own computing system tailored specifically for this problem, we can choose wisely a less *powerful* CAS (in the sense that it has less readily available functions and libraries), and pick one that has higher native speed of computation. In our case, we will use PARI/GP (which will be introduced later).

  We also can choose from several CAS solutions specifically for each task that we need to perform, and use the Sage interfaces to establish a communication bridge between them.

- Since we will need to make computations for tens of thousands of different 3-monomial systems, we must develop a system of parallelization of type *same program-multiple data* (SPMD) to carry out the computation of several systems at the same time and automatically feed idle processors a new system as soon as they have finished their previous task.

Observation 3.1. A further improvement that we can add to the previous algorithm is to work in $\mathbb{Z}/p\mathbb{Z}[a_1, a_2, b_1, b_2]$ with p a prime number, instead of in $\mathbb{R}[a_1, a_2, b_1, b_2]$. In this polynomial ring based on a finite field, if a polynomial can be reduced (in the Gröbner sense explained in the previous chapter) to zero then it is clear that it will be reduced also over the bigger ring. However, sometimes it can happen that a polynomial would reduce in the bigger ring and does not in the ring based on a finite field. This is not a big problem in this case, because we can consider a big enough p such that this problem is not met often, and the cases where this might occur are secondary in the sense that we care about *correctly discarding* polynomials (so a "false positive" in polynomial reduction would be an issue, but false negatives are not critical for the final results[2]).

3.2.1 On 3-monomial systems

We stated before that our systems will have three monomial nonsingularities, so equation (1.2) will in practice be:

$$\dot{z} = iz + Az^k\overline{z}^\ell + Bz^m\overline{z}^n + Cz^p\overline{z}^q, \tag{3.1}$$

where $A, B, C \in \mathbb{C}$, $3 \leqslant k + \ell \leqslant m + n \leqslant p + q$, $(k, \ell) \neq (m, n) \neq (p, q)$. Gasull et al. (2016) treat the similar problem of systems with two monomial nonlinearities with great depth.

We will use the integer values $\alpha = k - \ell - 1$, $\beta = m - n - 1$ and $\gamma = p - q - 1$. These will play an important role because when one is zero, its respective monomial appears as a *resonant monomial* in the Poincaré normal form of the complex differential equation (Gasull et al., 2016). For instance, we stated that our study dealt with Lyapunov constants living in a multivariate polynomial ring of 4 parameters over the real field, but system (3.1) has 6 parameters (A, B, C and their conjugates, or $a_1, a_2, b_1, b_2, c_1, c_2$ if we write $A = a_1 + a_2 i$, etc.). However, we could try and apply a change of variables $w = \mu z$, for $\mu = re^{i\varphi} \in \mathbb{C}$ and $r \in \mathbb{R}$, such that our system (3.1) is written as:

$$\dot{z} = iz + z^k\overline{z}^\ell + \tilde{A}z^m\overline{z}^n + \tilde{B}z^p\overline{z}^q, \tag{3.2}$$

where these $\tilde{A}, \tilde{B} \in \mathbb{C}$.

Under which conditions can we make this variable change? If we assume that $ABC \neq 0$ (so we have 3 monomials), we can write the change of variables as

$$z = re^{i\varphi}w, \quad (\text{resp. } \overline{z} = re^{-i\varphi}\overline{w}),$$

[2]This is the same problem as not being able to detect all the cases in which L_j reduces because it could be a power of it the one which does.

such that $re^{i\varphi}A = 1$. When we apply this change to the equation $\dot{z} = iz + Az^k\overline{z}^\ell$, we obtain the following expression:

$$\dot{w} = iw + Ar^{k+\ell-1}\mathrm{e}^{i\varphi(k-\ell-1)}w^k\overline{w}^\ell.$$

Thus, the term $Ar^{k+\ell-1}\mathrm{e}^{i\varphi(k-\ell-1)}$ cannot be cancelled if $k-\ell-1=0$, because in this case the equation would be $A=1$, which is false in general.

A monomial $Az^k\overline{z}^\ell$ that satisfies $k-\ell-1=0$ is called a *resonant monomial*. Therefore, if $\alpha = k-\ell-1 = 0$ (i.e., if the first of the three monomials is resonant) we will have to change the order of the monomials and put a non-resonant one as the first monomial if we want to write the equation as in (3.2).

Observation 3.2. The fewer cases where all three monomials are resonant will not be treated in this work, because the polynomial ideal where the Lyapunov constants belong has two extra parameters and we will focus our efforts in trying to characterize the centers in the 4-parameter case.

3.2.2 A vectorised computing system for polynomials

In order to fulfill our second improvement, we need to define a method of translation from vectors to polynomials. This method shall not be a general procedure, but one that suits our requirements for working in $\mathbb{C}[z,\overline{z}]$ and speeding up the implementation of the algorithm explained in the previous chapter.

Along this section we will use several PARI/GP[3] code snippets to show the implementation of some functions. The syntax is similar to other CAS and takes some elements from the C programming language (because PARI is a C library whereas GP is its interactive implementation), so the code is easy to read and understand.

Vector-polynomial-vector translation

Definition 3.3. Let $P_k(z,\overline{z}) \in \mathbb{C}[z,\overline{z}]$ be a complex polynomial in z and $\overline{z}$ of degree k. We can write it as

$$P_k(z,\overline{z}) = p_{k,0}z^k + p_{k-1,1}z^{k-1}\overline{z} + \cdots + p_{1,k-1}z\overline{z}^{k-1} + p_{0,k}\overline{z}^k,$$

where $p_{j,k-j} \in \mathbb{C}$. Then, we define the *vector corresponding to* P_k as:

$$\hat{P}_k(z,\overline{z}) = (p_{k,0}, p_{k-1,1}, \ldots, p_{1,k-1}, p_{0,k})^T. \tag{3.3}$$

[3]PARI/GP is a widely used computer algebra system designed for fast computations in number theory (see online resources at http://pari.math.u-bordeaux.fr/).

Observation 3.4. Notice that what defines the degree of a polynomial expressed as a vector is its *length*:

$$\deg(P_k) = \text{len}(\hat{P}_k) - 1.$$

Also, it is important to remark that the *order* of the coefficients in these vectors is really important, because each position in the vector corresponds to a certain monomial, depending on the lenght of the polynomial used in the representation.

A drawback of this notation is that we might need a long vector to express a single monomial. For example, the monomial $z\overline{z}^8$ will be a vector of length 10, with every coefficient equal to zero except the one before the last.

The following code snippets show two functions that translate a polynomial to a vector and a vector to a polynomial respectively:

```
/*Polynomial to vector. Arguments: P polynomial, n degree,
    vx 1st variable letter, vy 2nd variable letter*/
pol2vec(P,n,vx,vy)=
{
  my(aux);
  aux = vector(n+1);
  for(i=1,n,
    aux[i] = polcoeff(P,n-1+1,vx);
    if(i>1,
      aux[i] = polcoeff(aux[i],i,vy);
    );
  );
  return(aux);
}
```

Listing 3.1: PARI/GP function that takes in an homogeneous polynomial, its degree (although this could be obtained from the polynomial automatically), and the names of the variables, and returns a vector following Definition 3.3.

```
/*Vector to polynomial. Arguments: v vector*/
vec2pol(v)=
{
    my(pol,n);
    n=#v;
    for(i=1,n,
        pol += v[i]*z^i*w^(n-i);
    );
    return(pol)
}
```

Listing 3.2: PARI/GP function that takes a vector and returns the corresponding polynomial with variables z and $w = \overline{z}$.

Observation 3.5. The definition, and also the operations are all defined only for homogeneous polynomials. For non-homogeneous polynomial, we treat them as the sum of two or more homogeneous polynomials, so the expression will be a list of vectors of different length each, and our routines for summing, multiplying and differentiating them will have go through each (homogeneous) element of the list. PARI/GP supports dynamic lists using the `List()` constructor, which will make this implementation fairly easy.

In the following subsections we will explain how to multiply and compute derivatives of polynomials expressed as vectors. However, it is worth mentioning how to sum or substract using this method, which is trivial for homogeneous polynomials of the same degree (just sum the coefficients in a vector-like sum). For summations of polynomials of different coefficients we will use the list implementation explained in the previous observation.

Product of polynomials as vectors

Let $P_k(z,\overline{z}), Q_\ell(z,\overline{z}) \in \mathbb{C}[z,\overline{z}]$ be two homogeneous polynomials in $z, \overline{z}$ of degree k and ℓ, respectively. The product $R_{k+\ell} = P_k Q_\ell$ must be a polynomial of degree $k+\ell$, thus corresponding to a vector of length $k+\ell+1$ (and not $k+\ell+2$, which would be the addition of lengths of both vectors). Using vector notation,

$$\hat{P}_k(z,\overline{z}) = (p_{k,0}, p_{k-1,1}, \ldots, p_{1,k-1}, p_{0,k})^T,$$
$$\hat{Q}_\ell(z,\overline{z}) = (q_{\ell,0}, q_{\ell-1,1}, \ldots, q_{1,\ell-1}, q_{0,\ell})^T.$$

Remember that the length of P_k is $k+1$ and the length of Q_ℓ is $\ell+1$.

The **algorithm for multiplying** $\hat{P}_k$ and $\hat{Q}_\ell$ goes like this:

1. Define $v = (0, \ldots, 0)$ of length $k+\ell+1$.
2. For j from 0 to k do:
   (i) $aux = p_{k-j,j}\hat{Q}_\ell$ (as a scalar by vector product)
   (ii) Add j zeros to the beginning of aux (displacing the coefficients to the left)
   (iii) Add as $k-j$ zeros to the end of aux so that the vector has length $k+\ell+1$.
   (iv) $v = v + aux$.

Observation 3.6. In this algorithm no assumptions are made with respect to if $k < \ell$ or $k > \ell$. Notice that the algorithm works exactly the same for both cases.

Proposition 3.7. *The vector v defined in the previous algorithm is equivalent (via Definition 3.3) to the polynomial $S_{k+\ell} = P_k Q_\ell$.*

Proof. The algorithm tells us to perform the following operation:

$$p_{k,0}\begin{bmatrix} q_{\ell,0} \\ q_{\ell-1,1} \\ \vdots \\ q_{1,\ell-1} \\ q_{0,\ell} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + p_{k-1,1}\begin{bmatrix} 0 \\ q_{\ell,0} \\ q_{\ell-1,1} \\ \vdots \\ q_{1,\ell-1} \\ q_{0,\ell} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \cdots + p_{1,k-1}\begin{bmatrix} 0 \\ \vdots \\ 0 \\ q_{\ell,0} \\ q_{\ell-1,1} \\ \vdots \\ q_{1,\ell-1} \\ q_{0,\ell} \\ 0 \end{bmatrix} + p_{0,k}\begin{bmatrix} 0 \\ \vdots \\ 0 \\ q_{\ell,0} \\ q_{\ell-1,1} \\ \vdots \\ q_{1,\ell-1} \\ q_{0,\ell} \end{bmatrix}.$$

Take as an example $k = 3$, $\ell = 2$, and consider the following notation:

$$\begin{aligned} P_3 &= p_0 z^3 + p_1 z^2 \overline{z} + p_2 z \overline{z}^2 + p_3 \overline{z}^3, \\ Q_2 &= q_0 z^2 + q_1 z \overline{z} + q_2 \overline{z}^2. \end{aligned}$$

Now we have

$$p_0\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ 0 \\ 0 \\ 0 \end{bmatrix} + p_1\begin{bmatrix} 0 \\ q_0 \\ q_1 \\ q_2 \\ 0 \\ 0 \end{bmatrix} + p_2\begin{bmatrix} 0 \\ 0 \\ q_0 \\ q_1 \\ q_2 \\ 0 \end{bmatrix} + p_3\begin{bmatrix} 0 \\ 0 \\ 0 \\ q_0 \\ q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} p_0 q_0 \\ p_0 q_1 + p_1 q_0 \\ p_0 q_2 + p_1 q_1 + p_2 q_0 \\ p_1 q_2 + p_2 q_1 + p_3 q_0 \\ p_2 q_2 + p_3 q_1 \\ p_3 q_2 \end{bmatrix}.$$

Notice how the j-th term in this product is a sum of terms of P and Q, $p_m q_n$, such that $m + n = j$. We can define the sequences $\tilde{P}$ and $\tilde{Q}$ as the P_k and Q_ℓ extended by zeros:

$$\tilde{P} = \begin{bmatrix} p_{k,0} \\ \vdots \\ p_{0,k} \\ 0 \\ \vdots \end{bmatrix} \qquad \tilde{Q} = \begin{bmatrix} q_{\ell,0} \\ \vdots \\ q_{0,\ell} \\ 0 \\ \vdots \end{bmatrix}.$$

This assures us that the following formula works for describing the algorithm:

$$(S_{k+\ell})_j = \sum_{m=0}^{j} \tilde{p}_m \tilde{q}_{j-m}.$$

Notice that this is actually the formula for the $j-$th coefficient of the polynomial $S_{k+\ell} = P_k Q_\ell$:

$$P_k Q_\ell = \sum_{j=0}^{k+\ell} \sum_{m=0}^{j} p_m q_{j-m} z^{k+\ell-j} \overline{z}^j,$$

where the coefficients of P_k and Q_ℓ are also extended to be zero outside of the range of the polynomial.

Therefore, the algorithm for vector product yields an equivalent result to the polynomial multiplication, as we wanted to prove. □

Next, the code fragment shows this algorithm implemented in PARI/GP.

```
/* multiplies 2 homogeneous polynomials as vectors */
vpolmult(P,Q)=
{
    my(len,res,aux);
    len = #P + #Q - 1;
    res = vector(len);
    for(j=1,#P,
        aux = vector(len);
        for (i=1,#Q,
            aux[i+j-1] = P[j]*Q[i];
        );
        res += aux;
    );
    return(res)
};
```

Listing 3.3: Vectorised polynomial product in PARI/GP.

Derivative of polynomials as vectors

The last operation that we need to apply to our polynomials is to differentiate them with respect to z or $\overline{z}$. Integration is not needed for our purposes, so we did not implement it.

The derivative of a polynomial with respect to one of its variables is completely straightforward (multiply the coefficient by the exponent and substract one from the exponent). In vector notation, this is translated into a decrease of 1 in lenght (because the degree decreases in 1), removing the rightmost element for differentiation with respect to z and the leftmost element for differentiation with respect to $\overline{z}$, and multiplying every coefficient by the degree of the variable (which is the total degree minus the position,

starting from the left when differentiating by z and from the right when differentiating by $\overline{z}$).

The procedure will become clearer with this example of degree 4:

$$\begin{aligned}
P_4(z,\overline{z}) &= p_{4,0}z^4 + p_{3,1}z^3\overline{z} + p_{2,2}z^2\overline{z}^2 + p_{1,3}z\overline{z}^3 + p_{0,4}\overline{z}^4,\\
\frac{\mathrm{d}}{\mathrm{d}z}P_4(z,\overline{z}) &= 4p_{4,0}z^3 + 3p_{3,1}z^2\overline{z} + 2p_{2,2}z\overline{z}^2 + p_{1,3}\overline{z}^3,\\
\frac{\mathrm{d}}{\mathrm{d}\overline{z}}P_4(z,\overline{z}) &= p_{3,1}z^3 + 2p_{2,2}z^2\overline{z} + 3p_{1,3}z\overline{z}^2 + 4p_{0,4}\overline{z}^3,\\
\hat{P}_4 &= (p_{4,0}, p_{3,1}, p_{2,2}, p_{1,3}, p_{0,4})^T,\\
\frac{\mathrm{d}}{\mathrm{d}z}\hat{P}_4 &= (4p_{4,0}, 3p_{3,1}, 2p_{2,2}, p_{1,3})^T,\\
\frac{\mathrm{d}}{\mathrm{d}\overline{z}}\hat{P}_4 &= (p_{3,1}, 2p_{2,2}, 3p_{1,3}, 4p_{0,4})^T.
\end{aligned}$$

The simple functions for performing these operations are shown below:

```
/* differentiates a homogeneous polynomial with resp. to z
   */
vpoldz(P)=
{
    my(deg,res);
    deg = #P-1;
    res = vector(deg);
    for(i=1,deg,
        res[i] = (deg-i+1)*P[i];
       );
    return(res)
};
```

Listing 3.4: Function for differentiating by z.

```
/* differentiates a homogeneous polynomial with resp. to w
   */
vpoldw(P)=
{
    my(deg,res);
    deg = #P-1;
    res = vector(deg);
    for(i=1,deg,
        res[deg-i+1] = (deg-i+1)*P[#P-i+1];
       );
    return(res)
};
```

Listing 3.5: Function for differentiating by $\overline{z}$.

This procedure is really simple for the computer, because it only relies in following a for loop without having to search for degrees of monomials or variables, etc. The key to the speedup that this representation brings is the storage of the vectors, as the position of the coefficients encodes the degree of each monomial instead of allowing for repositioning of monomials and having to look for the information on the degree and coefficients *a posteriori*, as many CAS do.

Observation 3.8. As we have shown that the vector representation given by Definition 3.3 is equivalent to the polynomial expressions for adding, multiplying and differentiating polynomials, we will use the "no hat" notation throughout this work, even though operations are considered to be performed in the vectorised form.

3.2.3 Choice of CAS

As we showed in the previous section, our CAS of choice for performing computations is PARI/GP. Among other considerations, we took into account that the memory allocated by PARI available for performing computations is static (there is no dynamic memory allocation that allows to increase the stack on demand). This removes overhead at the possible cost of unused allocated memory (which is not a problem for systems with big amounts of available RAM memory like ours).

Another reason for choosing PARI/GP for our computations is its Sage[4] interface. As Sage builds on top of many computing packages, we can take advantage of the speed of PARI/GP when computing Lyapunov constants with our vectorised functions and the power of Singular[5] for treating these constants, which are polynomials in 4 variables for which our vectorised operations are no longer useful. Sage allows the results from one computing software to be passed to the other CAS and acts as a translator between the two.

3.2.4 SPMD implementation using PVM

As mentioned in Section 3.2 SMPD stands for *Same Program Multiple Data*, which is a parallelization paradigm that consists on executing a same piece of software many times but changing the input data for each execution.

This is the way to proceed in our case, because we need to run exactly the same computations for many different fields. For example, if we wanted to make some computations for every possible field of the type $\dot{z} = iz + R(z, \overline{z})$, with $R(z, \overline{z})$ a non-homogeneous

[4]SageMath is a free open-source mathematics software system licensed under the GPL (see http://www.sagemath.org/).

[5]Singular is a computer algebra system for polynomial computations, with special emphasis on commutative and non-commutative algebra, algebraic geometry, and singularity theory (see https://www.singular.uni-kl.de/).

complex polynomial of degree up to 3, we would need to run

$$\frac{(g(2)+g(3))!}{(g(2)+g(3)-3)!} = 504$$

executions, where $g(x) = 2(\mathcal{P}_2(x)+1)$ is the number of partitions of size 1 or 2 of $x \in \mathbb{N}$ taking into account the order of the sum. For degree 7, this result increases up to 110544 executions. This means that if every execution takes 20 minutes to complete, it would take around 1535 days (more than 4 years) of uninterrupted execution to study the full degree 7 case sequentially.

Instead, if we resort to SPMD and are able to run, for example, 50 executions at once, we would reduce the computation time from 1535 days to 30 days (which is still a long execution time, but affordable for some).

Parallel Virtual Machine

PVM is a software system that enables a heterogeneous computer cluster to be used as a concurrent resource. The functions of this system can be accessed via C or Fortran libraries, in order to initialise the system, take care of message passing and synchronization.

Through a configuration file, PVM can be told how many nodes there are in the cluster and the individual name of each one, in order to be able to execute certain functions in specific nodes to take advantage of their particular strengths (e.g. more available memory or a specific operating system).

We will use the C library for PVM, which is included using the `#include <pvm3.h>` clause. If the configuration file is called *hostfile*, we can initialize the PVM daemon from within our C program[6]:

```
1 char *pvmd_argv[1] = {"hostfile"}
2 int pvmd_argc = 1;
3 pvm_start_pvmd(pvmd_argc,pvmd_argv,1);
```

Listing 3.6: How to initialise the PVM daemon from inside a C program instead of initialising it prior to executing a program

PVM's message passing routines, although a bit outdated in design compared to MPI, are really easy to use. The sender fills up a container with all the variables that it wants to send, and the receiver gets the container and unpacks the variables from it:

[6]This is a simple example. For a full manual on how to use PVM and the daemon (*pvmd*) check out http://www.csm.ornl.gov/pvm/pvm_home.html.

```
/* Sender code */
int i=1;
long int j=100;
double r=0.2;
pvm_initsend(PvmDataRaw); // necessary for initialising the
     message
pvm_pkint(&i,1); // sending one integer located at the
    memory location of i
pvm_pklong(&j,1);
pvm_pkdouble(&r,1);
pvm_send(1,101); // send to node 1 a message with code 101

/* Receiver code */
int k;
long l;
double d;
pvm_recv(-1,101); // receive a message with code 101 from
    any sender
pvm_upkint(&k,1); // unpack an integer and store it in k
pvm_upklong(&j,1);
pvm_upkdouble(&d,1);
```

Listing 3.7: Example of message passing in PVM. The sender packs variables in a certain order that has to be respected by the receiver. The receiver might not want to unpack every variable received, however (e.g. unpack only the first two).

There are many paradigms for message passing distributed computation, but one of the simplest and yet more powerful is the *master-slave* paradigm. Here, there are two or more programs. One is the master, which is in charge of initialising the software and spawning slaves (or tasks) to be computed. The master sends data to the slaves, the slaves perform all the computational work[7], and when they finish they send the results back to the master and are *despawned* if there is no more work to do.

The *workload management* is also a central aspect of distributed computing. There are many ways that the master can distribute the work between the spawned slaves. For instance, one could decide to send $1/n$ of all workload to each one of the n slaves. This can work for completely homogeneous cases (i.e. all the slaves have identical hardware properties and all the tasks to perform have exactly the same length). However, it is common that either the slaves are spawned in different, heterogeneous machines, or the tasks could have different computing times, or both. This is why it is more common for the master to send a batch of work to the slaves (e.g. one or two jobs to each slave) and then the slaves ask for more work when they are finished with their tasks. This way the chances of one slave to finish their work really fast and becoming idle are almost nullified, and the performance of the execution increases greatly.

[7] In some cases, the master also performs the work of the slaves at the same time, although this is less frequent and adds unnecessary complexity in most cases.

PBala, software for managing SPMD executions

In order to use PVM, the computations must be carried out by a C or Fortran program, because the PVM library is not available for using from other languages such as PARI/GP or Sage. This arises the need for an intermediate software that manages the activation of pvmd, the spawning of tasks, the distribution of data, etc., all from C but compatible with any program written in Maple, Sage, PARI/GP, Python or even C itself. The idea is that once we have a middleware that can distribute our executions, we just need to write a task in a language of choice and pass the task and a data file to this program, and the program takes care of the rest. It is obvious that this is oriented towards SPMD (Single Program-Multiple Data) executions, because this is what we need for our computations.

We developed a program called **PBala** that does not only all of the above, but also takes care of the output files generated by each task, supports error reporting and has a verbose output of which task has been executed in which node. It also takes error recovery into account, generating a list of all the tasks that could not be executed (for example because they were killed by the system) in a format which is readily available for reexecuting the program using this new file instead of the original data file, for finishing the execution. What's more, it can also report memory usage data for each execution.

PBala is open source under the GNU GPLv3 license[8], the source code is available at https://github.com/oscarsaleta/PBala. It has been specifically programmed for working in the computing server of the Mathematics Department of the Autonomous University of Barcelona, although little or no modification would be needed for it to work on another cluster with PVM correctly installed and configured.

The software consists in two executables, *PBala* and *task*, of which only PBala has to be run by the user (task is executed by PBala slaves to fork processes that execute the program that the user provided). Currently, only Maple, C, Python, PARI/GP and Sage programs are supported, although the addition of new languages is really simple.

For a more in depth explanation of PBala and some code snippets, see Appendix A.

[8] See https://www.gnu.org/licenses/gpl-3.0.en.html.

Chapter 4

Recovering previous research results

4.1 Weak foci of high order and cyclicity (Liang & Torregrosa, 2016)

In their paper *Weak foci of high order and cyclicity*, Liang and Torregrosa (2016) found a family of systems with two non-homogeneous monomial nonlinearities that present the highest order of cyclicity currently known for explicit non-homogeneous systems of odd degree: $(n-1)^2$. Also remarkable is the fact that this family presents the same order of cyclicity for even and odd degrees, which is uncommon (most of the explicit systems with odd degree that have been found present a cyclicity order around half of their even counterparts (see Gasull et al., 2016, Qiu & Yang, 2009 for some examples that we will treat in this chapter).

One of the main results of the paper states the following:

Theorem 4.1 (Theorem 1.3, Liang & Torregrosa, 2016)**.** *For every integer* $3 \leqslant n \leqslant 100$*, the origin of equation*

$$\dot{z} = iz + \overline{z}^{n-1} + z^n \tag{4.1}$$

is a stable (unstable) weak focus of order $(n-1)^2$ *when* n *is even (odd).*

This theorem is proved by computing all these cases and checking that this is indeed true.

We wanted to put our algorithm (combined with the vectorised polynomial operations and the SPMD parallelization system, both developed by us) to test by recovering

their results. However, even for our PARI/GP implementation with vectorised operations, our performance is surely lower than the one of the algorithm from Liang and Torregrosa (2016), because the authors developed an ad-hoc algorithm (as opposed to our general purpose one) and also used PARI/GP for their computations. Therefore we performed the computations for only a subset of all the cases of the paper: from $n = 3$ to $n = 63$, because higher degrees proved to yield too long executions and *every degree computed was in perfect agreement with the result from the paper*.

For $n = 3$, the system

$$\dot{z} = iz + \overline{z}^2 + z^3$$

has its first non-zero Lyapunov constant of order 4, which agrees with the theorem that says that the order of this weak focus is $(n-1)^2$.

$$L_4 = 2.$$

Observation 4.2. Notice that in this case the Lyapunov constants are rational numbers, because there are no parameters in our systems. This speeds the computation compared to the case with parameters and allows us to compute systems of really high orders (compared to the case with 4 parameters, in which computing the Lyapunov constants for degree 7 is already a huge computational task).

The highest order computed was $n = 63$:

$$\dot{z} = iz + \overline{z}^{62} + z^{63}. \tag{4.2}$$

Its first non-zero Lyapunov constant is L_{3844}, and $62^2 = 3844$. This constant is a fraction with a 2369 digit numerator and a 2168 digit denominator, therefore we will not show it in this work (its floating point approximation is 1.30343×10^{201}). The computation time of this last case was 36h and 21min, taking up to 25.3 GB of RAM memory. The reason why the authors were able to compute up to such high degrees is, as we commented above, that they used a highly optimised *ad-hoc* algorithm specialised for this particular problem.

4.2 Center problem for systems with two monomial nonlinearities (Gasull et al., 2016)

This is the paper that has influenced most the present work. The authors treat the problem of characterising all the possible centers found within the following family of planar differential equations:

$$\dot{z} = iz + Az^k\overline{z}^\ell + Bz^m\overline{z}^n, \tag{4.3}$$

where $A, B \in \mathbb{C}$, $k, \ell, m, n \in \mathbb{N}$ and $(k, \ell) \neq (m, n)$. In the following chapter we attempt to get a similar result, generalising the family to 3 monomial nonlinearities, which adds some considerations for reversible centers that we will discuss later on.

Below is the theorem which presents the characterization for all the centers at the origin for the equation (4.3), giving all the possible cases depending on both the exponents k, ℓ, m, n and the coefficients A, B.

Theorem 4.3 (Theorem 1 of Gasull et al., 2016)**.** *The origin of equation (4.3) is a center when one of the following (nonexclusive) conditions hold:*

(a) $k = n = 2$ *and* $\ell = m = 0$ *(quadratic Darboux centers),*

(b) $\ell = n = 0$ *(holomorphic centers),*

(c) $A = -\overline{A}e^{i\alpha\varphi}$ *and* $B = -\overline{B}e^{i\beta\varphi}$ *for some* $\varphi \in \mathbb{R}$ *(reversible centers),*

(d) $k = m$ *and* $(\ell - n)\alpha \neq 0$ *(Hamiltonian or new Darboux centers).*

In this paper, aside from characterising the centers for the family (4.3), the authors introduce the following proposition:

Proposition 4.4 (Proposition 10 of Gasull et al., 2016)**.** *Consider the differential equations*

$$\dot{z} = iz + z^d + C_1 z^{d-1}\overline{z}, \tag{4.4}$$
$$\dot{z} = iz + z^{d-1}\overline{z} + C_2 z\overline{z}^{d-1}. \tag{4.5}$$

Then there exist values of C_1 *and* C_2 *such that the origin is a weak focus of order* $(d^2 + d - 2)/2$ *for any odd* $5 \leqslant d \leqslant 89$ *and of order* $d^2 - d$ *for any even* $\leqslant 34$.

We used the cases $d = 5$ and $d = 15$ to exemplify the case of equation (4.5) from the proposition above. For $d = 5$, the two first nonzero Lyapunov constants found are:

$$L_{10} = \frac{1}{16}(C_2 + \overline{C}_2)(3C_2\overline{C}_2 - 4),$$
$$L_{14} = \frac{5245}{2}(C_2 + \overline{C}_2).$$

Also, the following Lyapunov constants are found to completely reduce with respect to the Gröbner basis formed by these two. Thus, for $C = \frac{4}{3}\overline{C}_2^{-1}$ we have $L_{10} = 0$ and $L_{14} \neq 0$, which means that we obtain a weak focus of order $14 = (5^2 + 5 - 2)/2$, as the proposition says.

In the case $d = 15$, we have these two first significant center conditions:

$$L_{105} = \frac{54431}{12811298899230720}(4253459678612784C_2\overline{C}_2 + 206968225791341)$$
$$(C_2^2 + \overline{C}_2^2)(C_2^4 - C_2^2\overline{C}_2^2 + \overline{C}_2^4),$$
$$L_{119} = -\frac{3}{32}(C_2^2 + \overline{C}_2^2)(3725C_2^4 + 28278C_2^2\overline{C}_2^2 + 3725\overline{C}_2^4).$$

And the following Lyapunov constants are either 0 or they completely reduce in a Gröbner basis formed by these two. Clearly, if $C_2 + \overline{C}_2 = 0$ we have a reversible center (see Section 5.2). However, the other solutions for C_2 in L_{105} do not vanish L_{119}:

$$C_2 = -\frac{206968225791341}{4253459678612784}\overline{C}_2^{-1},$$
$$C_2 = \pm\frac{\overline{C}_2}{2}\sqrt{2 \pm 2i\sqrt{3}}.$$

Therefore, this system has a focus of order $119 = (15^2 + 15 - 2)/2$.

4.3 On the focus order of planar polynomial differential equations (Qiu & Yang, 2009)

In their work, Qiu and Yang (2009) showcase an interesting property of homogeneous planar polynomial differential equations, which is the fact that they usually "skip" orders of Lyapunov constants. Take the following equation as an example:

$$\dot{z} = iz + az^3\overline{z} + ibz^2\overline{z}, \tag{4.6}$$

with $a, b \in \mathbb{R}$. This system has the following first 3 nonzero Lyapunov constants:

$$L_3 = -2ab,$$
$$L_6 = 64ab^3 + 54a^3b,$$
$$L_9 = -3616ab^5 - 6504a^3b^3 - 3038a^5b,$$
$$L_{12} = 350436ab^7 + \frac{8409487}{9}a^3b^5 + \frac{7891651}{9}a^5b^3 + 289152a^7b.$$

As can be seen from these computations, focal values "skip" three orders each time in which they are exactly zero.

This property has been used by the authors of the paper to find a family of polynomial systems that make high order weak foci, because they were able to obtain a family which has every Lyapunov constant equal to zero up to a certain order, and then solving for a single parameter in this first nonzero Lyapunov constant they are able go further for several orders.

The main result of the work of Qiu and Yang is the following proposition.

Proposition 4.5 (Proposition 1 of Qiu & Yang, 2019)**.** *For $n \in \{4, 6, 8, 10, 12, 14, 16, 18\}$, the system*

$$\dot{z} = iz - \frac{n}{n-2} z^n + z\overline{z}^{n-1} + i\tau_n \overline{z}^n, \tag{4.7}$$

where τ_n is a fixed real number, has a focus at the origin with the focus order of n^2+n-2.

For $n \in \{3, 5, 7, 9, 11, 13, 15, 17, 19\}$, the following system

$$\dot{z} = iz + \frac{n}{n-2} z^n + z\overline{z}^{n-1} + (1 + i\tau)\overline{z}^n, \tag{4.8}$$

where τ is any transcendental number, has a focus at the origin with the focus order of $(n^2 + n - 2)/2$.

Observation 4.6. Notice that this proposition gives a family of odd degree systems with a focus of lower order than the 2 monomial family from Gasull et al. (2016) but higher order for even polynomial planar systems.

For some reason, probably related to available computing power and time of execution, the authors only check up to degrees 19 and 20. We were able to pick up from there and check up to $n = 40$ for the even systems and to $n = 59$ for odd ones.

The code snippet in Listing 4.1 shows our algorithm for performing the even cases. The file `polops.gp` is our GP library where we have implemented all the functions that compute Lyapunov constants using the vectorised operations. This program is prepared to be executed through PBala (notice that it takes the value of n from a vector, `taskArgs`, which is passed to the script by the PBala software, which has read it from an input file).

For $n = 20$, the first nonzero Lyapunov constant is

$$L_{399} = A\tau^{21} - B\tau^{19},$$

with $A, B \in \mathbb{Q}$ fractions of more than 220 digits in both numerator and denominator. We can solve for τ and then we obtain the following new first significant center condition:

$$L_{418} \approx 16303090276776173.952.$$

Notice that $418 = 20^2 + 20 - 2$, as expected. The same can be said for all cases from $n = 22$ to $n = 40$, being the latter the last one computed, although the computation of all cases from $n = 20$ to $n = 40$ took only around 2.5h making use of PBala and the vectorised operations developed. To round off, we tease the results for $n = 40$:

$$\begin{aligned} L_{1599} &= A\tau^{41} - B\tau^{39}, \\ \tau &\approx 0.275, \\ L_{1638} &\approx 41432785440135659008514752319048447.609. \end{aligned}$$

It is worth noticing that no computation was made in floating point arithmetics, but this is our only possible way of fitting such big rational numbers in this document. In fact, $A, B \in \mathbb{Q}$ for $n = 40$ are rational numbers with numerators and denominators of more than 800 digits each.

The odd case is simpler, because it is not needed to solve any equations, neither to go further in the order of the focus, because the main point in this family of odd systems is that the first nonzero Lyapunov constant *must not* be solvable for the system to have a focus and not a center (see Listing 4.2 at the end of this section).

For $n = 21$, the focus is of order 230:

$$L_{230} = \lambda_0\tau^{12} - \lambda_1\tau^{10} + \lambda_2\tau^8 - \lambda_3\tau^6 - \lambda_4\tau^4 + \lambda_5\tau^2 - \lambda_6,$$

where $\lambda_k \in \mathbb{Q}$ for $k = 0, \ldots, 6$. Clearly, if τ is transcendental, it cannot be a root of L_{230}, so the focus is of order $230 = (21^2 + 21 - 2)/2$.

The case $n = 59$ also agrees with Proposition 4.5, because the order of the first significant Lyapunov constant is $1769 = (59^2 + 59 - 2)/2$:

$$L_{1769} = \sum_{k=0}^{1} 5\lambda_k\tau^{30-2k}.$$

Clearly, for any τ transcendental, $L_{1769} \neq 0$. The whole odd case (from $n = 21$ to $n = 59$) took 1 hour, 42 minutes and 6 seconds to complete, because all cases were computed at once and the longest execution was $n = 59$, which took the aforementioned time.

Under the light of these new computations of higher degree systems, the conjecture of Qiu and Yang (2009) stands strong.

```
\r polops.gp
/* Get argument from PBala */
n=taskArgs[1];
/* Define the vectorised field of degree n */
evenfield(m)=
{
    my(v);
    v=vector(m+1);
    v[1]=-m/(m-2);
    v[m]=1;
    v[m+1]=I*a;
    return(List([v]));
}

{
    R=evenfield(n);
    /* compute first nonzero lyapunov constant */
    L=nextlyapunov(R);
    print("QiuYang n = ",n," (expected first nonzero constant: ",(n*n+n
        -2),")");
    print(" - First nonzero Lyapunov const: \n\tL[",L[1][1][1],"] = ",L
        [1][1][2]);
    /* factorise the Lyapunov constant */
    facts=factor(L[1][1][2]);
    /* tau is the solution of the linear factor */
    tau=-polcoeff(facts[2,1],0,a)/polcoeff(facts[2,1],2,a);

    print("\nWith a^2 = ",tau);
    Li=0;
    for(i=1,n^2+2*n-1,
        /* compute next lyapunov constant from the previous one */
        L=nextlyapunov(R,L[2],L[1]);
        /* check if it vanishes under tau */
        if(substpol(L[1][i][2],a^2,tau)!=0,
            Li=L[1][i];
            break;
        );
    );
    if(Li!=0,
        print(" - First nonzero Lyapunov const:\n\t L[",Li[1],"] = ",
            substpol(Li[2],a^2,tau));
        print("\tNumerical approximation = ",substpol(Li[2],a,sqrt(tau)))
            ;
    );
    if(Li==0,
        print(" - Suspected center, further analysis required\n");
    );
}
\q
```

Listing 4.1: PARI/GP program that computes the first significant Lyapunov constant of a *Qiu-Yang* even differential equation (Qiu & Yang, 2009), and solves it to find the condition for which the order of the focus can be increased.

```
\r polops.gp
/* Get argument from PBala */
n=taskArgs[1];
/* Define the vectorised field of degree n */
oddfield(n)=
{
    local(v);
    v=vector(n+1);
    v[1]=n/(n-2);
    v[n]=1;
    v[n+1]=1+I*a;
    return(List([v]));
}
/* compute first nonzero lyapunov constant */
L=firstlyapunov(oddfield(n));
print("QiuYang n = ",n," (expected first nonzero constant: ",(n*n+n-2)/2,
    ")");
print(" - First nonzero Lyapunov const:\n\tL[",L[1][1],"] = ",L[1][2]);
/* Show factorisation */
fctr=factor(L[1][2]);
print("\tFactorisation: ",fctr);
\q
```

Listing 4.2: PARI/GP script that computes the first nonzero Lyapunov constant for the odd cases in Proposition 4.5. This program does not need to solve any equation, since the key point is that the systems have a high order focus *if* the first nonzero Lyapunov constant *cannot* be solved, which only happens for τ transcendental.

Chapter 5

Study of low degree systems with 3 monomial nonlinearities

5.1 Center conditions for 3 monomial nonlinearities

We consider the following 3 monomial system

$$\dot{z} = iz + Az^k\overline{z}^\ell + Bz^m\overline{z}^n + Cz^p\overline{z}^q, \tag{5.1}$$

where $A, B, C \in \mathbb{C}$ and $A, B, C \neq 0$ (in order to have 3 monomial nonlinearities). We will also use the more general expression

$$\dot{z} = iz + F(z, \overline{z}) = iz + \sum_{k+\ell \geqslant 2} f_{k,l} z^k \overline{z}^\ell. \tag{5.2}$$

We will also use the notation $\alpha = k - \ell - 1$, $\beta = m - n - 1$ and $\gamma = p - q - 1$, for these numbers play a special role in the study of the system. Notice that if $\alpha = 0$, $\beta = 0$ or $\gamma = 0$, the respective associated monomial becomes *resonant* (see section 3.2.1 for a simple explanation of some difficulties resonant monomials can introduce).

The following theorem is an extension of Theorem 4.3 (Theorem 1 in Gasull et al., 2016) given by Gasull, Giné and Torregrosa. These authors show a possibly exhaustive classification of the centers for the 2 monomial system (4.3). We try to extend their results for our 3 monomial system, which will introduce several difficulties that will be discussed in this chapter.

Theorem 5.1. *The origin of equation (5.1) is a center when one of the following conditions hold:*

(a) $k = q = 2$, $m = n = 1$, $\ell = p = 0$ *and:*

(i) $B = 0$ *(Darboux centers)*

(ii) $A = -\overline{B}/2$ *(Hamiltonian centers),*

(iii) $\mathrm{Im}(AB) = \mathrm{Im}(\overline{B}^3 C) = \mathrm{Im}(A^3 C) = 0$ *(reversible centers),*

(iv) $A = 2\overline{B}$ *and* $|C| = |B|$ *(Darboux centers),*

(b) $\ell = n = q = 0$ *(holomorphic centers),*

(c) $A = -\overline{A}e^{i\alpha\varphi}$, $B = -\overline{B}e^{i\alpha\varphi}$ *and* $C = -\overline{C}e^{i\alpha\varphi}$ *for some* $\varphi \in \mathbb{R}$ *(reversible centers)*

(d) $k = m = p$, $(\ell - n)(\ell - q)(n - q) \neq 0$ *and* $\mathrm{Re}(A) = 0$, $\mathrm{Re}(B) = 0$, $\mathrm{Re}(C) = 0$ *if* $\alpha = 0$, $\beta = 0$ *or* $\gamma = 0$, *respectively (Hamiltonian or new Darboux centers).*

We stated previously that we would only consider cases in which the three monomials were present, so case (a.i) will not be taken into account. However, we include it in the list for completeness, so the 2-monomial case can be retrievable from this theorem.

Notice that these conditions are nonexclusive. For example, even though condition (c) characterizes all reversible centers for equation (5.1), case (a.ii). are also reversible centers. Moreover, there may be centers which are reversible and Hamiltonian or Darbouxian at the same time, fulfilling both conditions.

Observation 5.2. Theorem 5.1 is derived directly from the work of Gasull et al. (2016), and as such its classification is complete for the cases of two monomial nonlinearities. However, in the 3 monomial case, different, unclassified centers will appear that will have to be studied in detail before considering this characterization problem complete.

Although a proof for Theorem 1 is derived directly from Gasull et al. (2016) and Zoladek (1994), we will include here the main results that justify this theorem.

Definition 5.3. The origin of (5.2) is said to be a *persistent center* if it is center for $\dot{z} = iz + \lambda F(z, \overline{z})$ for all $\lambda \in \mathbb{C}$. Similarly, the origin is a *weakly persistent center* if it is a center for $\dot{z} = iz + \mu F(z, \overline{z})$, for all $\mu \in \mathbb{R}$ (Cima, Gasull, & Medrado, 2009).

Theorem 5.4 (Gasull, Giné and Torregrosa, 2016)**.** *Consider the differential equation* (5.2) *where* $F(z, \overline{z}) = z^k f(\overline{z}) = z^k \sum_{\ell \geqslant 0} f_\ell \overline{z}^\ell$ *for* $k \geqslant 0$, *and* F *starts at the origin at least with second degree terms. Then the origin is a center if and only if either* $k \in \{0, 1\}$ *or* $k > 1$ *and* $\mathrm{Re}(f_{k-1}) = 0$. *Indeed, in all cases the origin is a weakly persistent center, Hamiltonian when* $k = 0$ *and of Darboux type when* $k \geqslant 1$. *Moreover, it is a persistent center when* $k \in \{0, 1\}$ *or* $k > 1$ *and* $f_{k-1} = 0$.

Proof. First of all, notice that, since the origin is a monodromic critical point[1], to prove

[1] A critical point p of a system is called *monodromic* if there is a neighbourhood and a smooth arc originating at the singular point which is transversal to the field at every point except the origin, with the property that the direction field on the neighbourhood with the arc removed is diffeomorphic to the standard field (Aranson, Bronshtein, Grines, & Ilyashenko, 1996).

that it is a center it suffices to construct a smooth first integral of (5.2) that is continuous at the origin.

Also notice that for $k > 0$, the function $U(z, \overline{z}) = z\overline{z} = 0$ is an invariant algebraic curve for (5.2):

$$\dot{U}(z, \overline{z}) = \dot{z}\overline{z} + z\dot{\overline{z}} = 2\mathrm{Re}\left(z^{k-1} f(\overline{z})\right) U(z, \overline{z}).$$

Using this function it is easy to see that $U^{-k}(z, \overline{z}) = (z\overline{z})^{-k}$ is an integrating factor of the differential equation. First, recall that if X is the vector field associated to a differential equation $\dot{z} = G(z, \overline{z})$ then

$$\mathrm{div}(X) = 2\mathrm{Re}\left(\frac{\partial}{\partial z}(G(z, \overline{z}))\right).$$

Thus, for the equation $\dot{z} = (iz + z^k f(\overline{z}))(z\overline{z})^{-k}$:

$$\mathrm{div}(X) = 2\mathrm{Re}\left(\frac{\partial}{\partial z}\left((iz + z^k f(\overline{z}))(z\overline{z})^{-k}\right)\right) = 2(1-k)\mathrm{Re}(i(z\overline{z})^{-k}) = 0,$$

so $(z\overline{z})^{-k}$ is an integrating factor of the equation (5.2) as we wanted to show. This means that we can compute a first integral for the equation. Let $H_1(z, \overline{z})$ be a function such that

$$\frac{\partial H_1(z, \overline{z})}{\partial \overline{z}} = (iz + z^k f(\overline{z}))(z\overline{z})^{-k},$$

then

$$H_1(z, \overline{z}) = \begin{cases} i\frac{(z\overline{z})^{1-k}}{1-k} + g(\overline{z}) - \overline{(g(\overline{z}))} & \text{if } k \neq 1, \\ \log(z\overline{z})i + g(\overline{z}) - \overline{(g(\overline{z}))} & \text{if } k = 1, \end{cases}$$

where $g'(u) = u^{-k} f(u)$. Therefore, the real function

$$H_2(z, \overline{z}) = \begin{cases} \frac{(z\overline{z})^{1-k}}{2(1-k)} + \mathrm{Im}(g(\overline{z})) & \text{if } k \neq 1, \\ \frac{\log(z\overline{z})}{2} + \mathrm{Im}(g(\overline{z})) & \text{if } k = 1, \end{cases}$$

is a candidate to be a first integral of (5.2) at $\mathbb{C}\backslash\{0\}$.

First consider the case $k = 0$. The function

$$H(z, \overline{z}) = H_2(z, \overline{z}) = \frac{z\overline{z}}{2} + \mathrm{Im}(g(\overline{z})),$$

where $g(u) = \int_0^u f(s)\mathrm{d}s$, is a smooth first integral at the origin, so (5.2) under these conditions has a center at the origin.

When $k = 1$, consider the first integral

$$H(z,\overline{z}) = \mathrm{e}^{2H_2(z,\overline{z})} = z\overline{z}\mathrm{e}^{2\mathrm{Im}(g(\overline{z}))},$$

where $g(u) = \int_0^u f(s)/s \, \mathrm{d}s$. As $f(0) = 0$, $H(z,\overline{z})$ is smooth at the origin. Hence, for $k = 1$ the origin is also a center.

Finally, for $k > 1$ we define

$$H(z,\overline{z}) = \frac{1}{H_2(z,\overline{z})} = \frac{(z\overline{z})^{k-1}}{\frac{1}{2(1-k)} + (z\overline{z})^{k-1}\mathrm{Im}(g(\overline{z}))},$$

where

$$g(u) = \int_{u_0}^{u} \frac{f(s)}{s^k} \, \mathrm{d}s = \int_{u_0}^{u} \sum_{\ell \geqslant 0} f_\ell s^{\ell-k} \mathrm{d}s = \sum_{\ell \geqslant 0, \ell \neq k-1} \frac{f_\ell}{\ell - k + 1} u^{\ell-k+1} + f_{k-1} \log u + g_0,$$

for some $g_0 \in \mathbb{C}$. This function must not be multivaluated in $\mathbb{C}$, so we need that $\mathrm{Im}(f_{k-1} \log \overline{z})$ is not multivaluated, so we need $\mathrm{Re}(f_{k-1}) = 0$. Under these conditions, $H(z,\overline{z})$ is well defined in $\mathbb{C}$ and moreover

$$\lim_{z \to 0} (z\overline{z})^{k-1}\mathrm{Im}(g(\overline{z})) = 0,$$

so $H(z,\overline{z})$ is also continuous at the origin. Therefore, for $k > 1$ and $\mathrm{Re}(f_{k-1}) = 0$, the origin is a center. What is more, the same expression of H implies that if $\mathrm{Re}(f_{k-1}) \neq 0$, then the origin is not a center. □

Observation 5.5. The fact that $\mathrm{Re}(f_{k-1}) = 0$ means that if any monomial is resonant, then its coefficient must be purely imaginary.

The following proposition covers the reversible and holomorphic cases of Theorem 5.1. For a proof see Cima, Gasull, Mañosa, and Mañosas (1997), Gasull et al. (2016).

Proposition 5.6. *Equation (5.2) has a center at the origin when one of the following conditions hold:*

(a) There exists $\varphi \in \mathbb{R}$ such that $f_{k,\ell} = -\overline{f}_{k,\ell}\mathrm{e}^{i(k-\ell-1)\varphi}$ for all k, ℓ (reversible center).

(b) $F(z,\overline{z}) \equiv F(z)$ (holomorphic center).

Proof of Theorem 5.1. Case (a) is well-known for quadratic systems (see Zoladek (1994)). Cases (b) and (c) are derived directly from Proposition 5.6, and case (d) is a consequence of Theorem 5.4. □

Observation 5.7. Theorem 5.4 covers a set of sufficient conditions for a system to present a center. However, we have not proved that this list is exhaustive, i.e. we do not know if there are other kind of centers that are not covered by this theorem.

In order to check the exhaustivity of Theorem 5.1, we need to perform as Gasull et al. (2016), and sweep a wide range of polynomial systems to find conditions in which the theorem holds.

5.2 Reversible centers

We use the same notion of reversibility than Gasull et al. (2016), which is the sense introduced by Poincaré: we say a center is reversible when it has a line of symmetry. Notice that if a system has a line of symmetry along the origin, the singular point at the origin cannot be a weak focus, because this would break the symmetry. Therefore, finding a symmetry assures the presence of a center.

As stated in Theorem 5.1, reversible centers for 3 monomial systems such as (5.1) fulfill the following simple condition:

$$A = -\overline{A}\mathrm{e}^{i\alpha\varphi}, \tag{5.3}$$
$$B = -\overline{B}\mathrm{e}^{i\beta\varphi}, \tag{5.4}$$
$$C = -\overline{C}\mathrm{e}^{i\gamma\varphi}. \tag{5.5}$$

Observation 5.8. This condition must be held for the *same* value of φ in the three expressions.

5.2.1 The two monomial case

Let us review the way in which Gasull et al. solve this problem for the case of two monomials. First, they assume that the monomial $Az^k\overline{z}^\ell$ is not resonant, so they can apply a change of variables to make $A = 1$. Therefore, the conditions become

$$1 = -\mathrm{e}^{i\alpha\varphi},$$
$$B = -\overline{B}\mathrm{e}^{i\beta\varphi}.$$

Notice how the first condition asks for $\mathrm{e}^{i\varphi}$ to be an α-root of the unity. Knowing that, we can rise the second condition to the power of α:

$$B^\alpha = (-\overline{B}\mathrm{e}^{i\beta\varphi})^\alpha = (-\overline{B})^\alpha \mathrm{e}^{i\alpha\beta\varphi} = (-1)^{\alpha+\beta}\overline{B}^\alpha, \tag{5.6}$$

where we used the first equation to simplify the exponential and get rid of φ in the expression.

Equation (5.6) gives a condition for a reversible center in system (4.3) that depends only on B, α and β. In this case, this condition is really easy to identify in the focal values, because it will always be of the form $B^\alpha \pm \overline{B}^\alpha$. If any of the nonzero Lyapunov constants contains this factor, then we have a reversible center (i.e., if we consider that factor to vanish, so must do all other Lyapunov constants, because the origin is a center). This also means that if the origin *can* be a reversible center, then (5.6) must appear in the first nonzero Lyapunov constant, because all other constants must either be already zero or vanish under the reversibility condition.

5.2.2 Wrong conditions for the three monomial case

However, the case with three monomials has some issues that do not appear in the two monomial case. If we did the same procedure as above[2], we would obtain these two conditions

$$B^\alpha = (-1)^{\alpha+\beta}\overline{B}^\alpha, \tag{5.7}$$
$$C^\alpha = (-1)^{\alpha+\gamma}\overline{C}^\alpha. \tag{5.8}$$

But these conditions are not exactly equivalent to the full set from Theorem 5.1. Let us see this with an example.

Consider the following non homogeneous system of degree 5

$$\dot{z} = iz + \overline{z}^4 + Az\overline{z} + Bz\overline{z}^4, \tag{5.9}$$

with $A, B \in \mathbb{C}$. In this case, $\alpha = -5$, $\beta = -1$ and $\gamma = -4$. Thus, the numerators of (5.7) and (5.8) are

$$A^5 - \overline{A}^5 = 0,$$
$$B^5 + \overline{B}^5 = 0.$$

Its first significant center condition is

$$L_4 = -\frac{i}{15}(32A^5 - 32\overline{A}^5 + 15iAB + 15i\overline{AB}). \tag{5.10}$$

If we consider, for example, the first 19 Lyapunov constants for this system, and we solve them for A, $\overline{A}$, B and $\overline{B}$, we obtain the following sets of solutions:

$$\{A = \overline{A},\ B = -\overline{B}\},$$
$$\{A = \mu\overline{A},\ B = (\mu^3 + \mu^2 + \mu + 1)\overline{B}\},$$

[2]Here we also suppose that $A = 1$.

where μ is a root of $X^4+X^3+X^2+X+1$. Clearly, the first set of solutions corresponds to the reversibility conditions $A^5-\overline{A}^5=0$ and $B^5+\overline{B}^5=0$.

If we solve the two reversibility conditions we will obtain the following sets:

$$\{A=\overline{A},\ B=-\overline{B}\},$$
$$\{A=\overline{A},\ B=\nu\overline{B}\},$$
$$\{A=\mu\overline{A},\ B=-\overline{B}\},$$
$$\{A=\mu\overline{A},\ B=\nu\overline{B}\},$$

where ν is a root of $X^4-X^3+X^2-X+1$.

The second set of solutions for the reversibility conditions is not included in neither the first or the second set of center conditions. Also, the 3rd set of reversibility solutions does not belong to neither set of center conditions because there is no μ root of $X^4+X^3+X^2+X+1$ such that $\mu^3+\mu^2+\mu+1=-1$. The same reason implies that the 4th set does not belong to $\{A=\mu\overline{A},\ B=(\mu^3+\mu^2+\mu+1)\overline{B}\}$, because there is no μ such that $\mu^3+\mu^2+\mu+1=\nu$.

Therefore, there are solutions in (5.8) that do not provide reversibility conditions.

Observation 5.9. The problem lies in the fact that equations (5.7) and (5.8) do not impose that the φ that satisfies each reversibility condition for equations (5.3), (5.4) and (5.5) is the *same* φ.

5.2.3 Correct set of conditions for the three monomial case

In order to find a set of expressions that ensures a common solution for (5.3), (5.4) and (5.5), we need to make the following definition. Let $x=\mathrm{e}^{i\varphi}$. Then, the reversibility conditions can be written as

$$1-x^{\alpha}=0, \tag{5.11}$$
$$A-\overline{A}x^{\beta}=0, \tag{5.12}$$
$$B-\overline{B}x^{\gamma}=0. \tag{5.13}$$

Solving for $x, A, \overline{A}, B, \overline{B}$ will provide us with the complete reversibility conditions for three monomial nonlinearities.

Using the previous example, the numerators of these conditions are

$$x^5+1=0,$$
$$Ax+\overline{A}=0,$$
$$Bx^4+\overline{B}=0.$$

The solutions for this system of equations are:

$$\{A = \overline{A},\ B = -\overline{B},\ x = -1\},$$
$$\{A = (\nu^3 - \nu^2 + \nu - 1)\overline{A},\ B = \nu\overline{B},\ x = \nu\},$$

where ν is a root of $X^4 - X^3 + X^2 - X + 1$. There are obvious similarities between these solutions and the solutions of the focal values.

Clearly, the first solution set is the same for both systems[3]. The second set must be carefully inspected in order to see if the polynomials are the same. Indeed, take both polynomials

$$P_1(X) = X^4 + X^3 + X^2 + X + 1,$$
$$P_2(X) = X^4 - X^3 + X^2 - X + 1.$$

It is obvious that $P_1(-X) = P_2(X)$. Then, the roots must satisfy $\mu = -\nu$, so it is trivial to see that the second solution set for reversibility conditions is exactly the same as the set for the Lyapunov constants.

Observation 5.10. Notice that the first equation is $x^5+1 = (x+1)(x^4-x^3+x^2-x+1)$. In general we will obtain as many solution sets as factors in the factorisation of the first condition, because this first equation is independent from the other two.

Given that there are no other ways of vanishing the set of Lyapunov constants $\{L_4, \ldots, L_{19}\}$, we conclude that the poynomial differential equation (5.9) has a reversible center at the origin.

5.3 Considerations about an algorithmic search for centers

The question that arises from Theorem 5.1 is, as stated in previous sections, find out if the list of possible centers is exhaustive or there are other, unknown, kinds of centers in the family of differential equations (5.1).

Given a differential equation with a 3 monomial nonlinear field, we can use the tools developed by us during this work in order to detect whether it belongs to one of the simpler classes of centers given by Theorem 5.1 (a), (b) or (d). If it does not, we will have to start the computation of Lyapunov constants for this complex field, until we get a complete set of conditions for $A, \overline{A}, B$ and $\overline{B}$ for which the origin is a center (the solution of a system of equations formed by the Lyapunov constants). Given these sets of solutions, we then could compare them to the sets spawned by the reversibility

[3]We disregard the value of x in the solutions for reversibility conditions, because it is of no interest for us. We simply use x to ensure that the reversibility conditions are complete.

conditions, to see if they all match or if there is any solution for the Lyapunov constants that does not imply reversibility. In the latter case, this would mean that the list given by Theorem 5.1 is not exhaustive because there are new kinds of centers not acknowledged by the theorem.

One problem for this approach is that, unlike in Gasull et al. (2016), we cannot rely on identifying a "visual" condition stored in the first significant Lyapunov quantity, because the shape of the polynomials that result in reversibility conditions is not as predictable as in (5.6). Therefore, we can only make decisions based on solution sets, for which we requiere to compute *enough* Lyapunov constants. How many constants are required to get a complete solution set is not known *a priori*, which introduces a possible computational issue that can only be mitigated by setting a genereous threshold in the number of Lyapunov constants computed for each case.

Moreover, after having computed enough Lyapunov constants, there is still the issue of how to programmatically identify that the solution sets are equal or different for Lyapunov constants and reversibility conditions. For obvious reasons, the possibility of manually checking every case must be discarded, so a method must be developed that decides whether the solutions are equivalent or not.

The development of this method is not the aim of this Masters Thesis, but will be a problem tackled in future research based on the results and tools developed in this work.

5.4 Preliminary results for systems of degree 3

As an exemplification of the work that is pending to do to close the 3 monomial problem, we inspected all the degree 3 possible systems with three monomial nonlinearities in which the first monomial is non resonant (i.e., we can assume the first coefficient to be 1) in order to check Theorem 5.1. The total number of systems analysed is 174.

Consider the following system:

$$\dot{z} = iz + \overline{z}^2 + A\overline{z}^3 + Bz\overline{z} \tag{5.14}$$

The first significant Lyapunov constant for this system is

$$L_2 = \frac{1}{3}(B\overline{A} + A\overline{B}) + \frac{2i}{3}(B^3 - \overline{B}^3).$$

The following constant is

$$L_3 = -\frac{i}{576}\Big(-38i\overline{A}B^3 - 38iA\overline{B}^3 + 38iAB^3 - 108iA\overline{B}^4 + 38i\overline{AB}^3 - 108i\overline{A}B^4 + 108iAB^4 + 108i\overline{AB}^4 + 792iAB^2 - 372iA\overline{B} + 792i\overline{AB}^2 - 372iB\overline{A} + 228iAB\overline{B}^2 + 228i\overline{A}B^2\overline{B} - 540iAB^3\overline{B} - 108iAB\overline{B}^3 - 108i\overline{A}B^3\overline{B} - 540i\overline{A}B\overline{B}^3 + 19A^2\overline{B} - 19\overline{A}^2B - 270A^2\overline{B}^2 + 270\overline{A}^2B^2 + 19A\overline{A}B - 19A\overline{AB} + 54A\overline{A}B^2 - 54A\overline{AB}^2 + 54A^2B\overline{B} - 54\overline{A}^2B\overline{B}\Big),$$

and the rest are far too big to be included in this document. The Lyapunov constants that do not completely reduce in this field are L_2, L_3, L_4, L_5 and L_6. Solving this system for A, B we obtain:

$$\{A = -\overline{A},\ B = \overline{B}\},$$
$$\{A = -\overline{A}\mu_1,\ B = \overline{B}\mu_1\},$$

where μ_1 is a root of $P_1(X) = X^2 + X + 1$.

On the other hand, the solution for the reversible center conditions (5.11), (5.12), and (5.13) is:

$$\{A = -\overline{A},\ B = \overline{B},\ x = -1\},$$
$$\{A = -\overline{A}(\mu_2 - 1),\ B = \overline{B}(\mu_2 - 1),\ x = \mu_2\},$$

where μ_2 is a root of $P_2(X) = X^2 - X + 1$. In this case, notice how $P_1(X - 1) = P_2(X)$. Hence, both solution sets are equivalent. Therefore, this system has a reversible center at the origin.

5.4.1 Some notions on Darboux integrability

In order to provide some additional insight in the systems that will appear below, we need to introduce some definitions and a main result in the field of Darboux integrability of complex polynomial systems of differential equations. These are extracted from Dumortier et al. (2006).

We already explained in Section 1.4 the definition of an integrable system. The existence of a first integral is not easy to prove in general, and the concept of *invariants* provide a weaker but also useful information. Recall the definition of the associated vector field Z of a complex polynomial system (1.5).

Definition 5.11. Let $U \subset \mathbb{C}^2$ be an open set. We say that an analytic function $H(z, \overline{z}, t) : U \times \mathbb{C} \to \mathbb{C}$ is an *invariant* of the polynomial vector field Z on U if $H(z, \overline{z}, t) = h$ for some constant h for all values of t for which the solution $(z(t), \overline{z}(t))$ is defined and contained in U.

Notice that a time-independent invariant is a first integral.

Definition 5.12. Let $U \subset \mathbb{C}^2$ be an open subset and $F : U \to \mathbb{C}$ be an analytic function which is not identically zero on U. The function F is an *integrating factor* of the complex polynomial system (1.2) on U if one of the three following equivalent conditions holds on U:

$$\frac{\partial F\dot{z}}{\partial z} = -\frac{\partial F\dot{\overline{z}}}{\partial \overline{z}}, \quad \operatorname{div}(F\dot{z}, F\dot{\overline{z}}) = 0, \quad ZF = -F\operatorname{div}(\dot{z}, \dot{\overline{z}}). \tag{5.15}$$

If a system has an integrating factor, one can find a first integral of the form

$$H(z, \overline{z}) = \int F(z, \overline{z})\dot{z}\mathrm{d}\overline{z} + h(z) \tag{5.16}$$

for h such that $\partial H/\partial z = -F\dot{\overline{z}}$.

Definition 5.13. Let $f \in \mathbb{C}[z, \overline{z}]$, f not identically zero. The algebraic curve $f(z, \overline{z}) = 0$ is an *invariant algebraic curve* of (1.2) if

$$Zf = \dot{z}\frac{\partial f}{\partial z} + \dot{\overline{z}}\frac{\partial f}{\partial \overline{z}} = Kf \tag{5.17}$$

for some polynomial $K \in \mathbb{C}[z, \overline{z}]$, which is called *cofactor* and has degree at most $m - 1$. If f is irreducible in the ring $\mathbb{C}[z, \overline{z}]$, then $f = 0$ is an *irreducible invariant algebraic curve* of the system.

Definition 5.14. Let $h, g \in \mathbb{C}[z, \overline{z}]$, and assume h and g are relatively prime in the ring $\mathbb{C}[z, \overline{z}]$ or that $h \equiv 1$. Then, the function $\exp(h/g)$ is called *exponential factor* of (1.2) if for some *cofactor* $K \in \mathbb{C}[z, \overline{z}]$ of degree at most $m - 1$ it satisfies equation (5.17).

If $S(z, \overline{z}) = \sum_{j+k=0}^{m-1} a_{jk} z^j \overline{z}^k \in \mathbb{C}_{m-1}[z, \overline{z}]$ (i.e. the subring of $\mathbb{C}[z, \overline{z}]$ of polynomials of degree at most $m - 1$), it has $m(m + 1)/2$ coefficients. We identify the linear vector space $\mathbb{C}_{m-1}[z, \overline{z}]$ with $\mathbb{C}^{m(m+1)/2}$ through the isomophism

$$S \to (a_{00}, a_{10}, a_{01}, \ldots, a_{m-1,0}, a_{m-2,1}, \ldots, a_{0,m-1}).$$

Observation 5.15. The isomorphism $\mathbb{C}_{m-1}[z, \overline{z}] \to \mathbb{C}^{m(m+1)/2}$ is the same that we have used in Section 3.2.2 to develop the vectorised polynomial operations for our algorithm of computation of the Lyapunov constants.

Definition 5.16. We say that r points $(z_\ell, \overline{z}_\ell) \in \mathbb{C}^2$ for $\ell = 1, \ldots, r$ are *independent* with respect to $\mathbb{C}_{m-1}[z, \overline{z}]$ if the intersection of the r hyperplanes

$$\sum_{j+k=0}^{m-1} z_\ell^j \overline{z}_\ell^k a_{jk} = 0, \quad \ell = 1, \ldots, r,$$

in $\mathbb{C}^{m(m+1)/2}$ is a linear subspace of dimension $[m(m + 1)/2] - r$.

Finally, we state the main result of the Darboux Theory of integrability for complex polynomial systems:

Theorem 5.17. *Suppose that a $\mathbb{C}$-polynomial system (1.2) of degree m admits p irreducible invariant algebraic $f_j = 0$ with cofactors K_j for $j = 1, \dots, p$, q exponential factors $\exp(g_k/h_k)$ with cofactors L_k for $k = 1, \dots, q$ and r independent singular points $(z_\ell, \overline{z}_\ell) \in \mathbb{C}^2$ such that $f_j(z_\ell, \overline{z}_\ell) \neq 0$ for $j = 1, \dots, p$ and for $\ell = 1, \dots, r$.*

(i) There exist $\lambda_j, \mu_k \in \mathbb{C}$ not all zero such that $\sum_{j=1}^{p} \lambda_j K_j + \sum_{k=1}^{q} \mu_k L_k = 0$, if and only if the (multivaluated) function

$$f_1^{\lambda_1} \cdots f_p^{\lambda_p} \left(\exp\left(\frac{g_1}{h_1}\right)\right)^{\mu_1} \cdots \left(\exp\left(\frac{g_q}{h_q}\right)\right)^{\mu_q} \tag{5.18}$$

is a first integral of system (1.2)

(ii) If $p + q + r \geqslant [m(m+1)/2] + 1$ then there exist $\lambda_j, \mu_k \in \mathbb{C}$ not all zero such that $\sum_{j=1}^{p} \lambda_j K_j + \sum_{k=1}^{q} \mu_k L_k = 0$.

(iii) If $p + q + r \geqslant [m(m+1)/2] + 2$, then system (1.2) has a rational first integral, and consequently all trajectories of the system are contained in invariant algebraic curves.

(iv) There exist $\lambda_j, \mu_k \in \mathbb{C}$ not all zero, such that $\sum_{j=1}^{p} \lambda_j K_j + \sum_{k=1}^{q} \mu_k L_k = -\mathrm{div}(\dot{z}, \dot{\overline{z}})$, if and only if function (5.18) is an integrating factor of system (1.2).

(v) If $p + q + r = m(m+1)/2$ and the r independent singular points are weak (i.e., $\mathrm{div}(\dot{z}, \dot{\overline{z}}) = 0$ at $(z_\ell, \overline{z}_\ell)$), then function (5.18) is a first integral if $\sum_{j=1}^{p} \lambda_j K_j + \sum_{k=1}^{q} \mu_k L_k = 0$, or an integrating factor if $\sum_{j=1}^{p} \lambda_j K_j + \sum_{k=1}^{q} \mu_k L_k = -div(\dot{z}, \dot{\overline{z}})$, under the condition that not all $\lambda_j, \mu_k \in \mathbb{C}$ are zero.

(vi) If there exist $\lambda_j, \mu_k \in \mathbb{C}$ not all zero such that $\sum_{j=1}^{p} \lambda_j K_j + \sum_{k=1}^{q} \mu_k L_k = -s$ for some $s \in \mathbb{C}\backslash\{0\}$, then the (multivalued) function

$$f_1^{\lambda_1} \cdots f_p^{\lambda_p} \left(\exp\left(\frac{g_1}{h_1}\right)\right)^{\mu_1} \cdots \left(\exp\left(\frac{g_q}{h_q}\right)\right)^{\mu_q} \exp(st)$$

is an invariant of system (1.2).

This theorem provides many tools that allow to study a system and find a first integral should it be integrable.

5.4.2 Non classified systems with non reversible centers

As predicted in a previous observation, there are several systems that fall outside of the classification of Theorem 5.1. In the remaining of this section we will provide a complete

list of the systems found to have solutions for the system of Lyapunov constants that do not belong to the set of reversible conditions, along with their center candidates and a small study for some of them.

The complete list of the "offending" systems is:

(I) $\dot{z} = iz + \overline{z}^2 + A\overline{z}^3 + Bz^2\overline{z}$ (one Hamiltonian center),

(II) $\dot{z} = iz + \overline{z}^3 + A\overline{z}^2 + Bz^2\overline{z}$ (one Hamiltonian center),

(III) $\dot{z} = iz + z\overline{z} + Az\overline{z}^2 + Bz^2\overline{z}$ (one not classified center),

(IV) $\dot{z} = iz + z\overline{z}^2 + Az\overline{z} + Bz^2\overline{z}$ (one not classified center),

(V) $\dot{z} = iz + \overline{z}^2 + Az\overline{z}^2 + Bz^3$ (three reversible, one Hamiltonian, and one Darbouxian centers),

(VI) $\dot{z} = iz + \overline{z}^3 + Az\overline{z} + Bz^2$ (four reversible, and one Hamiltonian centers),

(VII) $\dot{z} = iz + \overline{z}^3 + Az\overline{z}^2 + Bz^3$ (two reversible, and one Hamiltonian centers),

(VIII) $\dot{z} = iz + z^2 + Az\overline{z}^2 + Bz^3$ (one reversible, and one Darbouxian centers),

(IX) $\dot{z} = iz + z^3 + Az\overline{z} + Bz^2$ (two reversible, and one not classified centers),

with $A, B \in \mathbb{C}$. Notice that since A and B are parameters (although not *free* parameters, since the center conditions often tie their values), the systems where the coefficients of the last two monomials are swapped are equivalent to these ones.

Observation 5.18. Theorem 5.1 (d) fails to predict the existence of all these Hamiltonian and Darbouxian centers found. Also, work remains to be done in order to identify which type of centers the cases (III), (IV) and (IX) are.

In the following subsections we will break down some of these cases, in order to give a notion of the difficulty that lies in completing the general 3 monomial case for any degree. The computations of Lyapunov constants and center candidates, and also the reversibility conditions, for each of the systems considered can be parallelized using PBala and every execution automatised easily using the tools developed in this Masters Thesis. However, the study of the centers that are not included in some of the four categories of Theorem 5.1 must be done in an individual basis as of now, because there are no theoretical results that show other general conditions that would serve to classify such cases. Therefore, the remaining job of studying these cases and extrapolating results that would aid in completing the characterisation is a huge task that will only be hinted at in the following pages of this work.

Cases (I), (II), (VI) and (VII)

The solution of the system formed by the first eleven Lyapunov constants provides us with the center candidates the case of degree 3. For case (I):

$$\begin{aligned}\dot{z} &= iz + \overline{z}^2 + A\overline{z}^3 + Bz^2\overline{z},\\ L_1 &= B + \overline{B}, \quad L_2 = \cdots = L_{11} = 0,\\ c_1 &= \{B = -\overline{B}\}.\end{aligned}$$

The fact that all the Lyapunov constants are zero except for the first one means that they completely reduce in the Gröbner basis formed by $\langle L_1 \rangle$. From now on, we will not distinguish whether a Lyapunov constant is actually zero or if it completely reduces with respect to the previous non zero constants.

The condition c_1 is the only center candidate for this system, and it provides a Hamiltonian center. This can be checked by the following simple computation: write $\overline{z}^2 + A\overline{z}^3 + Bz^2\overline{z} = P + iQ$, with $P, Q \in \mathbb{R}[x, y]$, and take $H = \int P\mathrm{d}y + h(x)$, with $h(x)$ such that $\partial H \partial x = Q$. Then if H satisfies $H_x P + H_y Q = 0$, it is a Hamiltonian function of the system and thus the center is of Hamiltonian kind.

Notice that Theorem 5.1 does not predict the existence of such a Hamiltonian system, since we cannot take common factor z^k for any k for the three monomials in this system, as requires Theorem 5.4. However, the condition that the coefficient of a resonant monomial must be imaginary, which is predicted by the theorem, is also fulfilled here.

For case (II) we have the Lyapunov constants and center condition:

$$\begin{aligned}\dot{z} &= iz + \overline{z}^3 + A\overline{z}^2 + Bz^2\overline{z},\\ L_1 &= B + \overline{B}, \quad L_2 = \cdots = L_{11} = 0,\\ c_1 &= \{B = -\overline{B}\}.\end{aligned}$$

This case is also Hamiltonian. Both these two previous cases show common traits that might allow us to find out a general pattern for these kind of centers in order to add it to Theorem 5.1, but this work is left to future research based on this Masters Thesis.

Case (VI) has $L_1 = i(AB - \overline{AB})$, and $L_3, L_4, L_5, L_5, L_9 \neq 0$ too big to fit the document. The center conditions are:

$$\begin{aligned}\dot{z} &= iz + \overline{z}^3 + Az\overline{z} + Bz^2,\\ c_1 &= \left\{A = -2\overline{B},\ B = -\frac{1}{2}\overline{A}\right\},\\ c_{2,3,4,5} &= \left\{A = \mp\frac{\sqrt{2}}{2}(1 \pm i)\overline{A},\ B = \mp\frac{\sqrt{2}}{2}(1 \mp i)\overline{B}\right\}.\end{aligned}$$

Centers c_2, c_3, c_4 and c_5 are reversible. The remaining case, c_1, is Hamiltonian.

Case (VII) has the following center conditions:

$$\begin{aligned}
\dot{z} &= iz + \overline{z}^3 + Az\overline{z} + Bz^2,\\
L_2 &= i(AB - \overline{AB}),\\
L_3 &= \frac{3}{8}(A^2 + \overline{A}^2) + (A\overline{B} + \overline{A}B) - \frac{3}{8}(B^2 + \overline{B}^2),\\
L_4 &= 0,\\
L_5 &= -\frac{2}{27}(B^3\overline{B} + B\overline{B}^3) + \frac{1}{18}(A\overline{B} + \overline{A}B) + \frac{1}{6}(B^2 + \overline{B}^2)\\
&\quad - \frac{1}{54}(AB\overline{B}^2 + \overline{A}B^2\overline{B}) - \frac{1}{162}(AB^3 + \overline{AB}^3),\\
L_6 &= \cdots = L_{11} = 0.
\end{aligned}$$

Its center candidates at the origin are

$$\begin{aligned}
c_1 &= \left\{A = -3\overline{B},\ B = -\frac{1}{3}\overline{A}\right\},\\
c_{2,3} &= \{\pm i\overline{A},\ B = \mp i\overline{B}\},
\end{aligned}$$

which are two reversible centers (c_2 and c_3), and a Hamiltonian one (c_1).

Case (V)

The non zero Lyapunov constants for case (V), $\dot{z} = iz + \overline{z}^2 + Az\overline{z}^2 + Bz^3$, are

$$\begin{aligned}
L_2 &= i(AB - \overline{AB}),\\
L_4 &= \frac{2}{3}(A^3 + \overline{A}^3) + \frac{8}{3}(AB\overline{B} + \overline{A}B\overline{B}) - \frac{1}{2}(A^2B - A\overline{A}B - A\overline{AB} + \overline{A}^2\overline{B})\\
&\quad + (A^2\overline{B} + \overline{A}^2B) + (B^3 + \overline{B}^3) - \frac{8}{3}(AB^2 + A\overline{B}^2 + \overline{A}B^2 + \overline{AB}^2),\\
L_5 &= \frac{7}{16}(B^3 + \overline{B}^3) + \frac{7}{24}(AB\overline{B} + \overline{A}B\overline{B} - AB^2 - A\overline{B}^2 - \overline{A}B^2 - \overline{AB}^2)\\
&\quad - \frac{7}{48}(A^2\overline{B} + \overline{A}^2B) + \frac{7}{96}(A^2B\overline{A}^2\overline{B} - A\overline{A}B - A\overline{AB}).
\end{aligned}$$

This system has several center candidates:

$$\begin{aligned}
c_1 &= \left\{A = \overline{B},\ B = \overline{A}\right\},\\
c_2 &= \left\{A = -\overline{A},\ B = -\overline{B}\right\},\\
c_3 &= \left\{A = -3\overline{B},\ B = -\frac{1}{3}\overline{A}\right\},\\
c_{4,5} &= \left\{A = \frac{1}{2}\left(1 \mp i\sqrt{3}\right)\overline{A},\ B = \frac{1}{2}\left(1 \pm i\sqrt{3}\right)\overline{A}\right\}.
\end{aligned}$$

Centers c_2, c_4 and c_5 are reversible centers. The candidate c_3 is a Hamiltonian center. The only remaining center candidate is c_1.

The system with c_1 is written as

$$\dot{z} = iz + \overline{z}^2 + Bz^3 + \overline{B}z\overline{z}^2. \tag{5.19}$$

It has four invariant straight lines $F = 1 + rx + sy$, with the slope r solution of the degree 4 polynomial:

$$P(X) = X^4 + (4\mathrm{Im}(B) - 3)X^2 + 8\mathrm{Re}(B)X - 4\mathrm{Re}(B)^2. \tag{5.20}$$

There exists a value of B such that $P(X)$ has all four real solutions: $B = \frac{1}{5}(3 + \frac{1}{10}i)$. Thus, in this case we can solve for r, s:

$$\left\{r = 1,\ s = -\frac{1}{5}\right\}, \quad \left\{r = \frac{3}{5},\ s = -1\right\},$$
$$\left\{r = \frac{2}{5}(-2 \pm \sqrt{19}),\ s = \frac{1}{5}(3 \mp \sqrt{19})\right\}.$$

In this case, the cofactors for each invariant line are

$$\begin{aligned}
K_1 &= \frac{6}{5}x^2 + \frac{34}{25}xy - \frac{6}{5}y^2 - \frac{1}{5}x - y,\\
K_2 &= \frac{6}{5}x^2 + \frac{34}{25}xy - \frac{6}{5}y^2 - x - \frac{3}{5}y,\\
K_3 &= -\frac{\sqrt{19}}{5}x - \frac{2\sqrt{19}}{5}y + \frac{6}{5}x^2 + \frac{34}{25}xy - \frac{6}{5}y^2 + \frac{3}{5}x + \frac{4}{5}y,\\
K_4 &= \frac{\sqrt{19}}{5}x + \frac{2\sqrt{19}}{5}y + \frac{6}{5}x^2 + \frac{34}{25}xy - \frac{6}{5}y^2 + \frac{3}{5}x + \frac{4}{5}y,
\end{aligned}$$

and there exist $\lambda_1, \lambda_2, \lambda_3, \lambda_4 \in \mathbb{R}$ such that

$$\lambda_1 K_1 + \lambda_2 K_2 + \lambda_3 K_3 + \lambda_4 K_4 + \mathrm{div}(P, Q) = 0,$$

where $P, Q \in \mathbb{R}[x, y]$ are the real polynomials of the system with $B = \frac{1}{5}(3 + \frac{1}{10}i)$, such that

$$P + iQ = z^2 + \frac{1}{5}\left(3 - \frac{1}{10}i\right)z\overline{z}^2 + \frac{1}{5}\left(3 + \frac{1}{10}i\right)z^3.$$

Hence, in this particular case where everything is real, applying Theorem 5.17 (iv) we can assure the existence of a first integral for the system. Therefore, this case is a Darbouxian center when $B = \frac{1}{5}(3 + \frac{1}{10}i)$.

The more general case is harder to treat, since one has to work with the formal possibly complex roots of the degree 4 polynomial (5.20), but the conclusion should also be that there exist an integrating factor of the system so the center is of Darboux type.

Case (VIII)

This case, $\dot{z} = iz + z^2 + Az\overline{z}^2 + Bz^3$, with significant Lyapunov constants

$$\begin{aligned} L_2 &= i(AB - \overline{AB}), \\ L_4 &= -\frac{1}{3}(A + \overline{A})(B - \overline{A})(A - \overline{B}), \end{aligned}$$

has one reversible center, c_1, and one which is not Hamiltonian:

$$\begin{aligned} c_1 &= \{A = -\overline{A},\ B = -\overline{B}\}, \\ c_2 &= \{A = \overline{B},\ B = \overline{A}\}. \end{aligned}$$

For c_2, the system can be written as

$$\dot{z} = iz + z^2 + Bz^3 + \overline{B}z\overline{z}^2.$$

In this case, as in (V) there are also 4 invariant straight lines with slope that is solution of a degree 4 biquadratic polynomial:

$$P(X) = X^4 + (4\mathrm{Im}(B) + 1)X^2 - 4\mathrm{Re}(B)^2.$$

It can be found that the four (complex) invariant lines have cofactors such that

$$\lambda_1 K_1 + \lambda_2 K_2 + \lambda_3 K_3 + \lambda_4 K_4 = 0,$$

for $\lambda_3, \lambda_4 \in \mathbb{C}$ and $\lambda_1 = f(\lambda_3, \lambda_4), \lambda_2 = g(\lambda_3, \lambda_4)$ complex functions of λ_3, λ_4. Thus, by Theorem 5.17 (i), we have a first integral and the center is Darbouxian.

Cases (III), (IV), and (IX)

Systems (III) and (IV) share the same center condition as (I) and (II), because their only non zero Lyapunov constant is $L_1 = B + \overline{B}$. However, the centers that arise are not Hamiltonian:

$$\begin{aligned} \dot{z} &= iz + z\overline{z} + Az\overline{z}^2 + Bz^2\overline{z}, \\ \dot{z} &= iz + z\overline{z}^2 + Az\overline{z} + Bz^2\overline{z}, \\ c_1 &= \{B = -\overline{B}\}. \end{aligned}$$

Finally, case (IX) has the following center conditions:

$$\begin{aligned} \dot{z} &= iz + z^3 + Az\overline{z}^2 + Bz^2, \\ L_1 &= i(AB - \overline{AB}), \\ L_2 &= -2(A^2 + \overline{A}^2) - (A\overline{B} + \overline{A}B), \end{aligned}$$

and $L_3 = \cdots = L_{11} = 0$. The center candidates are

$$c_1 = \{A = -\frac{1}{2}\overline{B},\ B = -2\overline{A}\},$$
$$c_{2,3} = \{A = \pm i\overline{A},\ B = \mp i\overline{B}\}.$$

Centers c_2 and c_3 are reversible, and center c_1 is not Hamiltonian. For c_1, there are no invariant algebraic straight lines, so in order to apply Theorem 5.17 we should look for higher order algebraic invariants.

These three last cases will not be classified in this work and are left to future development of this research.

Chapter 6

Final remarks and future prospect

In this Masters Thesis we developed a set of computer software that will allow us to attack not only the 3 monomial centers characterization that we discussed in the previous chapter, but also other problems which will require the computational power of PBala and the vectorised polynomial operations developed here.

The development of PBala was without doubt one of the biggest challenges faced during the months of work in this Masters Thesis. A complete interface which enables not only the author of this Thesis, but also any other researcher with access to the computing server of the Mathematics Department of the Autonomous University of Barcelona, or an equivalent, PVM-enabled, computing cluster, to easily tackle problems of a greater order of magnitude than what was available before without investing any effort in writing ad-hoc scripts or running several programs by hand to distribute the execution of their simulations.

Also, the creation of the "vectorised" operations for planar polynomials opens up the prospect not only of finishing the characterization of centers in three monomials but also new possibilities such as the search for the focus with the highest possible order in three monomials (or for other families of planar differential polynomial equations), which is a problem that has traditionally been faced by other means than this kind of *brute force* computation (e.g. the highest order systems found up to now have been obtained through perturbating known centers, see Giné, 2012).

In the near future, the author of this work will continue his work in order to obtain results about the exhaustivity of the set of centers showed in Theorem 5.1 in order to complete the work started here and complement the results obtained by Gasull et al. (2016).

Appendix A

PBala

A.1 Source code for PBala v4.0.1

A.1.1 Master program

PBala stands for Princess Bala. In the Disney film *Ant Z*, the main character is an ant that falls in love with the princess, Bala, making him work his hardest in order to earn her appreciation. As the name of the computation server used in the Mathematics Department of the Autonomous University of Barcelona is *antz*, it is only logic that a piece of software that makes Antz work as hard as possible, enabling parallelization of SPMD kind in all its cores, is named after the Princess Bala.

```
/* Job parallelizer in PVM for SPMD executions in antz computing server
 * URL: https://github.com/oscarsaleta/PVMantz
 *
 * Copyright (C) 2016  Oscar Saleta Reig
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */


/*! \mainpage PVM general parallelizer for antz
```

```
 * \author Oscar Saleta Reig
 */

/*! \file PBala.c
 * \brief Main PVM program. Distributes executions of SDMP in antz
 * \author Oscar Saleta Reig
 */

#include "antz_errcodes.h"
#include "antz_lib.h"

#include <config.h>
#include <argp.h>
#include <dirent.h>
#include <pvm3.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>


/* Program version and bug email */
const char *argp_program_version = VERSION;
const char *argp_program_bug_address = "<osr@mat.uab.cat>";

/* Program documentation */
static char doc[]= "PBala -- PVM SPMD execution parallellizer";
/* Arguments we accept */
static char args_doc[] = "programflag programfile datafile nodefile
    outdir";

/* Options we understand */
static struct argp_option options[] = {
    {"max-mem-size",      'm', "MAX_MEM", 0, "Max memory size of a task (
        KB)"},
    {"maple-single-core", 's', 0,         0, "Force single core Maple"},
    {"create-errfiles",   'e', 0,         0, "Create stderr files"},
    {"create-memfiles",   103, 0,         0, "Create memory files"},
    {"create-slavefile",  104, 0,         0, "Create node file"},
    {0}
};

/* Struct for communicating arguments to main */
struct arguments {
    char *args[5];
    long int max_mem_size;
    int maple_single_cpu, create_err, create_mem, create_slave;
};

/* Parse a single option */
static error_t parse_opt (int key, char *arg, struct argp_state *state) {
    struct arguments *arguments = state->input;
```

```
72
73      switch(key) {
74          case 'm':
75              sscanf(arg,"%ld",&(arguments->max_mem_size));
76              break;
77          case 's':
78              arguments->maple_single_cpu = 1;
79              break;
80          case 'e':
81              arguments->create_err = 1;
82              break;
83          case 103:
84              arguments->create_mem = 1;
85              break;
86          case 104:
87              arguments->create_slave = 1;
88              break;
89
90          case ARGP_KEY_ARG:
91              if (state->arg_num >= 5)
92                  argp_usage(state);
93              arguments->args[state->arg_num] = arg;
94              break;
95
96          case ARGP_KEY_END:
97              if (state->arg_num < 5)
98                  argp_usage(state);
99              break;
100
101         default:
102             return ARGP_ERR_UNKNOWN;
103     }
104     return 0;
105 }
106
107 /* argp parser */
108 static struct argp argp = {options, parse_opt, args_doc, doc};
109
110
111
112
113
114 /**
115  * Main PVM function. Handles task creation and result gathering.
116  * Call: ./PBala programFlag programFile dataFile nodeFile outDir [
        max_mem_size (KB)] [maple_single_core]
117  *
118  * \param[in] argv[1] flag for program type (0=maple,1=C,2=python,3=pari
        ,4=sage)
119  * \param[in] argv[2] program file (maple library, c executable, etc)
120  * \param[in] argv[3] data input file
121  * \param[in] argv[4] nodes file (2 cols: node cpus)
122  * \param[in] argv[5] output file directory
```

```
123  * \param[in] argv[6] (optional) aprox max memory size of biggest
         execution in KB
124  * \param[in] argv[7] (optional) flag for single core execution (Maple
         only: 0=no, 1=yes)
125  *
126  * \return 0 if successful
127  */
128 int main (int argc, char *argv[]) {
129     // Program options and arguments
130     struct arguments arguments;
131     arguments.max_mem_size = 0;
132     arguments.maple_single_cpu = 0;
133     arguments.create_err = 0;
134     arguments.create_mem = 0;
135     arguments.create_slave = 0;
136     // PVM args
137     int myparent, mytid, nTasks, taskNumber;
138     int itid;
139     int work_code;
140     // File names
141     char inp_programFile[FNAME_SIZE];
142     char inp_dataFile[FNAME_SIZE];
143     char inp_nodes[FNAME_SIZE];
144     char out_dir[FNAME_SIZE];
145     char logfilename[FNAME_SIZE];
146     char out_file[FNAME_SIZE];
147     char cwd[FNAME_SIZE];
148     // Files
149     FILE *f_data;
150     FILE *logfile;
151     FILE *f_out;
152     // Nodes variables
153     char **nodes;
154     int *nodeCores;
155     int nNodes,maxConcurrentTasks;
156     // Aux variables
157     char buffer[BUFFER_SIZE];
158     int i,j,err;
159     char aux_str[BUFFER_SIZE];
160     size_t aux_size;
161     // Task variables
162     int task_type;
163     int maple_single_cpu;
164     long int max_mem_size;
165     // Execution time variables
166     double exec_time,total_time;
167     double total_total_time=0;
168     time_t initt,endt;
169     double difft;
170
171     time(&initt);
172
173
```

```
174     /* MASTER CODE */
175     setvbuf(stderr,NULL,_IOLBF,BUFFER_SIZE);
176
177     /* Read command line arguments */
178     argp_parse(&argp, argc, argv, 0, 0, &arguments);
179
180     if (sscanf(arguments.args[0],"%d",&task_type)!=1
181             || sscanf(arguments.args[1],"%s",inp_programFile)!=1
182             || sscanf(arguments.args[2],"%s",inp_dataFile)!=1
183             || sscanf(arguments.args[3],"%s",inp_nodes)!=1
184             || sscanf(arguments.args[4],"%s",out_dir)!=1
185        ) {
186         fprintf(stderr,"%s:: ERROR - reading arguments\n",argv[0]);
187         return E_ARGS;
188     }
189     max_mem_size = arguments.max_mem_size;
190     maple_single_cpu = arguments.maple_single_cpu;
191
192
193     // sanitize maple library if single cpu is required
194     if (maple_single_cpu) {
195         sprintf(aux_str,"grep -q -F 'kernelopts(numcpus=1)' %s || (sed '1
                ikernelopts(numcpus=1);' %s > %s_tmp && mv %s %s.bak && mv %
                s_tmp %s)",
196                 inp_programFile,inp_programFile,inp_programFile,
                        inp_programFile,inp_programFile,inp_programFile,
                        inp_programFile);
197         system(aux_str);
198     }
199
200     // check if task type is correct
201     if(task_type!=1
202             && task_type!=2
203             && task_type!=3
204             && task_type!=4) {
205         fprintf(stderr,"%s:: ERROR - wrong task_type value (must be one
                of: 1,2,3,4)\n",argv[0]);
206         return E_WRONG_TASK;
207     }
208
209
210     // prepare node_info.log file if desired
211     if (arguments.create_slave) {
212         strcpy(logfilename,out_dir);
213         strcat(logfilename,"/node_info.log");
214         logfile = fopen(logfilename,"w");
215         if (logfile==NULL) {
216             fprintf(stderr,"%s:: ERROR - cannot create file %s, make sure
                        the output folder %s exists\n",argv[0],logfilename,
                    out_dir);
217             return E_OUTDIR;
218         }
219         fprintf(logfile,"# NODE CODENAMES\n");
```

```
220     }
221
222     /* Read node configuration file */
223     // Get file length (number of nodes)
224     if ((nNodes = getLineCount(inp_nodes))==-1) {
225         fprintf(stderr,"%s:: ERROR - cannot open file %s\n",argv[0],
                inp_nodes);
226         return E_NODE_LINES;
227     }
228     // Read node file
229     if ((err=parseNodefile(inp_nodes,nNodes,&nodes,&nodeCores)) == 1) {
230         fprintf(stderr,"%s:: ERROR - cannot open file %s\n",argv[0],
                inp_nodes);
231         return E_NODE_OPEN;
232     } else if (err==2) {
233         fprintf(stderr,"%s:: ERROR - while reading node file %s\n",argv
                [0],inp_nodes);
234         return E_NODE_READ;
235     }
236
237     /* INITIALIZE PVMD */
238     if (getcwd(cwd,FNAME_SIZE)==NULL) {
239         fprintf(stderr,"%s:: ERROR - cannot resolve current directory\n",
                argv[0]);
240         return E_CWD;
241     }
242     sprintf(aux_str,"echo '* ep=%s wd=%s' > hostfile",cwd,cwd);
243     system(aux_str);
244     for (i=0; i<nNodes; i++) {
245         sprintf(aux_str,"echo '%s' >> hostfile",nodes[i]);
246         system(aux_str);
247     }
248     char *pvmd_argv[1] = {"hostfile"};
249     int pvmd_argc = 1;
250     pvm_start_pvmd(pvmd_argc,pvmd_argv,1);
251     sprintf(out_file,"%s/outfile.txt",out_dir);
252     if ((f_out = fopen(out_file,"w")) == NULL) {
253         fprintf(stderr,"%s:: ERROR - cannot open output file %s\n",argv
                [0],out_file);
254         pvm_halt();
255         return E_OUTFILE_OPEN;
256     }
257     pvm_catchout(f_out);
258     // Error task id
259     mytid = pvm_mytid();
260     if (mytid<0) {
261         pvm_perror(argv[0]);
262         pvm_halt();
263         return E_PVM_MYTID;
264     }
265     // Error parent id
266     myparent = pvm_parent();
267     if (myparent<0 && myparent != PvmNoParent) {
```

```
268         pvm_perror(argv[0]);
269         pvm_halt();
270         return E_PVM_PARENT;
271     }
272     /***/
273 
274     // Max number of tasks running at once
275     maxConcurrentTasks = 0;
276     for (i=0; i<nNodes; i++) {
277         maxConcurrentTasks += nodeCores[i];
278     }
279 
280     // Read how many tasks we have to perform
281     if((nTasks = getLineCount(inp_dataFile))==-1) {
282         fprintf(stderr,"%s:: cannot open data file %s\n",argv[0],
                inp_dataFile);
283         pvm_halt();
284         return E_DATAFILE_LINES;
285     }
286 
287     // Print execution info
288     fprintf(stderr,"PRINCESS BALA v%s\n",VERSION);
289     fprintf(stderr,"System call: ");
290     for (i=0;i<argc;i++)
291         fprintf(stderr,"%s ",argv[i]);
292     fprintf(stderr,"\n\n");
293 
294     fprintf(stderr,"%s:: INFO - will use executable %s\n",argv[0],
            inp_programFile);
295     fprintf(stderr,"%s:: INFO - will use datafile %s\n",argv[0],
            inp_dataFile);
296     fprintf(stderr,"%s:: INFO - will use nodefile %s\n",argv[0],inp_nodes
            );
297     fprintf(stderr,"%s:: INFO - results will be stored in %s\n\n",argv
            [0],out_dir);
298 
299 
300     fprintf(stderr,"%s:: INFO - will use nodes ",argv[0]);
301     for (i=0; i<nNodes-1; i++)
302         fprintf(stderr,"%s (%d), ",nodes[i],nodeCores[i]);
303     fprintf(stderr,"%s (%d)\n",nodes[nNodes-1],nodeCores[nNodes-1]);
304     fprintf(stderr,"%s:: INFO - will create %d tasks for %d slaves\n\n",
            argv[0],nTasks,maxConcurrentTasks);
305 
306 
307 
308 
309     // Spawn all the tasks
310     int taskId[maxConcurrentTasks];
311     itid = 0;
312     int numt;
313     int numnode=0;
314     for (i=0; i<nNodes; i++) {
```

```
315         for (j=0; j<nodeCores[i]; j++) {
316             numt = pvm_spawn("task",NULL,PvmTaskHost,nodes[i],1,&taskId[
                itid]);
317             if (numt != 1) {
318                 fprintf(stderr,"%s:: ERROR - %d creating task %4d in node
                        %s\n",
319                     argv[0],numt,taskId[itid],nodes[i]);
320                 fflush(stderr);
321                 pvm_perror(argv[0]);
322                 pvm_halt();
323                 return E_PVM_SPAWN;
324             }
325             // Send info to task
326             pvm_initsend(PVM_ENCODING);
327             pvm_pkint(&itid,1,1);
328             pvm_pkint(&task_type,1,1);
329             pvm_pklong(&max_mem_size,1,1);
330             pvm_pkint(&(arguments.create_err),1,1);
331             pvm_pkint(&(arguments.create_mem),1,1);
332             pvm_send(taskId[itid],MSG_GREETING);
333             fprintf(stderr,"%s:: CREATED_SLAVE - created slave %d\n",argv
                [0],itid);
334             if (arguments.create_slave)
335                 fprintf(logfile,"# Node %2d -> %s\n",numnode,nodes[i]);
336             numnode++;
337             // Do not create more tasks than necessary
338             if (numnode >= nTasks)
339                 break;
340             itid++;
341         }
342     }
343     fprintf(stderr,"%s:: INFO - all slaves created successfully\n\n",argv
            [0]);
344
345     // First batch of work sent at once (we will listen to answers later)
346     f_data = fopen(inp_dataFile,"r");
347     if (arguments.create_slave)
348         fprintf(logfile,"\nNODE,TASK\n");
349     int firstBatchSize = nTasks < maxConcurrentTasks ? nTasks :
            maxConcurrentTasks;
350     work_code = MSG_GREETING;
351     for (i=0; i<firstBatchSize; i++) {
352         if (fgets(buffer,BUFFER_SIZE,f_data)!=NULL) {
353             if (sscanf(buffer,"%d",&taskNumber)!=1) {
354                 fprintf(stderr,"%s:: ERROR - first column of data file
                        must be task id\n",argv[0]);
355                 pvm_halt();
356                 return E_DATAFILE_FIRSTCOL;
357             }
358             pvm_initsend(PVM_ENCODING);
359             pvm_pkint(&work_code,1,1);
360             pvm_pkint(&taskNumber,1,1);
361             pvm_pkstr(inp_programFile);
```

```
362             pvm_pkstr(out_dir);
363             /* parse arguments (skip tasknumber) */
364             sprintf(aux_str,"%d",taskNumber);
365             aux_size = strlen(aux_str);
366             buffer[strlen(buffer)-1]=0;
367             // copy to aux_str the data line from after the first ","
368             sprintf(aux_str,"%s",&buffer[aux_size+1]);
369             pvm_pkstr(aux_str);
370             // create file for pari execution if needed
371             if (task_type == 3) {
372                 parifile(taskNumber,aux_str,inp_programFile,out_dir);
373                 fprintf(stderr,"%s:: CREATED_SCRIPT - creating auxiliary
                        Pari script for task %d\n",argv[0],taskNumber);
374             } else if (task_type == 4) {
375                 sagefile(taskNumber,aux_str,inp_programFile,out_dir);
376                 fprintf(stderr,"%s:: CREATED_SCRIPT - creating auxiliary
                        Sage script for task %d\n",argv[0],taskNumber);
377             }
378 
379             // send the job
380             pvm_send(taskId[i],MSG_WORK);
381             fprintf(stderr,"%s:: TASK_SENT - sent task %4d for execution\
                    n",argv[0],taskNumber);
382             if (arguments.create_slave)
383                 fprintf(logfile,"%2d,%4d\n",i,taskNumber);
384         }
385     }
386     fprintf(stderr,"%s:: INFO - first batch of work sent\n\n",argv[0]);
387 
388 
389     // Close logfile so it updates in file system
390     if (arguments.create_slave)
391         fclose(logfile);
392     // Keep assigning work to nodes if needed
393     int status, unfinished_tasks_present = 0;
394     FILE *unfinishedTasks;
395     char unfinishedTasks_name[FNAME_SIZE];
396     sprintf(unfinishedTasks_name,"%s/unfinished_tasks.txt",out_dir);
397     unfinishedTasks=fopen(unfinishedTasks_name,"w");
398     fclose(unfinishedTasks);
399     if (nTasks > maxConcurrentTasks) {
400         for (j=maxConcurrentTasks; j<nTasks; j++) {
401             // Receive answer from slaves
402             pvm_recv(-1,MSG_RESULT);
403             pvm_upkint(&itid,1,1);
404             pvm_upkint(&taskNumber,1,1);
405             pvm_upkint(&status,1,1);
406             pvm_upkstr(aux_str);
407             // Check if response is error at forking
408             if (status == ST_FORK_ERR) {
409                 fprintf(stderr,"%s:: ERROR - could not fork process for
                        task %d at slave %d\n",
410                     argv[0],taskNumber,itid);
```

```
411                 unfinishedTasks = fopen(unfinishedTasks_name,"a");
412                 fprintf(unfinishedTasks,"%d,%s\n",taskNumber,aux_str);
413                 fclose(unfinishedTasks);
414                 unfinished_tasks_present = 1;
415             } else {
416                 pvm_upkdouble(&exec_time,1,1);
417                 // Check if task was killed or completed
418                 if (status == ST_TASK_KILLED) {
419                     fprintf(stderr,"%s:: ERROR - task %4d was stopped or
                            killed\n",argv[0],taskNumber);
420                     unfinishedTasks = fopen(unfinishedTasks_name,"a");
421                     fprintf(unfinishedTasks,"%d,%s\n",taskNumber,aux_str)
                            ;
422                     fclose(unfinishedTasks);
423                     unfinished_tasks_present = 1;
424                 } else
425                     fprintf(stderr,"%s:: TASK_COMPLETED - task %4d
                            completed in %10.5G seconds\n",argv[0],taskNumber,
                            exec_time);
426             }
427             // Assign more work until we're done
428             if (fgets(buffer,BUFFER_SIZE,f_data)!=NULL) {
429                 // Open logfile for appending work
430                 if (arguments.create_slave)
431                     logfile = fopen(logfilename,"a");
432                 if (sscanf(buffer,"%d",&taskNumber)!=1) {
433                     fprintf(stderr,"%s:: ERROR - first column of data
                            file must be task id\n",argv[0]);
434                     pvm_halt();
435                     return E_DATAFILE_FIRSTCOL;
436                 }
437                 pvm_initsend(PVM_ENCODING);
438                 pvm_pkint(&work_code,1,1);
439                 pvm_pkint(&taskNumber,1,1);
440                 pvm_pkstr(inp_programFile);
441                 pvm_pkstr(out_dir);
442                 sprintf(aux_str,"%d",taskNumber);
443                 aux_size = strlen(aux_str);
444                 buffer[strlen(buffer)-1]=0;
445                 sprintf(aux_str,"%s",&buffer[aux_size+1]);
446                 pvm_pkstr(aux_str);
447                 // create file for pari execution if needed
448                 if (task_type == 3) {
449                     parifile(taskNumber,aux_str,inp_programFile,out_dir);
450                     fprintf(stderr,"%s:: CREATED_SCRIPT - creating
                            auxiliary Pari script for task %d\n",argv[0],
                            taskNumber);
451                 } else if (task_type == 4) {
452                     sagefile(taskNumber,aux_str,inp_programFile,out_dir);
453                     fprintf(stderr,"%s:: CREATED_SCRIPT - creating
                            auxiliary Sage script for task %d\n",argv[0],
                            taskNumber);
454                 }
```

```
455                 // send the job
456                 pvm_send(taskId[itid],MSG_WORK);
457                 fprintf(stderr,"%s:: TASK_SENT - sent task %3d for
                        execution\n",argv[0],taskNumber);
458                 if (arguments.create_slave) {
459                     fprintf(logfile,"%2d,%4d\n",itid,taskNumber);
460                     fclose(logfile);
461                 }
462             }
463         }
464     }
465
466     // Listen to answers from slaves that keep working
467     work_code = MSG_STOP;
468     for (i=0; i<firstBatchSize; i++) {
469         // Receive answer from slaves
470         pvm_recv(-1,MSG_RESULT);
471         pvm_upkint(&itid,1,1);
472         pvm_upkint(&taskNumber,1,1);
473         pvm_upkint(&status,1,1);
474         pvm_upkstr(aux_str);
475         // Check if response is error at forking
476         if (status == ST_FORK_ERR) {
477             fprintf(stderr,"%s:: ERROR - could not fork process for task
                        %d at slave %d\n",
478                     argv[0],taskNumber,itid);
479             unfinishedTasks = fopen(unfinishedTasks_name,"a");
480             fprintf(unfinishedTasks,"%d,%s\n",taskNumber,aux_str);
481             fclose(unfinishedTasks);
482             unfinished_tasks_present = 1;
483         } else {
484             pvm_upkdouble(&exec_time,1,1);
485             // Check if task was killed or completed
486             if (status == ST_TASK_KILLED) {
487                 fprintf(stderr,"%s:: ERROR - task %4d was stopped or
                            killed\n",argv[0],taskNumber);
488                 unfinishedTasks = fopen(unfinishedTasks_name,"a");
489                 fprintf(unfinishedTasks,"%d,%s\n",taskNumber,aux_str);
490                 fclose(unfinishedTasks);
491                 unfinished_tasks_present = 1;
492             } else
493                 fprintf(stderr,"%s:: TASK_COMPLETED - task %4d completed
                            in %10.5G seconds\n",argv[0],taskNumber,exec_time);
494         }
495         pvm_upkdouble(&total_time,1,1);
496         // Shut down slave
497         pvm_initsend(PVM_ENCODING);
498         pvm_pkint(&work_code,1,1);
499         pvm_send(taskId[itid],MSG_STOP);
500         fprintf(stderr,"%s:: INFO - shutting down slave %2d (total
                execution time: %13.5G seconds)\n",argv[0],itid,total_time);
501         total_total_time += total_time;
502     }
```

```
503
504     // Final message
505     time(&endt);
506     difft = difftime(endt,initt);
507     fprintf(stderr,"\n%s:: END OF EXECUTION.\nCombined computing time:
            %14.5G seconds.\nTotal execution time:    %14.5G seconds.\n",argv
            [0],total_total_time,difft);
508
509     free(nodes);
510     free(nodeCores);
511     fclose(f_data);
512     fclose(f_out);
513
514
515     // remove tmp program (if modified)
516     if (maple_single_cpu) {
517         sprintf(aux_str,"[ ! -f %s.bak ] || mv %s.bak %s",inp_programFile
                ,inp_programFile,inp_programFile);
518         system(aux_str);
519     }
520     // remove tmp pari/sage programs (if created)
521     if (task_type == 3 || task_type == 4) {
522         DIR *dir;
523         struct dirent *ent;
524         dir = opendir(out_dir);
525         while ((ent = readdir(dir))) {
526             if (strstr(ent->d_name,"auxprog")!=NULL) {
527                 sprintf(aux_str,"%s/%s",out_dir,ent->d_name);
528                 remove(aux_str);
529             }
530         }
531         closedir(dir);
532     }
533     // remove unfinished_tasks.txt file if empty
534     if (!unfinished_tasks_present) {
535         sprintf(aux_str,"rm %s",unfinishedTasks_name);
536         system(aux_str);
537     }
538
539     pvm_catchout(0);
540     pvm_halt();
541
542     return 0;
543 }
```

Listing A.1: PBala v4.0.1 master program source code, *PBala.c*.

A.1.2 Slave program

```
1 /* Job parallelizer in PVM for SPMD executions in antz computing server
2  * URL: https://github.com/oscarsaleta/PVMantz
```

```
 *
 * Copyright (C) 2016  Oscar Saleta Reig
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

/*! \file task.c
 * \brief PVM task. Adapts to different programs, forks execution to
     track mem usage
 * \author Oscar Saleta Reig
 */
#include "antz_errcodes.h"
#include "antz_lib.h"

#include <config.h>

#include <fcntl.h>
#include <pvm3.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

#include <sys/resource.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>


/**
 * Main task function.
 *
 * \return 0 if successful
 */
int main(int argc, char *argv[]) {
    int myparent,taskNumber; // myparent is the master
    int me; // me is the PVM id of this child
    // work_code is a flag that tells the child what to do
    int work_code;
    char inp_programFile[FNAME_SIZE]; // name of the program
    char out_dir[FNAME_SIZE]; // name of output directory
```

```
54      char arguments[BUFFER_SIZE]; // string of arguments as read from data
            file
55      int task_type; // 0:maple, 1:C, 2:python
56      long int max_task_size; // if given, max size in KB of a spawned
            process
57      int flag_err; // 0=no err files, 1=yes err files
58      int flag_mem; // 0=no mem files, 1=yes mem files
59      int i; // for loops
60      clock_t initt,endt;
61      double difft, totalt=0;
62      int state;
63
64      myparent = pvm_parent();
65
66
67      // Be greeted by master
68      pvm_recv(myparent,MSG_GREETING);
69      pvm_upkint(&me,1,1);
70      pvm_upkint(&task_type,1,1);
71      pvm_upklong(&max_task_size,1,1);
72      pvm_upkint(&flag_err,1,1);
73      pvm_upkint(&flag_mem,1,1);
74
75
76      /* Perform generic check or use specific size info?
77       *  memcheck_flag = 0 means generic check
78       *  memcheck_flag = 1 means specific info
79       */
80      int memcheck_flag = max_task_size > 0 ? 1 : 0;
81
82      // Work work work
83      while (1) {
84          /* Race condition. Mitigated by executing few CPUs on each node
85           * Explanation: 2 tasks could check memory simultaneously and
86           *  both conclude that there is enough because they see the same
87           *  output, but maybe there is not enough memory for 2 tasks.
88           */
89          if (memcheck(memcheck_flag,max_task_size) == 1) {
90              sleep(60); // arbitrary number that could be much lower
91              continue;
92          }
93          // Receive inputs
94          pvm_recv(myparent,MSG_WORK);
95          pvm_upkint(&work_code,1,1);
96          if (work_code == MSG_STOP) // if master tells task to shutdown
97              break;
98          pvm_upkint(&taskNumber,1,1);
99          pvm_upkstr(inp_programFile);
100         pvm_upkstr(out_dir);
101         pvm_upkstr(arguments); // string of comma-separated arguments
                read from datafile
102
103         time(&initt);
```

```
104         /* Fork one process that will do the execution
105          * the "parent task" will only wait for this process to end
106          * and then report resource usage via getrusage()
107          */
108         pid_t pid = fork();
109         // If fork fails, notify master and exit
110         if (pid<0) {
111             fprintf(stderr,"ERROR - task %d could not spawn execution
                process\n",taskNumber);
112             int state=ST_FORK_ERR;
113             pvm_initsend(PVM_ENCODING);
114             pvm_pkint(&me,1,1);
115             pvm_pkint(&taskNumber,1,1);
116             pvm_pkint(&state,1,1);
117             pvm_pkstr(arguments);
118             pvm_pkdouble(&totalt,1,1);
119             pvm_send(myparent,MSG_RESULT);
120             pvm_exit();
121             exit(1);
122         }
123         //Child code (work done here)
124         if (pid == 0) {
125             char output_file[BUFFER_SIZE];
126             int err; // for executions
127 
128             // Move stdout to taskNumber_out.txt
129             sprintf(output_file,"%s/task%d_stdout.txt",out_dir,taskNumber
                );
130             int fd = open(output_file,O_WRONLY|O_CREAT|O_TRUNC,0666);
131             dup2(fd,1);
132             close(fd);
133 
134             // Move stderr to taskNumber_err.txt
135             if (flag_err) {
136                 sprintf(output_file,"%s/task%d_stderr.txt",out_dir,
                    taskNumber);
137                 fd = open(output_file,O_WRONLY|O_CREAT|O_TRUNC,0666);
138                 dup2(fd,2);
139                 close(fd);
140             }
141 
142             /*
143              * GENERATE EXECUTION OF PROGRAM
144              */
145             /* MAPLE */
146             if (task_type == 0) {
147                 // NULL-terminated array of strings for calling the Maple
                     script
148                 char **args;
149                 // 0: maple, 1: taskid, 2: X, 3: input, 4: NULL
150                 int nargs=4;
151                 args = (char**)malloc((nargs+1)*sizeof(char*));
152                 // Do not malloc for NULL
```

```
153                 for (i=0;i<nargs;i++)
154                     args[i] = malloc(BUFFER_SIZE);
155                 // Fill up the array with strings
156                 sprintf(args[0],"maple");
157                 sprintf(args[1],"-tc \"taskId:=%d\"",taskNumber);
158                 sprintf(args[2],"-c \"taskArgs:=[%s]\"",arguments);
159                 sprintf(args[3],"%s",inp_programFile);
160                 args[4] = NULL;
161
162                 // Call the execution and check for errors
163                 err = execvp(args[0],args);
164                 perror("ERROR:: child Maple process");
165                 exit(err);
166
167             /* C */
168             } else if (task_type == 1) {
169                 // Preparations for C or Python execution
170                 char *arguments_cpy;
171                 arguments_cpy=malloc(strlen(arguments));
172                 strcpy(arguments_cpy,arguments);
173                 // This counts how many commas there are in arguments,
                        giving the number
174                 // of arguments passed to the program
175                 for(i=0;arguments_cpy[i];arguments_cpy[i]==','?i++:*
                        arguments_cpy++);
176                 int nargs = i+1; // i = number of commas
177                 int nargs_tot; // will be the total number of arguments (
                        nargs+program name+etc)
178                 char *token; // used for tokenizing arguments
179                 char **args; // this is the NULL-terminated array of
                        strings
180
181                 /* In this case it's necessary to parse the arguments
                        string
182                  * breaking it into tokens and then arranging the args of
                            the
183                  * system call in a NULL-terminated array of strings
                            which we
184                  * pass to execvp
185                  */
186                 // Tokenizing breaks the original string so we make a
                        copy
187                 strcpy(arguments_cpy,arguments);
188                 // Args in system call are (program tasknum arguments),
                        so 2+nargs
189                 nargs_tot = 2+nargs;
190                 args = (char**)malloc((nargs_tot+1)*sizeof(char*));
191                 args[nargs_tot]=NULL; // NULL-termination of args
192                 for (i=0;i<nargs_tot;i++)
193                     args[i] = malloc(BUFFER_SIZE);
194                 // Two first command line arguments
195                 //strcpy(args[0],inp_programFile);
196                 sprintf(args[0],"./%s",inp_programFile);
```

```
197                 sprintf(args[1],"%d",taskNumber);
198                 // Tokenize arguments_cpy
199                 token = strtok(arguments_cpy,",");
200                 // Copy the token to its place in args
201                 for (i=2;i<nargs_tot;i++) {
202                     strcpy(args[i],token);
203                     token = strtok(NULL,",");
204                 }
205                 // Call the execution and check for errors
206                 err = execvp(args[0],args);
207                 perror("ERROR:: child C process");
208                 exit(err);
209 
210             /* PYTHON */
211             } else if (task_type == 2) {
212                 // Preparations for C or Python execution
213                 char *arguments_cpy;
214                 arguments_cpy=malloc(strlen(arguments));
215                 strcpy(arguments_cpy,arguments);
216                 // This counts how many commas there are in arguments,
                        giving the number
217                 // of arguments passed to the program
218                 for(i=0;arguments_cpy[i];arguments_cpy[i]==','?i++:*
                        arguments_cpy++);
219                 int nargs = i+1; // i = number of commas
220                 int nargs_tot; // will be the total number of arguments (
                        nargs+program name+etc)
221                 char *token; // used for tokenizing arguments
222                 char **args; // this is the NULL-terminated array of
                        strings
223 
224                 // Same as in C, but adding "python" as first argument
225                 // Tokenizing breaks the original string so we make a
                        copy
226                 strcpy(arguments_cpy,arguments);
227                 // args: (0)-python (1)-program (2)-tasknumber (3..nargs
                        +3)-arguments
228                 nargs_tot = 3+nargs;
229                 args = (char**)malloc((nargs_tot+1)*sizeof(char*));
230                 args[nargs_tot]=NULL; // NULL-termination of args
231                 for (i=0;i<nargs_tot;i++)
232                     args[i] = malloc(BUFFER_SIZE);
233                 // Two first command line arguments
234                 strcpy(args[0],"python");
235                 strcpy(args[1],inp_programFile);
236                 sprintf(args[2],"%d",taskNumber);
237                 // Tokenize arguments_cpy
238                 token = strtok(arguments_cpy,",");
239                 // Copy the token to its place in args
240                 for (i=3;i<nargs_tot;i++) {
241                     strcpy(args[i],token);
242                     token = strtok(NULL,",");
243                 }
```

```
244
245                 // Call the execution and check for errors
246                 err = execvp(args[0],args);
247                 perror("ERROR:: child Python process");
248                 exit(err);
249
250             /* PARI/GP */
251             } else if (task_type == 3) {
252                 // NULL-terminated array of strings
253                 char **args;
254                 char filename[FNAME_SIZE];
255                 sprintf(filename,"%s/auxprog-%d.gp",out_dir,taskNumber);
256                 // 0: gp, 1: -f, 2: -s400G, 3: file, 4: NULL
257                 int nargs=4;
258                 args = (char**)malloc((nargs+1)*sizeof(char*));
259                 // Do not malloc for NULL
260                 for (i=0;i<nargs;i++)
261                     args[i] = malloc(BUFFER_SIZE);
262                 // Fill up the array with strings
263                 sprintf(args[0],"gp");
264                 sprintf(args[1],"-f");
265                 sprintf(args[2],"-s400G");
266                 sprintf(args[3],"%s",filename);
267                 args[4] = NULL;
268
269                 // Call the execution and check for errors
270                 err = execvp(args[0],args);
271                 perror("ERROR:: child PARI process");
272                 exit(err);
273
274             /* SAGE */
275             } else if (task_type == 4) {
276                 // NULL-terminated array of strings
277                 char **args;
278                 char filename[FNAME_SIZE];
279                 sprintf(filename,"%s/auxprog-%d.sage",out_dir,taskNumber)
                        ;
280                 // 0: sage, 1: file, 3: NULL
281                 int nargs=2;
282                 args = (char**)malloc((nargs+1)*sizeof(char*));
283                 // Do not malloc for NULL
284                 for (i=0;i<nargs;i++)
285                     args[i] = malloc(BUFFER_SIZE);
286                 // Fill up the array with strings
287                 sprintf(args[0],"sage");
288                 sprintf(args[1],"%s",filename);
289                 //sprintf(args[1],"-c \"taskId=%d;taskArgs:=[%s];load('%s
                        ')\"",taskNumber,arguments,inp_programFile);
290                 args[2] = NULL;
291
292                 // Call the execution and check for errors
293                 err = execvp(args[0],args);
294                 perror("ERROR:: child Sage process");
```

```
295                     exit(err);
296                 }
297
298             }
299
300
301             /* Attempt at measuring memory usage for the child process */
302             siginfo_t infop; // Stores information about the child execution
303             waitid(P_PID,pid,&infop,WEXITED); // Wait for the execution to
                    end
304             // Computation time
305             time(&endt);
306             difft = difftime(endt,initt);
307             totalt += difft;
308             if (infop.si_code == CLD_KILLED
309                     || infop.si_code == CLD_DUMPED) {
310                 prterror(pid,taskNumber,out_dir,difft);
311                 state=ST_TASK_KILLED;
312             } else if (flag_mem) {
313                 struct rusage usage; // Stores information about the child's
                        resource usage
314                 getrusage(RUSAGE_CHILDREN,&usage); // Get child resource
                        usage
315                 prtusage(pid,taskNumber,out_dir,usage); // Print resource
                        usage to file
316                 state=0;
317             }
318
319             // Send response to master
320             pvm_initsend(PVM_ENCODING);
321             pvm_pkint(&me,1,1);
322             pvm_pkint(&taskNumber,1,1);
323             pvm_pkint(&state,1,1);
324             pvm_pkstr(arguments);
325             pvm_pkdouble(&difft,1,1);
326             pvm_pkdouble(&totalt,1,1);
327             pvm_send(myparent,MSG_RESULT);
328         }
329
330
331         // Dismantle slave
332         pvm_exit();
333         exit(0);
334     }
```

Listing A.2: PBala v4.0.1 slave program source code, *task.c*.

A.1.3 Readme extract

All the source code, license files and manuals can be found at https://github.com/oscarsaleta/PBala. The following is an extract from the *README.md* file.

Introduction

This C program uses PVM libraries in order to create a parallellization interface for

- **Maple** scripts
- **C** programs
- **Python** scripts
- **Pari/GP** scripts (although Pari should be also supported through the Pari C/C++ library or through gp2c by manually editing the output C code and compiling as a C program using the command provided by gp2c. The executable can then be run using the C module of this software.)
- **Sage** scripts

This interface lets the user execute a same script/program over multiple input data in several CPUs located at the antz computing server. It sports memory management so nodes do not run out of RAM due to too many processes being started in the same node. It also reports resource usage data after execution.

Usage

As of v4.0.0:

The program admits standard `--help` (`-?`), `--usage` and `--version` (`-V`) arguments.

Output from `./PBala --help`:

```
Usage: PBala [OPTION...] programflag programfile datafile nodefile outdir
PBala -- PVM SPMD execution parallellizer

  -e, --create-errfiles      Create stderr files
  -g, --create-memfiles      Create memory files
  -h, --create-slavefile     Create node file
  -m, --max-mem-size=MAX_MEM Max memory size of a task (KB)
  -s, --maple-single-core    Force single core Maple
  -?, --help                 Give this help list
      --usage                Give a short usage message
```

```
  -V, --version              Print program version

Mandatory or optional arguments to long options are also mandatory or
    optional
for any corresponding short options.

Report bugs to <osr@mat.uab.cat>.
```

Output from `./PBala --usage`:

```
Usage: PBala [-eghs?V] [-m MAX_MEM] [--create-errfiles] [--create-
    memfiles]
             [--create-slavefile] [--max-mem-size=MAX_MEM] [--maple-single
                 -core]
             [--help] [--usage] [--version]
             programflag programfile datafile nodefile outdir
```

Mandatory arguments explained:

- `programflag`:
  - 0 = Maple
  - 1 = C
  - 2 = Python
  - 3 = Pari (as of v1.0.0)
  - 4 = Sage (as of v3.0.0)
- `programfile`: path to program file
- `datafile`: path to data file
  - Line format is "tasknumber,arg1,arg2,...,argN"
- `nodefile`: path to PVM node file
  - Line format is "nodename numer_of_processes"
- `outdir`: path to output directory

Options explained:

- `-e, --create-errfiles`: Save stderr output for each execution in a *task*_stderr.txt* file
- `-g, --create-memfiles`: Save memory info for each execution in a *task*_mem.txt* file
- `-h, --create-slavefile`: Save a log of which task is given to which slave in *node_info.txt*

- `-m, --max-mem-size=MAX_MEM`: max amount of RAM (in KB) that a single execution can require
- `-s, --maple-single-core`: Force Maple to use a single core for its executions

Conventions

For Maple, we define 2 variables: `taskId` and `taskArgs`. `taskId` is an identifier for the task number that we are sending to the Maple script. `taskArgs` are the actual arguments that Maple has to use to do the computations. It is important to use these names because they are passed to Maple this way.

For C and Python we use the argv arrays so make sure the program can read and use those variables (and perform the error checking because this software has no way of knowing if the data file is suitable for your program).

Pari and Sage are executed by creating auxiliary scripts where `taskId` and `taskArgs` are defined, so they could be directly used in the scripts just like in Maple.

Appendix B

Lyapunov computation functions

B.1 Algorithm for computing "the next Lyapunov constant" (PARI/GP)

The following code snippet is the PARI/GP function `nextlyapunov()`. It takes as input a field R in the format of a GP *List*, and optional arguments for H (the first integral with each of its coefficients in *List* form) and L, which is a *List* of all the Lyapunov constants computed and their degrees. This function is designed with the idea that its output can be used as input for successive calls to it. This way, if we want to compute a certain number of Lyapunov constants, perfom some manipulations on them, and compute more depending on the results obtained, we do not have to start all over again, because intermediate results are taken so computation starts in the correct order.

The source code for every Lyapunov related script used in this work is publicly available at https://github.com/oscarsaleta/Lyapunov (although, unlike with the PBala repository, this one lacks completely of proper documentation).

```
nextlyapunov(R,H=List([[0,1,0]]),L=List())=
{
    my(i,N,maxL,lastdg,g,d,h);
    if (#H%2==0,
        return("Invalid H");
    );
    i = #H+2;
    k = floor(i/2)-1;
    N = 0;
    for(i=1,#R,
        N = max(N,#R[i]);
    );
    maxL = N*N+3*N-7;
    lastdg = 2*(maxL+1);
```

```
15      while (i<lastdg,
16          /* Odd degree */
17          g = indcoef(i,H,R);
18          d = diagmat(i);
19          h = vector(i+1);
20          for (j=1,i+1,
21              h[j] = g[j]/d[j];
22          );
23          listput(H,h);
24          /* Even degree */
25          k++;
26          g = indcoef(i+1,H,R);
27          d = diagmat(i+1);
28          h = vector(i+2);
29          for (j=1,i+2,
30              if (d[j]!=0,
31                  h[j] = g[j]/d[j];
32              );
33          );
34          listput(H,h);
35          const = g[((i+1)/2)+1]/I;
36          if (const!=0,
37              listput(L,[k,const]);
38              /* Return H and the constant found */
39              return(List([L,H]));
40          );
41          i+=2;
42      );
43      return(-1);
44  }
```

Listing B.1: PARI/GP function `nextlyapunov()`.

B.2 Sage script for computations with 3 monomials

The following Sage script is prepared to be executed in parallel using PBala. It takes as input the `taskArgs` vector, which contains the exponents of each monomial. Then it checks if the exponents alone are enough to classify the system as a center using Theorem 5.1. If these conditions are not met, the program calls PARI/GP. First (line 38) it loads the `taskArgs` vector into GP, then it executes the *lyap.gp* script (which also can be read below), which defines a GP *List* called `R` that is used in line 42 to compute the first nonzero Lyapunov constant.

If the `nextlyapunov(R)` call does not find a nonzero constant, then we call it a center and exit the program. Else, we define a multivariate polynomial ring over a finite field using Singular (this is where the ideals and Gröbner basis will be defined). Then we append the Lyapunov constant and its order to two Python (Sage) lists, and iteratively

compute the following constants and reduce them. If they do not completely reduce, we add them to the list (and print them in a Maple-friendly format for later processing).

Finally, we define a polynomial ring over a finite field with a parameter i and minimal polynomial i^2+1, so we can write the reversibility conditions in the same Maple format.

```
# -*- coding: utf-8 -*-
set_verbose(-1)

# Llegir dades
taskId=taskArgs[0]
k=taskArgs[1]
l=taskArgs[2]
m=taskArgs[3]
n=taskArgs[4]
p=taskArgs[5]
q=taskArgs[6]

status=""+str(taskId)+","+str(k)+","+str(l)+","+str(m)+","+str(n)+","+str
    (p)+","+str(q)


print("#R = i*z + z^"+str(k)+"*w^"+str(l)+" + (a1+b1*i)*z^"+str(m)+"*w^"+
    str(n)+" + (a2+b2*i)*z^"+str(p)+"*w^"+str(q))
grau = max(max(k+l,m+n),p+q)
print("#Field degree = "+str(grau))
print("\n")

# Comprovar si es centre d'algun tipus abans de fer cap calcul
# (a) Quadratic Darboux (Zoladek)
if k==2 and q==2 and m==1 and n==1 and l==0 and p==0:
    print(status+",CENTRE A")
    sys.exit(0)
# (b) Holomorphic centers
if l==0 and n==0 and q==0:
    print(status+",CENTRE B")
    sys.exit(0)
# (d) Hamiltonian/new Darboux
if k==m and m==p and l!=n and n!=q and k-l-1!=0:
    print(status+",CENTRE D")
    sys.exit(0)

# (c) Si hem arribat fins aqui, tenim un possible centre reversible
print("\n#Possible reversible centre, computing Liapunov constants...")

# Carregar dades i funcions en Pari
gp("taskArgs=["+str(k)+","+str(l)+","+str(m)+","+str(n)+","+str(p)+","+
    str(q)+"]")
gp("read(\"lyap.gp\")")

# Calcular primera constant no nul.la
gp("l=nextlyapunov(R);")
if sage_eval(gp.eval("l==-1"))==1:
```

```
    print("\n#"+status+"Center")
    sys.exit()

primer = 32003
#primer = 0
R = singular.ring(primer,'(a1,b1,a2,b2)','dp')
if (primer!=0):
    print("#Using ring on a finite field modulo 32003")

lyaps = []
lyaps.append(gp.eval("l[1][1][2]"))
ordres = []
ordres.append(gp.eval("l[1][1][1]"))

print("\nL"+ordres[0]+":="+lyaps[0]+":\n")
ordre = ordres[0]

i = 1
reduct = 0
while (int(ordres[0])<=grau*grau+3*grau-7):
    # Calcular constant no nul.la
    gp("l=nextlyapunov(R,l[2],l[1]);")
    f=gp.eval("l[1]["+str(i+1)+"][2]")
    o=gp.eval("l[1]["+str(i+1)+"][1]")
    # Generar ideal amb constants anteriors
    I = singular.ideal(lyaps)
    # Reduir nova constant resp les anteriors
    B = I.groebner()
    # Si redueix, pararem si en portem 2 seguides o passem de n*n+3*n-7
    g = singular(f).sage().reduce(B.sage())
    if g==0:
        reduct += 1
        print("L"+o+":="+str(g)+": #reduced\n")
        if int(o)>grau*(grau+2)-1 or reduct>3/2*grau:
            break
    else:
        # Si no redueix, guardem l'ultim ordre
        lyaps.append(str(g))
        print("L"+o+":="+str(g)+":\n")
        ordres.append(o)
        ordre = o
        reduct = 0
    i += 1


# Busquem les condicions de centre
print("# Computing reversible center conditions\n")
if primer!=0:
    S.<i>=GF(primer)[]
    SS.<I>=S.quotient(i^2+1)
else:
    SS.<I>=QQ[I]
K.<a1,a2,b1,b2,x>=PolynomialRing(SS)
```

```
c0=numerator(1+x^(k-l-1))
c1=numerator((a1+b1*I)+(a1-b1*I)*x^(m-n-1))
c2=numerator((a2+b2*I)+(a2-b2*I)*x^(p-q-1))
#condicions = singular.facstd(singular.ideal(c0,c1,c2))
print("c0:="+str(c0)+"\n")
print("c1:="+str(c1)+"\n")
print("c2:="+str(c2)+"\n")
```

Listing B.2: Main Sage script for computing and iteratively reducing Lyapunov constants, and printing them in a Maple-friendly format for later processing.

```
\r polops.gp

k=taskArgs[1];
l=taskArgs[2];
m=taskArgs[3];
n=taskArgs[4];
p=taskArgs[5];
q=taskArgs[6];

gentrifield(k,l,m,n,p,q)=
{
    local(v1,v2,v3);
    v1=vector(k+l+1);
    v1[l+1]=1;
    v2=vector(m+n+1);
    v2[n+1]=a1+b1*I;
    v3=vector(p+q+1);
    v3[q+1]=a2+b2*I;
    return(List([v1,v2,v3]));
}

R=gentrifield(k,l,m,n,p,q);
```

Listing B.3: PARI/GP script *lyaps.gp* used to parse `taskArgs` and generate the field

Bibliography

Aranson, S. K., Bronshtein, I. U., Grines, V. Z., & Ilyashenko, Y. (1996). *Dynamical Systems I: Ordinary Differential Equations and Smooth Dynamical Systems* (D. V. Anosov & V. I. Arnold, Eds.). Springer Science & Business Media.

Becker, T. & Weispfenning, V. (1993). *Gröbner Bases*. Graduate Texts in Mathematics. New York, NY: Springer New York.

Christopher, C. (2006). Estimating Limit Cycle Bifurcations from Centers. *Trends in Mathematics: Differential Equations with Symbolic Computation*, 23–35.

Cima, A., Gasull, A., Mañosa, V., & Mañosas, F. (1997). Algebraic properties of the Liapunov and period constants. *Rocky Mountain J. Math., 27*(2), 471–501.

Cima, A., Gasull, A., & Medrado, J. C. (2009). On persistent centers. *Bulletin des Sciences Mathematiques*, *133*(6), 644–657.

Dukarić, M., Giné, J., & Llibre, J. (2016). Reversible nilpotent centers with cubic homogeneous nonlinearities. *Journal of Mathematical Analysis and Applications*, *433*(1), 305–319.

Dumortier, F., Llibre, J., & Artés, J. C. (2006). *Qualitative Theory of Planar Differential Systems*. Springer-Verlag Berlin Heilderberg.

Gasull, A., Giné, J., & Torregrosa, J. (2016). Center problem for systems with two monomial nonlinearities. *Communications on Pure and Applied Analysis*, *15*(2), 577–598.

Gasull, A. & Torregrosa, J. (2001). A new approach to the computation of the Lyapunov constants. *Comput. Appl. Math*, *20*(1999), 1–29.

Giné, J. (2007). On the number of algebraically independent Poincaré-Liapunov constants. *Applied Mathematics and Computation*, *188*(2), 1870–1877.

Giné, J. (2012). Limit cycle bifurcations from a non-degenerate center. *Applied Mathematics and Computation*, *218*(9), 4703–4709.

Greuel, G.-M. & Pfister, G. (2007). *A Singular Introduction to Commutative Algebra* (2nd). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hartman, P. (1960). A lemma in the theory of structural stability of differential equations. *Proceedings of the American Mathematical Society*, *11*(4), 610–610.

Hilbert, D. (1902). Mathematical Problems. *Bulletin of the American Mathematical Society*, *8*(10), 437–480.

Hirsch, M. W., Smale, S., & Devaney, R. L. (2004). *Differential Equations, Dynamical Systems, and an Introduction to Chaos*.

Ilyashenko, Y. & Yakovenko, S. (2006). *Lectures on Analytic Differential Equations*.

Liang, H. & Torregrosa, J. (2016). Weak-Foci of High Order and Cyclicity. *Qualitative Theory of Dynamical Systems*.

Llibre, J., Ramírez, R., & Sadovskaia, N. (2010). On the 16th Hilbert problem for limit cycles on nonsingular algebraic curves. *Journal of Differential Equations*, *250*(2), 983–999.

Qiu, Y. & Yang, J. (2009). On the focus order of planar polynomial differential equations. *Journal of Differential Equations*, *246*(8), 3361–3379.

Songling, S. (1981). A Method of Constructing Cycles without Contact around a Weak Focus. *Journal of Differential Equations*, *41*, 301–312.

Songling, S. (1984). On the Structure of Poincaré-Lyapunov for the Weak Focus of Polynomial. *Journal of Differential Equations*, *52*, 52–57.

Timochouk, L. A. (1994). Focal values for quadratic systems with four real singular points. *Reports of the Faculty of Technical Mathematics and Informatics, Delft University of Technology*, (94-64).

Zoladek, H. (1994). Quadratic Systems with Center and Their Perturbations. *Journal of Differential Equations*, *109*(2), 223–273.