Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Máster en Ingeniería Industrial

An Object Oriented Approach for Robust Optimization of Delayed Systems

Autor:

Morán Río, Diana Patricia

Ángel Martín Martínez

Ruhr Universität Bochum

Valladolid, Noviembre 2016.

TFM REALIZADO EN PROGRAMA DE INTERCAMBIO

TÍTULO: An Object Oriented Approach for Robust Optimization of Delayed Systems

ALUMNO: Diana Patricia Morán Río

FECHA: 28 de Septiembre de 2016

CENTRO: Fakultät für Maschinenbau

TUTOR: Jonas Otten

RESUMEN

La optimización de sistemas en estado estacionario puede resultar en la obtención de óptimos inestables. La adicción de restricciones de estabilidad es suficiente si el sistema es conocido exactamente. En aplicaciones reales los sistemas cuentan con ciertas incertidumbres, por lo que la robustez del sistema debe estudiarse al mismo tiempo que las incertidumbres en los parámetros. El método de los vectores normales puede emplearse para resolver este tipo de problemas, añadiendo restricciones y variables auxiliares. Por ello es necesario estructurar adecuadamente la resolución del problema.

El objetivo del presente Trabajo Final de Máster es el desarrollo de un programa de MATLAB capaz de llevar a cabo la optimización en estado estable de sistemas con retraso. Esto incluye la generación de restricciones de estabilidad con MAPLE, exportar esas restricciones como código C y la inicialización automática de las restricciones. La resolución del presente problema se realizará usando programación orientada a objetos.

PALABRAS CLAVE

Optimización, robustez, programación orientada a objetos, sistemas con retraso, método de vectores normales.

ABSTRACT

Steady state optimization easily leads to an optimum representing an unstable steady state. Adding stability constraints is only su_cient, if the system is exactly known. This is not the case for real applications, thus the stability has to be robust w.r.t. parameter uncertainty. The normal vector method can be used to achieve this robustness by adding some constraints and auxiliary variables to the optimization problem. This decreases the clarity and makes analysis of the result di_cult. It is necessary to structure the optimization/auxiliary variables and the constraints, specially for larger systems.

The objective is to provide a Matlab program for running a complete steady state optimization for delayed systems. This includes the generation of stability constraints with Maple, exporting those constraints as C-code and an automated initialization of those constraints. Constraints and variables will be managed using the programming paradigm of object orientation.

KEY WORDS

Optimization, robustness, object-oriented programming, delayed systems, normal vector method.

RUS Lehrstuhl für Regelungstechnik und Systemtheorie
Fakultät für Maschinenbau
Prof. Dr.-Ing. M. Mönnigmann

RUHR UNIVERSITÄT BOCHUM RUB

Lehrstuhl für Regelungstechnik und Systemtheorie · Universitätsstr. 150 · D-44801 Bochum

An Object Oriented Approach for Robust Optimization of Delayed Systems

Master Thesis

vorgelegt von
Diana Patricia Morán Río
Matr.-Nr.: 108016109309

1. Fachprüfer:	Prof. Dr.-Ing. M. Mönnigmann
2. Fachprüfer:	Dr.-Ing. S. Leonow
Betreuer	Jonas Otten, M.Sc.

Bochum, den 28. September, 2016

Aufgabenstellung

Steady state optimization easily leads to an optimum representing an unstable steady state. Adding stability constraints is only sufficient, if the system is exactly known. This is not the case for real applications, thus the stability has to be robust w.r.t. parameter uncertainty. The normal vector method can be used to achieve this robustness by adding some constraints and auxiliary variables to the optimization problem. This decreases the clarity and makes analysis of the result difficult. It is necessary to structure the optimization/auxiliary variables and the constraints, specially for larger systems.
The objective is to provide a MATLAB program for running a complete steady state optimization for delayed systems. This includes the generation of stability constraints with MAPLE, exporting those constraints as C-code and an automated initialization of those constraints. Constraints and variables will be managed using the programming paradigm of object orientation.

Eidesstattliche Erklärung

Ich versichere, dass ich meine Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfbehörde vorgelegen.

Bochum, den 28. September, 2016

_____

Diana Patricia Morán Río

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In an optimization problem of a system, the search of an optimum can not be carried out without taking into account the stability of the system. This is due to the fact that the optimum of the system can easily lead to unstable states. An example of this is given by Kastsian and Mönnigmann in [2] where the model of a reactor-separator system is studied. This model is defined and optimized in order to maximize the production of a species. The step response of the system, operating at the calculated optimal state becomes oscillating. The dynamics of the evaluated system are studied at the calculated steady state and the characteristic equation of its differential equations is evaluated. A pair of complex roots with a positive real part was found which revealed that the calculated optimal point was indeed unstable. A steady state optimization where the the dynamics of the system are also taken into account in the optimization would be sufficient to ensure the stability if the system was exactly know, however the real systems are defined by parameters that may be a source of uncertainty.

The uncertainty of the parameters of a system is due to the their variability. Very often an exact value of the parameters can not be stated, but an interval in which they vary can be described. The impact of the uncertainty in the steady state optimization is depicted by Elliott and Luyben [1], where the case of a process, whose profit is to be maximized, it is presented. The sole steady-state economic analysis of the process leads to a plant that meets the market needs with the minimum costs. However, its operability and controllability have not been taken into account and the response of the plant varies within large ranges which results in big amounts of final product out of the customer specifications. They also propose a method of optimization in which the controllability of the system is taken into account. The study of a reactor-stripper system shows the parameters which have a positive influence on the controllability of the system. Due to the

variability of the controllability depending on the parameters of the system, it is necessary to conduct a robust optimization in which the inherent uncertainties of the parameters of the problem are considered.

The method which is used to ensure robust stability during the optimization of the problem is the normal vector method, presented by Mönnigmann and Marquardt [4]. This method consists of measuring the distance of a proposed optimum to the closest points of instability along normal vectors to critical manifolds. A critical manifold is a manifold defined by points where the system becomes unstable, that is critical points. The different formulations of the normal vector method according to the bifurcation which is considered were stated by Mönnigmann and Marquardt for systems which can be described by means of ordinary differential equations (ODE). The normal vector method has been applied to delayed systems in order to obtain a robust optimum by Kastsian and Mönnigmann [2] and Otten and Mönnigmann [5] and [6]. These authors had probed the usefulness of the normal vector method applied to delayed systems which include different types of bifurcations.

1.1 Mathematical statement of the problem

The problem consists of finding an optimal steady state of a dynamical system. The optimal steady state may be determined by an economic profit, a physical variable, a ratio or any other parameter of interest. The optimization of the system may drive it to an unstable optimal point of operation [2]. Consequently, the optimization must take the stability properties of the system into account. In this case, in addition to the stability properties of the problem it must be considered that the system which is to be optimized is a delayed system.

The origin of the delays in a system depends on each system. In a feedback control system the presence of delays are due to the time which take the acquisition of the data, the creation of decisions and their execution. The transport of mass and the control of the flow-temperature-composition may cause delays in process control. In biology, delays may arise for instance from maturation times of the individuals in order to reproduce [7]. The differential equations that define the dynamic behaviour of the delayed systems, the delay differential equations or DDE's. can be described as follows

$$\dot{x}(t) = f(x(t), x(t-\tau_1(t)), ..., x(t-\tau_m(t)), \alpha) \tag{1.1}$$

where $x(t) \in \mathbb{R}^{n_x}$ is the state vector, $\alpha \in \mathbb{R}^{n_\alpha}$ the vector of uncertain parameters, τ_k,

$k = 1, ..., m$ are the delays, f is smooth and maps from an open subset of $\mathbb{R}^{n(m+1)} \times \mathbb{R}^{n_\alpha}$, into $\mathbb{R}^n$.

The objective is to find an optimal and stable steady state of the delayed system, which is robust with respect to parameter variations. This optimization problem has the form

$$\min_{\tilde{x}, \alpha^{(0)}} \quad \Phi(\tilde{x}, \alpha^{(0)}) \tag{1.2}$$

$$\text{s.t.} 0 = f(\tilde{x}, \tilde{x}, ..., \tilde{x}, \alpha^{(0)}) \tag{1.3}$$

$$0 \leq h(\tilde{x}, \alpha^{(0)}) \tag{1.4}$$

where $\tilde{x} \in \mathbb{R}^{n_x}$ is the optimal steady state, $\alpha^{(0)} \in \mathbb{R}^{n_\alpha}$ is the optimal point in the parameter space. The function $h : \mathbb{R}^{n_x} \times \mathbb{R}^{n_\alpha} \mapsto \mathbb{R}^{n_h}$ is sufficiently smooth and defines feasibility constraints. The objective function $\Phi : \mathbb{R}^{n_x \times n_\alpha} \mapsto \mathbb{R}$ is also assumed to be sufficiently smooth and models the variable which is minimized [2].

In order to guarantee the stability of the optimal point, additional constraints are included. The stability boundaries define $(n_\alpha - 1)$-dimensional manifolds in the parameter space which can be described by means of the augmented system

$$G(\tilde{x}^{(c)}, \alpha^{(c)}, u^{(c)}) = 0 \tag{1.5}$$

where the upper-index (c) denotes a critical state. The variable $u^{(c)}$ comprises the auxiliary variables of the augmented system. The number of auxiliary variables and their characteristics depend on the bifurcation considered. These aspects are addressed further on when the bifurcations are described. An example of augmented system is given in Sec. 6.1.

The stability of a system is determined by its characteristic equation. the characteristic equation of a delayed system has the form

$$\det\left(\lambda I - J_0 - \sum_{k=1}^{m} J_k e^{-\lambda \tau_k}\right) = 0 \tag{1.6}$$

where $I \in \mathbb{R}^{n_x \times n_x}$ is the identity matrix, J_0 and J_k, $k = 1, ..., m$ are the Jacobian matrices with respect to $x(t)$ and $x(t - \tau_k)$ respectively evaluated at $(\tilde{x}, \alpha^{(0)})$. If the real part of all the roots of the equation (1.6) is negative, i.e., $\text{Re}\{\lambda_j\} < 0$, $j \in \mathbb{N}$, then the considered steady state is stable.

The normal vector method characterizes steady states by their distance to stability boundaries measured along a normal vector in the parameter space. This is depicted in Fig.

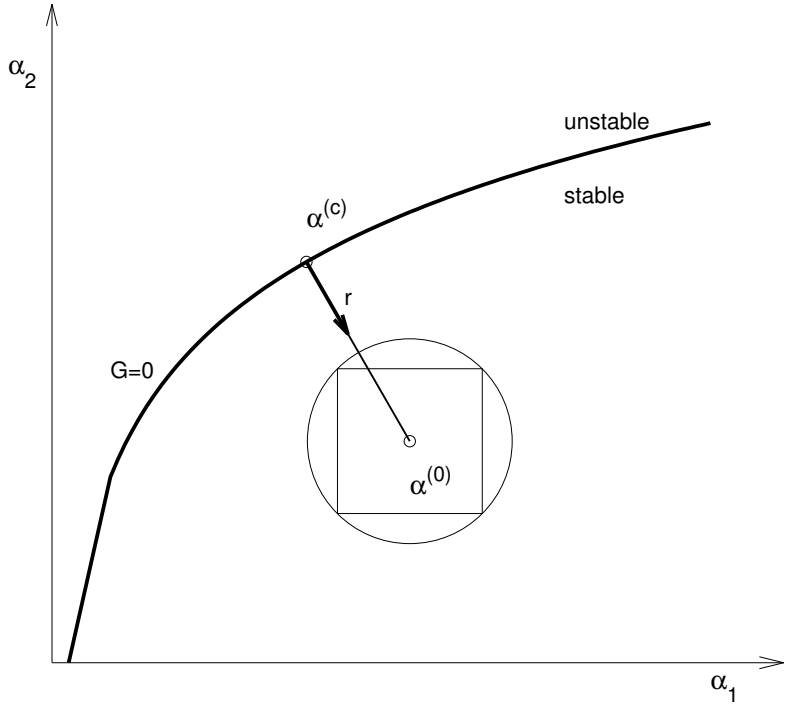


Figure 1.1: Normal vector method. Uncertainty region of $\alpha^{(0)}$, normal vector r and manifold described by $G = 0$.

1.1, where $\alpha^{(0)}$ is an arbitrary point, with normal vector r .The normal vector is defined by a system of nonlinear equations

$$H(\tilde{x}^{(c)}, \alpha^{(c)}, u^{(c)}, q, r) = 0 \tag{1.7}$$

where $\tilde{x}^{(c)}$, $\alpha^{(c)}$ and $u^{(c)}$ are as in (1.14), r is the normal vector at $\alpha^{(c)}$ and q collects the auxiliary variables concerning the normal vector system. In Sec. 6.1 an example of these systems of equations is given.

In general, the parameters of the models are measured by different units, due to this fact the parametric uncertainty must be taken into account in a dimensionless way. In addition, the parameters of the systems which can not be fixed at nominal values vary within ranges, these are described in a dimensionless way

$$\alpha_i \in \left[\alpha_i^{(0)} - 1, \alpha_i^{(0)} + 1\right], i = 1, ..., n_\alpha \tag{1.8}$$

being $\alpha_i^{(0)}$ dimensionless nominal points. The uncertainty of the parameters is result of different physical aspects such as measurement errors, intervals of confidence or variation of the parameters in the time.

The equation (1.8) defines a n_α-dimensional hypercube of side length 2 enclosed by an n_α-

dimensional hyperball of radius $\sqrt{n_\alpha}$. The hyperball whose center is $\alpha^{(0)}$ is the uncertainty region of the point. If the hyperball touches the manifold tangentially, any variation of the parameters within the described hypercube in (1.8) do not lead a crossing of stability boundaries. The stability of the system is guaranteed if the point in the center of the hyperball is at a $\sqrt{n_\alpha}$ distance of the manifold. The constraints

$$\alpha^{(0)} = \alpha^{(c)} + d\frac{r}{\|r\|_2}, \tag{1.9}$$

$$d \geq \sqrt{n_\alpha} \tag{1.10}$$

where $(x^{(0)}, \alpha^{(0)})$ refers to a candidate steady state, $\alpha^{(c)}$ are the critical parameter values and r is the normal vector to the critical manifold at $\alpha^{(c)}$ must hold for some $d \in \mathbb{R}$ [3]. Equation (1.16) is named connection constraint because connects the initial point and a critical point. Inequation (1.17) is named robustness constraint because guaranties that the optimum remains stable in the uncertainty region.

The form of the equations of the manifold and the normal vector system (1.14) and (1.15) depends on the type of instability. A summary of the characteristics of these equations is given below for the bifurcations that are taken into account in the present master thesis. The complete description of these equations is given by Kastsian and Mönnigmann [2] and Otten and Mönnigmann [5] and [6]. Tables 1.1 and 1.2 collect the variables which define each system of equations according to the bifurcation and their number of equations.

- **Fold bifurcation**. A Fold bifurcation point $(\tilde{x}, \alpha)$ is a point at which the delayed system loses its stability due to the existence of a single real eigenvalue λ crossing the imaginary axis, defined λ by the characteristic equation of the system (1.6). The augmented system which describes the Fold bifurcation consists of $2n_x+1$ equations which are defined by the variables $x \in \mathbb{R}^{n_x}, w \in \mathbb{R}^{n_x}$ and $\alpha \in \mathbb{R}^{n_\alpha}$. The normal vector system of the Fold bifurcation consists of $2n_x + n_\alpha + 1$ equations defined by the variables x, w, α and $r \in \mathbb{R}^{n_\alpha}$ [6].

- **Modified Fold bifurcation**. The modified Fold bifurcation is a generalization of the Fold bifurcation in which a decay rate $\sigma < 0$ is defined by means of a critical boundary $\mathrm{Re}\{\lambda\} = \sigma$ [6]. The augmented system which describes the modified Fold bifurcation consists of $2n_x + 1$ equations which are defined by the variables $x \in \mathbb{R}^{n_x}, w \in \mathbb{R}^{n_x}$ and $\alpha \in \mathbb{R}^{n_\alpha}$. The normal vector system of the modified Fold bifurcation consists of $4n_x + 2n_\alpha + 1$ equations defined by the variables x, w, α and $v \in \mathbb{R}^{n_x}, \gamma \in \mathbb{R}, u \in \mathbb{R}^{n_x}$ and $r \in \mathbb{R}^{n_\alpha}$ [6]. The expressions of the augmented system and the normal vector system for this type of bifurcations are stated in Sec. 6.1.

- **Hopf bifurcation**. The instability of the Hopf bifurcations is due to the existence of a pair of complex conjugated eigenvalues on the imaginary axis. The augmented system of this type of bifurcations consists of $3n_x + 2$ described by $x \in \mathbb{R}^{n_x}, w_1 \in \mathbb{R}^{n_x}, w_2 \in \mathbb{R}^{n_x}, \omega \in \mathbb{R}$ and $\alpha \in \mathbb{R}^{n_\alpha}$. The normal vector system of the Hopf bifurcation consists of $6n_x + n_\alpha + 4$ equations defined by the variables $x, w_1, w_2, \omega, \alpha, v_1$ and $v_2 \in \mathbb{R}^{n_x}, \gamma_1 \in \mathbb{R}, \gamma_2 \in \mathbb{R}, u \in \mathbb{R}^{n_x}$ and $r \in \mathbb{R}^{n_\alpha}$ [2].
- **Modified Hopf bifurcation**. The modified Hopf bifurcation is characterized by the existence of a pair of leading complex conjugated eigenvalues with non-zero real part $\mathrm{Re}\{\lambda\} = \sigma$. The augmented system of this type of bifurcations consists of $3n_x + 2$ equations described by $x \in \mathbb{R}^{n_x}, w_1 \in \mathbb{R}^{n_x}, w_2 \in \mathbb{R}^{n_x}, \omega \in \mathbb{R}$ and $\alpha \in \mathbb{R}^{n_\alpha}$. The normal vector system of the modified Hopf bifurcation consists of $6n_x + n_\alpha + 4$ equations defined by the variables $x, w_1, w_2, \omega, \alpha, v_1$ and $v_2 \in \mathbb{R}^{n_x}, \gamma_1 \in \mathbb{R}, \gamma_2 \in \mathbb{R}, u \in \mathbb{R}^{n_x}$ and $r \in \mathbb{R}^{n_\alpha}$ [5].

Table 1.1: Variables and number of equations of the augmented systems.

Bifurcation	Variables	Number of equations
Fold	x, w, α	$2n_x + 1$
Hopf	$x, \omega, w_1, w_2, \alpha$	$3n_x + 2$
Modified Fold	x, w, α	$2n_x + 1$
Modified Hopf	$x, \omega, w_1, w_2, \alpha$	$3n_x + 2$

Table 1.2: Variables and number of equations of the normal vector systems.

Bifurcation	Variables	Number of equations
Fold	x, w, r, α	$2n_x + n_\alpha + 1$
Hopf	$x, \omega, w_1, w_2, v_1, v_2, \gamma_1, \gamma_2, u, r, \alpha$	$6n_x + n_\alpha + 4$
Modified Fold	$x, w, v, \gamma, u, r, \alpha$	$4n_x + 2n_\alpha + 1$
Modified Hopf	$x, \omega, w_1, w_2, v_1, v_2, \gamma_1, \gamma_2, u, r, \alpha$	$6n_x + n_\alpha + 4$

The optimization problem including the normal vector constraints described above reads

as follows:

$$\min_{\tilde{x},\alpha^{(0)}} \Phi(\tilde{x}, \alpha^{(0)}) \tag{1.11}$$

$$\text{s.t.} 0 = f(\tilde{x}, \tilde{x}, ..., \tilde{x}, \alpha^{(0)}) \tag{1.12}$$

$$0 \leq h(\tilde{x}, \alpha^{(0)}) \tag{1.13}$$

$$G_j(\tilde{x}_j^{(c)}, \alpha_j^{(c)}, u_j^{(c)}) = 0 \tag{1.14}$$

$$H_j(\tilde{x}_j^{(c)}, \alpha_j^{(c)}, u_j^{(c)}, q_j, r_j) = 0 \tag{1.15}$$

$$\alpha^{(0)} = \alpha_j^{(c)} + d_j \frac{r_j}{\|r_j\|_2}, \tag{1.16}$$

$$d_j \geq \sqrt{n_\alpha} \tag{1.17}$$

where the sub-index j denotes the manifold.

Among the constraints of the problem can be found equality and inequality constraints. The set of equality constraints consists of the equations that describe the manifold and the normal vector ((1.14) and (1.15)) of each bifurcation, defined according to the different bifurcations, and the connection constraints of each manifold (1.16). The inequality constraints of the problem are the robustness constraints of each manifold (1.17). In appendix 6.1 an example of the equations (1.14) and (1.15) can be found; in this case the modified Fold bifurcation was taken into account.

1.2 The existing numerical implementation of the solution of the problem

The resolution of the optimization problem can be divided in two main actions: the statement of the problem and its resolution. In the statement of the problem the expressions of the cost function (1.11), the steady state constraints (1.12), the feasibility constraints (1.13), the normal vector constraints (1.15), the connection constraints (1.16) and the robustness constraints (1.17) are defined. The resolution of the problem consists of the numerical optimization of the problem and the previous actions that have to be performed in order to carry out the optimization.

The definition of the cost function, the steady state constraints and and the feasibility constraints remains unchanged regardless of the type of bifurcation and is part of the definition of the problem, it is made manually. The normal vector constraints depend on the system and on the bifurcation and their expressions are more complex. These constraints are defined symbolically by MAPLE and then used in the numerical optimization of the

problem. The symbolic definition of the normal vector equations is possible due to the existence of several modules that contain the expressions of the equations according to the bifurcation. The connection and robustness constraints are defined manually due to their simplicity and because their expressions do not depend on the system nor the type of the bifurcations.

In order to define the normal vector constraints the specific MAPLE modules of the bifurcations have to be called. Because the normal vector system depends on the dynamic system, it has to be defined in MAPLE before the modules are called. In the definition of the system, the names of the dynamic variables x, the names of the uncertain parameters α, the expressions of the delay differential equations (1.1), the names of the delayed variables $x_n(t - \tau_k(t))$, the expressions of the delays $\tau_k(t)$ and the values of the known parameters of the system are stated. The generation of the symbolic constraints of a problem involves the writing of a MAPLE code that strongly depends on the system, its variables and bifurcations. The necessary MAPLE commands which are executed can be typed directly in the MAPLE command line or they can be executed in MAPLE by an external program such as MATLAB.

The mathematical statement of the problem is complete when the constraints of each bifurcation are defined. Therefore the symbolic definition of the normal vector constraints has to be repeated for each different bifurcation and the definition of the connection and robustness constraints has to be done for each bifurcation.

The numerical optimization is performed after an initial value is calculated. The initial value of the optimization is calculated as the combination of the initial values of each bifurcation and the initial values of the dynamic variables and the uncertain parameters. The process to obtain the initial value of each bifurcation is as follows [6]:

1. **Find arbitrary critical point**. The term critical point refers to a point on the manifold that is taken into account. The initial critical point is found by solving the equations that define the manifold. The initial point which is used to solve the equations of the manifold is obtained by means of a continuation analysis [3] or with the DDE-BIFTOOL [1].

2. **Find closest critical point**. The closest critical point to the initial point is calculated by minimizing the distance d, in a n_α-dimensional space, between the calculated critical points and the initial point; where n_α is the number of uncertain parameters.

[1] DDE-BIFTOOL is a Matlab package for the numerical analysis of bifurcation and stability of delay differential equations

3. **Find normal vector**. When the normal vector equations are solved for the found closest critical point, the normal vector of that manifold is obtained.

The numerical optimization of the problem is performed entirely by and in MATLAB. After the calculation of the initial value, a solution of the optimization problem taking into account the defined constraints is found.

As it can be seen, the steps that must be performed to carry out all the optimization are not specially complex but they must be performed several times according to the number of bifurcations and its type. The actions performed during the definition of the equations of the manifold (1.14) and the normal vector system (1.15) and the numerical optimization vary from one bifurcation to another due to the different form of the systems according to the bifurcation. Because the systems are defined by a different number of variables, the symbolic definition of them changes and also the optimization due to the different amount of data which has to be handled. Currently the resolution of the optimization problem has the following characteristics:

- MAPLE is used for each different bifurcation.
- The command executed by MAPLE, although similar, include small changes according to the bifurcation.
- The equations of the manifold (1.14) and the normal vector system (1.15) defined by MAPLE have to be integrated in the MATLAB code.
- The connection and robustness constraints, equations (1.16) and (1.17), have the same expression regardless of the bifurcation but they have to be defined to each bifurcation.
- The actions carried out to look for the closest critical points are repeated with small changes according to the bifurcation. These changes are performed manually.
- Most of the changes are due to the different number of variables of the equations which define the stability boundaries and the normal vectors, equations (1.14) and (1.15), according to the bifurcation.

The characteristics mentioned above result in the following facts:

- **Knowledge of Maple syntax** in order to generate the functions that contain the equations of the system.
- **A deep knowledge of the bifurcation type** is necessary when the MAPLE code is written as well as during the numerical steps performed in MATLAB before the actual optimization.

- **Knowledge of the C-files generated by Maple** in order to be used in a MATLAB environment.
- **Repetitive:** Most of the actions which are performed are repeated, with small changes, for each bifurcation.
- **Error prone**: due to the required previous knowledge and the changes that have to be performed according to the type of bifurcation which is being considered, both the generation of the functions of the system and the optimization are error prone.
- **Time consuming programming** due to the fact that all the optimization has to be done manually.

1.3 Introduction to Object Oriented Programming in Matlab

The mentioned characteristics of the resolution of the optimization problem suggest that it may be implemented following an approach which minimizes the possible errors. The way in which data and actions are managed in object oriented programming represents an advantage over procedural programming in this aspect. In object oriented programming the data and the set of actions that can be performed on this data are combined into a logical structure called object, which names this programming approach. The complexity of programming and data structures usually increases with the size of the applications. The use of this approach improves programming in these situations [26]. Even though large functions can be broken into smaller functions and data can be managed from one function to another, the design and management of the data can become more and more difficult and error-prone. Because the actions are associated to certain data in object oriented programming, the general programming can be simplified when large number of functions or programming structures work with special kinds of data.

While in procedural programming data is passed to functions that perform the necessary operations and modify that data, in object oriented programming data is stored in an object and modified by the defined methods of that object [27]. These concepts are discussed below.

Classes

A class describes a set of objects with common characteristics. The values of those characteristics, called properties, are what make an object different from the others of the same class. Objects are specific instances of a class [27]. The basic purpose of a class is to define an object that encapsulates data and the operations performed on that data. All classes have a group of attributes which define the behaviour of the class and cannot be changed by the user. For instance, it can be decided by means of the attributes if a class is abstract, it allows subclasses or if it is listed as a superclass. The attributes can be specified not only for a class but also for its properties, methods and events [9] and [8].

A superclass is a class from which one or more classes derive. The classes that derive from a superclass are called subclasses. The subclasses inherit the properties, methods and events defined by all their specified superclasses [10] and [18].

The definition of superclasses and subclasses is very useful to foster code reuse. For example, if a class to implement an interface to a serial port of a computer is defined, it would be similar to a class for the parallel port. In order to reuse code a superclass with the commonalities of both classes could be implemented and later a subclass for each port derived where the specific characteristics of the port are stated. In this way the subclasses would inherit the common functionality from the superclass [16].

Properties

As mentioned above, a class is defined by its properties and methods as well as its events. The properties collect the data which belongs to the instances of a class and distinguish one instance from the others [12]. The stored data can be a fix set of constant values [13] or can be calculated when required [15]. These aspects of the property are defined by the property attributes. Characteristics such as access, data storage, and visibility of properties can be controlled by setting their attributes [14]. Moreover, properties can trigger an event [11], control the access to a property value or control if its value is saved with the object [17].

Methods

The methods are special functions that perform operations which are usually carried out only on instances of the class [16]. While the class properties contain the data, the class methods operate on that data. As it happens for the properties, the attributes

of the methods define its behaviour. There are five different attributes which can be defined for the methods [20]: Abstract, Access, Hidden, Sealed, and Static. Most of the class and method attributes are common and they share a similar definition although the specific behaviour of the element depends on the specific element, class or method. Taking into account the attributes of the methods, among others, and according to [19] there are seven kinds of methods: ordinary, constructor, destructor,property access, static, conversion and abstract methods; each of them with its own behaviour. For instance, while ordinary methods require the object of the class on which to operate, static methods do not necessarily operate on the class objects. A constructor method must always be defined in order to create the objects of the class. Nevertheless there is no need to define a destructor method to destroy an object, in other words, when the object is deleted, because MATLAB automatically destroys the object.

Except for some kinds of methods [22], generally the class methods can be defined in different files that are separate from the class definition file [21]. This is a clear advantage when working with complex code or class files that contain a large number of methods. Defining the code in different files also fosters code reuse. On the other hand, other functions that are not methods of the class can be defined in the file that contains the class definition. These local functions can be called from anywhere within that file but they remain not visible outside the class definition file [21].

When a class method must behave as an already defined specific MATLAB function, an interesting feature of the object oriented programming is the overloading. Overloading a MATLAB function implies that the name of the class method and the MATLAB function are the same. If a MATLAB function has been overloaded there are at least two definitions of that function, the MATLAB definition and the class definition. The class method is just called when the object belongs to the class that overloaded the function, in any other case MATLAB calls the original MATLAB function [23].

Events and Listeners

Besides the methods and the properties other components of a class are the events and listeners. Events are notices created by the objects when an special situation takes place, such as a property that changes its value or the use of certain function. Listeners run certain routines when an event of interest occurs [24].

In summary, the use of object oriented programming simplifies large structures and reduces the probability of making an error. This is achieved through a different programming structure. Instead of storing data in variables which are passed to the necessary functions that modify that data or build new variables, the data is stored in objects and modified by the specific functions defined for those objects. To define an object, a class is needed. The classes define the behaviour of the objects and they consist of properties, where data is stored; methods that modify that data and events, specific actions which are carried out when a particular action occurs. The behaviour of classes, properties and methods depends on its attributes. The main advantage of carrying all the actions within the objects, and therefore of the object oriented programming, is that the external code can remain unchanged while the specific actions on the data change.

1.4 Objectives

The main objective is to solve the robust optimization of delayed systems by means of the object oriented programming. In order to do that a new class must be created which allows to define the optimization problem and solve it in a object oriented environment. Although all the optimization problems which will be solved with the defined class are of the same kind, a robust optimum of a delayed system must be found, the problems differ from each other in terms of number of dynamic variables and uncertain parameters and number and type of bifurcations. Due to this reason the objects of this class must be defined in a way that allows the definition and resolution of the problems regardless of their specific characteristics. Moreover, it must be also possible to visualize the final result in a way that it can be understood intuitively. In the following points it is stated a description of the mentioned actions which have to be performed:

- **Definition.** In order to define the optimization problem the following parameters must be established: the objective function of the optimization, the steady state constraints of the model and the expressions of the delays. Other important information that have to be established is the number and type of the bifurcations, as well as the delay differential equations and the number of dynamic variables and uncertain parameters of the system.
- **Optimization.** The so called optimization of the problem comprises all the actions that must be performed in order to carry out properly the actual optimization. Among these actions are the definition of the constraints of the problem and the

calculation of an initial value for the optimization. It must be noticed that the complexity of the problem grows with the number of dynamic variables, the number of uncertain parameters and the the number of bifurcations. Despite this fact, the calculation of the initial value and the definition of the constraints of the problem as well as the resolution of the optimization have to remain unchanged so that the user does not have to make any change in the code. The proposed solution will have to be able to adapt to the different optimization problems that can arise.

- **Visualization.** It is also an objective to hand back the results in a way that can be understood intuitively.

This Master Thesis is organized in six chapters including the present chapter, Introduction, and the sixth chapter, Appendix. Chapter 2 gives a description of the defined class which can be used to solve the optimization problem. It is presented how the data is stored in the defined objects of the new class and how the optimization is carried out using the a defined object. The use of the defined class is described in a theoretical way in the Chapter 3; special attention is paid to the necessary conditions that must occur to define or optimize the problem and the avoidable steps of the process. It is also explained here how the results can be visualized. The verification of the proposed class and the results obtained are discussed in the Chapter 4 an the conclusions are presented in the Chapter 5. The Chapter 6 contains the commands used to describe, solve and visualize the results of an optimization problem with the defined class.

Chapter 2

Resolution of the robust optimization problem

The mathematical problem has been described in Sec. 1.1, in the present chapter the implementation of the resolution using object oriented programming will be presented. In order to solve the optimization problem, it has been defined a new class with the necessary properties and methods to store the data of the problem and solve the problem respectively. The methods have been defined in such a manner that the resolution of the problem is performed by calling these methods one after the other, without intermediate steps. Loosely speaking, the actions carried out to solve the problem with the defined class are:

1. **Construction of the object**: the first step it to construct an object of the new class. This object will store all the data of the problem. After the object has been created, all its properties that are not hidden can be visualized; although most of them are empty variables because the problem has not been set out yet. Several objects can be defined, thus more than one optimization problem can be available.

2. **Setting the problem out**: it consists of defining the optimization problem. The number of dynamic variables, parameters, cost function, delay differential equations, manifolds, the essential information used during the optimization and important directories must be indicated.

3. **Generation of the constraints of the problem**: the first time that a problem is defined, the constraints of the specific system must be created. The generation of the constraints is carried out by MAPLE by means of the information supplied by the user on the previous step. The constraints comprise the augmented system and normal vector system equations, equations (1.14) and (1.15).

4. **Resolution of the problem**: with the definition of the problem and the augmented system and normal vector system constraints the problem can be solved. Due to the nature of the problem, it is an optimization problem of a nonlinear function with equality and inequality constraints, it is necessary to use the MATLAB function `fmincon`. The main input arguments of this function are the cost function; an initial value of the solution; the constraints of the solution, linear and non linear; the boundaries of the solution and the optimization options [25]. In this step, these input arguments are created automatically and the problem is solved.
5. **Displaying the results**: after solving the optimization the results are stored in a class property and can be displayed accessing that property or using a defined class method.

The proposed class for the implementation of the solution of the problem using object oriented programming consists of five different methods that perform the actions described above as well as the necessary properties to not only store the data of the problem but also the different variables needed on the resolution. More specifically the class methods define the functions that construct the object, set the problem out, generate the constraints, solve the problem and display the results. In the following sections a detailed description of the five methods and the properties of the class are stated as well as the structure of the proposed class.

2.1 Construct the object: `OptimProb` method

`OptimProb` is the name of the created class and also the constructor method, which is the function that must be called in the first place. By means of this function the optimization object is created. This is actually the only function that appears defined in the `classdef` file.

2.2 Set the problem out: `set` function

The `set` function is used when any parameter of the optimization problem has to be defined. The function not only changes the values of the parameters of the problem but also generates variables that are used by other functions. The use of the `set` function can be eluded if the definition of the particular parameter does not involve the generation of data used by the other functions. In general the parameters of the optimization problem

which have to be stated with the set function are the parameters that define the optimization problem: number of dynamic variables, number of uncertain parameters, delay differential equations, expressions of the delays, number of bifurcations and type. This aspect is discussed for each parameter further on.

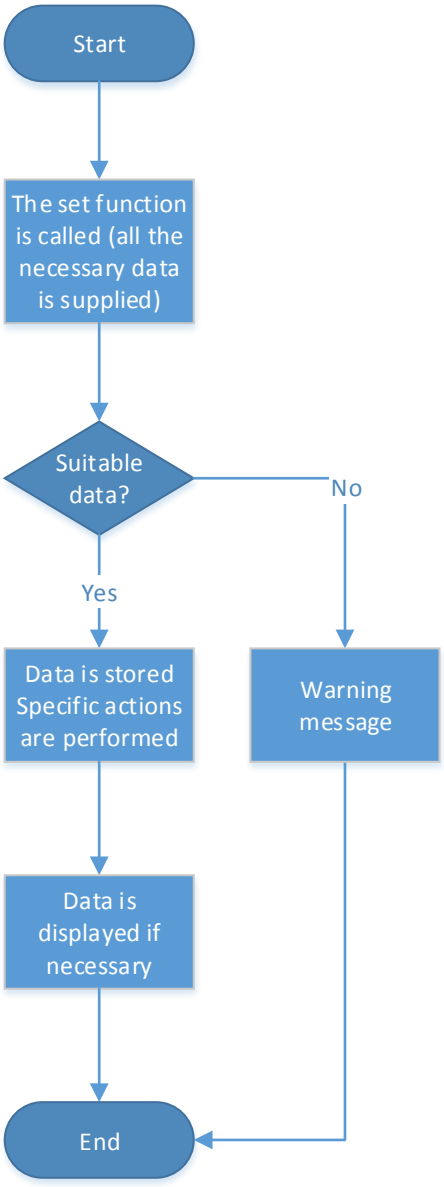
As the function is called, the parameter to change or assign must be stated. In general, there are two different ways to change or assign the value of a parameter depending on how the `set` function is called. The parameters of the problem can be set with the `set` function solely or typing the asked information on the command line after calling the `set` function. All the necessary data, according to the parameter that is being defined, must be supplied to the function when it is called, if the first calling option is chosen. On the other hand, just the name of the parameter is stated when the `set` function is called. The remaining data is requested with a message on the screen and must be supplied via the command line. The definition of each parameter is independent of the rest, thus, both methods can be used to define a problem.

Both, the data supplied when the `set` function is called an the data supplied via the command line must have a specific format that depends on the parameter which is being defined. If the data is supplied via the command line, after calling the set function, the type of data expected is displayed. The type of data has to be known previously if the other definition option is chosen. In order to avoid errors during the definition of the parameters, it is internally established how the data must be supplied and a warning message is displayed if the format is not suitable.

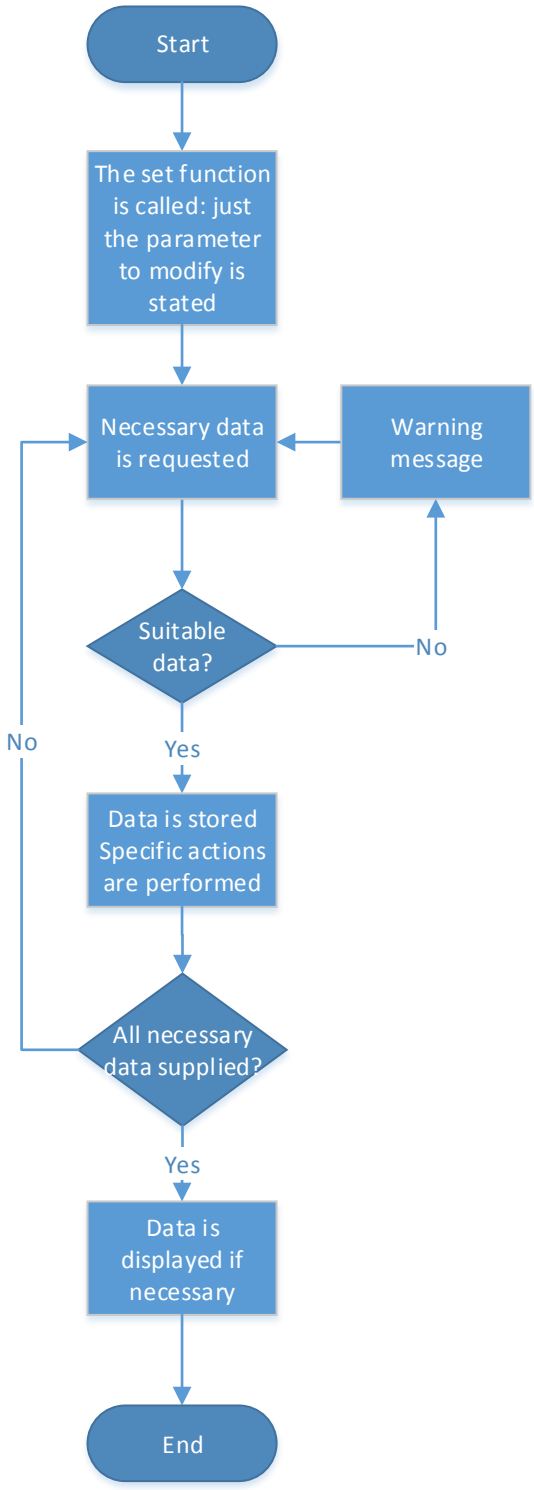
The option in which the command line is not used is the fastest because all the data is supplied directly, however in the second option it is displayed on the screen the data which must be supplied and the type of the variable that contains the data. The use of any of the presented options will depend on the final user of the defined class. In Fig. 2.1 the flowcharts of both options are depicted. Figure 2.1 (a) depicts the process carried out when the arguments of the `set` function contain the information to be defined. In Fig. 2.1 (b) the flowchart of the option in which the data is supplied via the command line is represented.

A brief description of the performed actions during the definition of the fundamental parameters of the problem is given below.

- **Cost function**. The expression of the cost function is stated if this option is specified. Because the cost function does not influence the definition of other parameters of the problem, it can be defined accessing its specific property.
- **Steady state constraints**. The expressions of the steady state constraints are set



(a) Data is supplied as an input argument.



(b) Data is supplied via the command line.

Figure 2.1: Flowchart of the `set` function.

by means of this option. As in the case of the cost function, the property can be modified without using the function.

- **Dynamic variables**. The only parameter needed for the definition of the dynamic variables is the number of them. The `set` function assigns the number of dynamic variables to its class property. As the number of dynamic variables is stated, the set function generates the names of the variables systematically (x1, x2,...). These names are used when the equations are symbolically defined; they are supplied as the names of the variables when the system is defined and the constraints generated in MAPLE. Because these names are used in the symbolical definition of the constraints, they have to be used when the expressions of the delay differential equations are required. There is a possibility of using other names for the dynamic variables and it is defining its names by a set option. These names can be the original names of the variables, facilitating the definition of the problem, specially when there is a large number of involved variables. Certain variables which depend on the number of dynamic variables are also initialized when the number of dynamic variables is available. The definition of the number of dynamic variables can be just carried out through the `set` function because it implies the change of other variables such as the initialization of vectors whose length depends on the number of dynamic variables.

- **Parameters**. All data concerning the parameters, both known and uncertain, is set at the same time. The number of them and their values must be supplied by the user. The values supplied for the uncertain parameters will be treated as their initial values. In this point the names of the parameters are generated. They are used on the definition of the delays and the delay differential equations. Unless other definition of the names of the parameters is given. Because the number of uncertain parameters influences several aspects of the optimizations, its definition or modification has to be done with the `set` function.

- **Delay differential equations**. Through this option the expressions of the delay differential equations (1.1) are assigned to their property. The expressions of the delays are adapted and assigned to a property which is used when the equations (1.14) and (1.15) are generated, for this reason this parameter of the problem cannot be modified by acceding the property.

- **Expressions of the delays**. The number of delays and its expressions are set if this option is specified. The values are assigned to the properties in order to be used in the definition of the equations of the problem by MAPLE. Due to the use of the expressions of the delays in the generation of the equations (1.14) and (1.15) of the problem, this parameter of the problem can just be modified using the `set`

function.

- **Bifurcations**. The number of bifurcations an its type must be indicated in order to define the equations (1.14) and (1.15) accordingly. Therefore they must be defined with the `set` function. Besides, the kind of bifurcation defines the number of variables of the system according to the Tab. 1.2. The number of variables that are involved in each of the bifurcations is stated at this point, when the kinds of the bifurcations are specified. The number of variables per bifurcation is used in the definition of the mentioned equations and in the optimization of the problem. After the number of variables of the augmented system and normal vector system is calculated, some of the vectors which depend on these numbers are initialized.

- **Sigma**. When the value of the parameter σ is needed, it can be assigned with the `set` function; its property can be modified without the use of the `set` function because the value of sigma does not affect the definition of other parameters or internal values.

- **Initial augmented system value**: in order to look for a initial critical point, an initial value of the variables which define the augmented system defined by the equation (1.14) can be supplied. A random initial point will be generated when looking for the initial critical point if no initial point is available. If this initial point is supplied, it has to be done by means of the `set` function.

- **Initial normal vector value**. An initial value of the normal vector can be supplied, otherwise the default value will be used when looking for the initial value of the optimization. The assignment of the value is just preformed by means of the function.

The properties that store the data mentioned above has different behaviour according to the data which is stored in them and the way this data is stated. This aspect is explained in detail in Sec. 2.6.

2.3 Generate the constraints: `gencode` function

The purpose of the `gencode` function is to generate MATLAB callable functions that contain the constraints of the optimization problem. It must be noticed that for each different bifurcation of the problem two different functions will be generated, one of them will contain the augmented system constraints and the other the normal vector system constraints. For instance if a problem has three bifurcations and two of them are the

same, four different functions will be generated, two functions for the augmented system definition and two more functions for the normal vector system definition.

2.3.1 The generation of the functions

The generation of the constraint functions is an avoidable step of the resolution of the optimization problem. When it is not the first time that the solution is calculated, the constraint functions of that specific problem are already available. For this reason, if it is attempted to generate the constraints functions but they already exist, the functions are not generated. This check is made if the problem is completely defined. If the problem were not completely defined the functions would not be generated and a message is displayed.

If the functions of the present problem are not found, the functions of the first bifurcation of the problem are generated. At this point MAPLE is called and by means of the information generated during the definition of the problem, the symbolic definition of the manifold and normal vector constraints is performed. The names of the different variables: state variables, parameters, delay variables, defined when the `set` function was called, as well as the names of the bifurcations and the expressions of the delay differential equations and the delays are used to define the system and generate the manifold and normal vector constraints in the MAPLE symbolic environment. This is made in a automated way because the information supplied by the user was adapted during the definition of the problem.

The functions of all the bifurcations are attempted to be generated, but it is avoided if the functions of a bifurcation already exist. This can be schematically visualized in Fig. 2.2 which shows the verifications performed during the call to the `gencode` function. It must be taken into account that, even though the functions of a previous solution were available, new functions must be generated if any change that could change the constraints expressions has been made. This topic is addressed in the next chapter.

2.3.2 Characteristics of the generated functions

As it has been stated, for each different bifurcation, two different functions will be generated. Those functions comprise the expressions of the different constraints, the augmented system constraints and the normal vector system constraints, equations (1.14) and (1.15) respectively. These functions will be used later, not only on the optimization function, but also when looking for the initial value of the optimization.

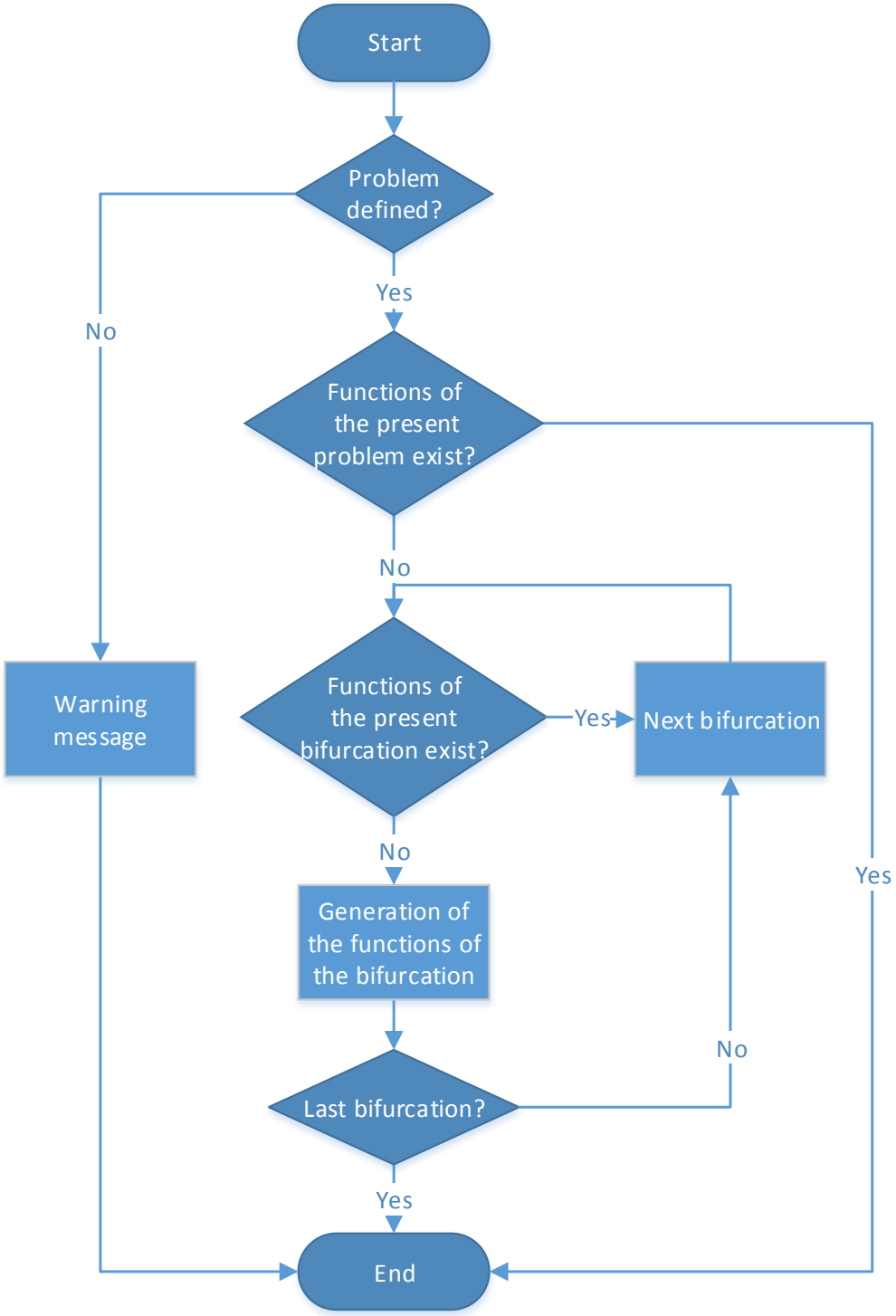


Figure 2.2: Flowchart of the `gencode` function.

In order to call these functions properly their input arguments are defined. In general, the input arguments of both functions are the dynamic variables, the auxiliary variables and the uncertain parameters respectively. However, the number of auxiliary variables varies from one bifurcation to another and from one function to another as it was stated in Tab. 1.1 and Tab. 1.2. In addition to the variables that must be supplied, the type of the expected variables must be taken into account. According to the definition of the variables in Sec. 1.1, the state variables, the uncertain parameters and the auxiliary variables w_1, w_2, v_1, v_2, u and r are vectors and the variables ω, γ_1 and γ_2 are scalars. However, MAPLE handles the state variables and the uncertain parameters as a set of scalars instead of vectors. For this reason, when the generated functions are called, the state variables and the uncertain parameters must be supplied as a set of scalars.

In Tab. 2.1, the dynamic variables, the auxiliary variables and the uncertain parameters of the implemented bifurcations for both cases the augmented and the and the normal vector systems are shortly stated. The order of the auxiliary variables in the table is also the order in which they must be supplied to the functions. It can be noticed that the values of the dynamic variables and the uncertain parameters have to be supplied as a set of scalars, while each of the auxiliary variables corresponds to a single input of the functions.

Table 2.1: Input variables of the generated functions in the same order that they must be supplied to the function.

Bifurcation	Augmented system function	Normal vector system function
Fold	$x_1, ..., x_{n_x}$, $w, \alpha_1, ..., \alpha_{n_\alpha}$	$x_1, ..., x_{n_x}, w$, $r, \alpha_1, ..., \alpha_{n_\alpha}$
Hopf	$x_1, ..., x_{n_x}, \omega$, $w_1, w_2, \alpha_1, ..., \alpha_{n_\alpha}$	$x_1, ..., x_{n_x}, \omega, w_1, w_2, v_1, v_2$, $\gamma_1, \gamma_2, u, r, \alpha_1, ..., \alpha_{n_\alpha}$
Modified Fold	$x_1, ..., x_{n_x}$, $w, \alpha_1, ..., \alpha_{n_\alpha}$	$x_1, ..., x_{n_x}, w, v$, $\gamma, u, r, \alpha_1, ..., \alpha_{n_\alpha}$
Modified Hopf	$x_1, ..., x_{n_x}, \omega$, $w_1, w_2, \alpha_1, ..., \alpha_{n_\alpha}$	$x_1, ..., x_{n_x}, \omega, w_1, w_2, v_1, v_2$, $\gamma_1, \gamma_2, u, r, \alpha_1, ..., \alpha_{n_\alpha}$

In summary, the `gencode` function generates the functions that contain the augmented system and normal vector system constraints of the problem and, due to the structure of the function, it is done if it is exclusively necessary. Among the actions included in the generation of the functions highlights the use of MAPLE to carry out the symbolic definition of the constraints. The generation of the constraint code by MAPLE and the complementary actions performed during the call of the `gencode` function require the complete definition of the system as well as the knowledge of the characteristics of the involved bifurcations, therefore this information has to be available before the call. Finally

the functions that contain the constraints, with the described characteristics, can be found in the directory where the problem is to be solved.

2.4 Solve the optimization problem: `fmincon` function

The actual optimization of the problem is carried out when the `fmincon` function is called. It performs not only the optimization but also other essential actions actions such as finding an initial value for the optimization and the definition of the connection and robustness constraints not defined symbolically by MAPLE and the combination with the constraints already defined with MAPLE. The `fmincon` function performs the numerical analysis of the problem in the MATLAB environment. The main output of the function is a vector that contains the results of the optimization, that is the values of the state variables, the uncertain parameters and the normal vectors that connect the nominal point and the manifolds. In order to perform the optimization, the `fmincon` MATLAB function is used. That means that information about the numerical optimization is also available. Among the information supplied by the `fmincon` function are, in addition to the optimal value, the objective function value, mathematical characteristics of the numerical solution and information of the optimization process [25].

Due to the existence of another MATLAB `fmincon`, this class method overloads the `fmincon` MATLAB function. As stated before, overloading a function consists of defining it at least in two different ways. The overloading of the existing MATLAB function allows to use it with the new specific class that has been defined. Moreover, the new definition of the function simplifies the resolution of the problem by the user, who just have to supply the object of the problem correctly defined. The actions carried out during the call to the `fmincon` function are explained in the following paragraphs and schematically in Fig. 2.3.

Firstly it is verified that the optimization can be carried out. Therefore, the problem must be completely defined and the constraints functions must already be generated before the function is called. Although the generation of the functions already requires the complete definition of the problem, which is performed before the optimization, the verification of the complete definition of the problem is necessary because the generation of the code is an avoidable step. This verification is made considering the values of the properties updated during the definition of the object. The availability of the specific functions of the problem is verified just before the optimization starts. If any of the previous conditions are not fulfilled a short message, specifying the problem, is displayed and the function ends. This step is visualized in Fig. 2.4

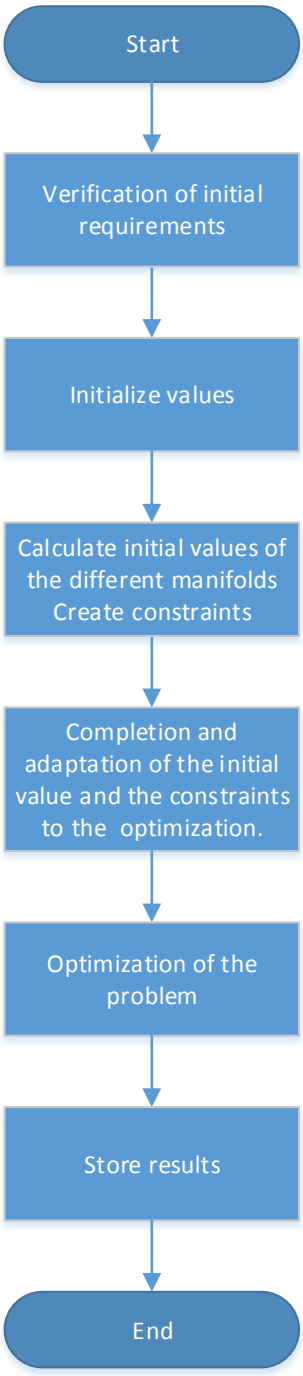


Figure 2.3: Flowchart of the `fmincon` function.

The core of the optimization step is the definition of the remaining constraints, the connection (1.16) and robustness (1.17) constraints, and the calculation of the initial values corresponding to each normal vector. This part of the new `fmincon` function is explained

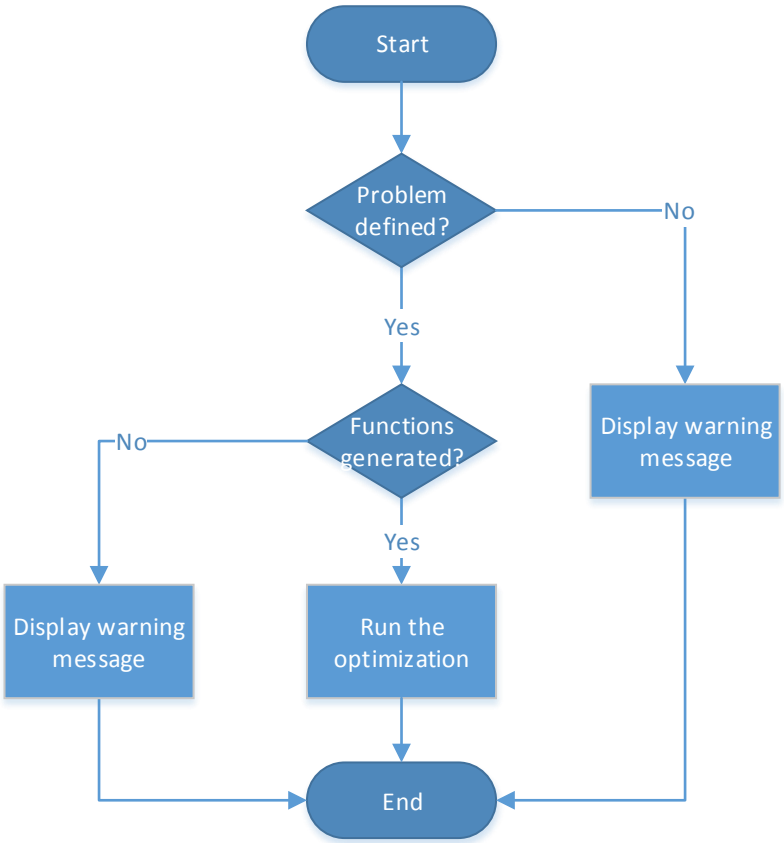


Figure 2.4: Verification of the initial conditions of the `fmincon` function.

later and can be schematically visualize in Fig. 2.5. Once that all the constraints are stated, they are combined and used as an input argument of the `fmincon` function. In addition to the normal vectors initial value, an initial value of the state variables is calculated. The initial values calculated for each normal vector and the initial values of the state variables and the uncertain parameters form the initial value supplied to the `fmincon` function.

The `fmincon` function is called taking into account the feasibility boundaries if they have been defined and as a result the optimal state variables, uncertain parameters and normal vectors are obtained, as well as the mentioned parameters that the MATLAB function provides [25]. Finally the solution of the optimization, that is the optimal state variables, the uncertain parameters and the normal vectors, is stored in a property of the object so it can be available any time after the optimization.

As it can be observed in Fig. 2.5, the definition of the constraints not defined by MAPLE,

the connection constraint and the robustness constraint ((1.16) and (1.17)), and the calculation of the initial values are carried out in a loop. Each iteration of this step corresponds to a different manifold of the problem. Regardless of the number of manifolds or the type of bifurcation, the followed steps to achieve the expected result are always the same. This feature allows to generate the constraints and the initial values automatically and therefore to run the optimization easily just by supplying the object that defines the problem.

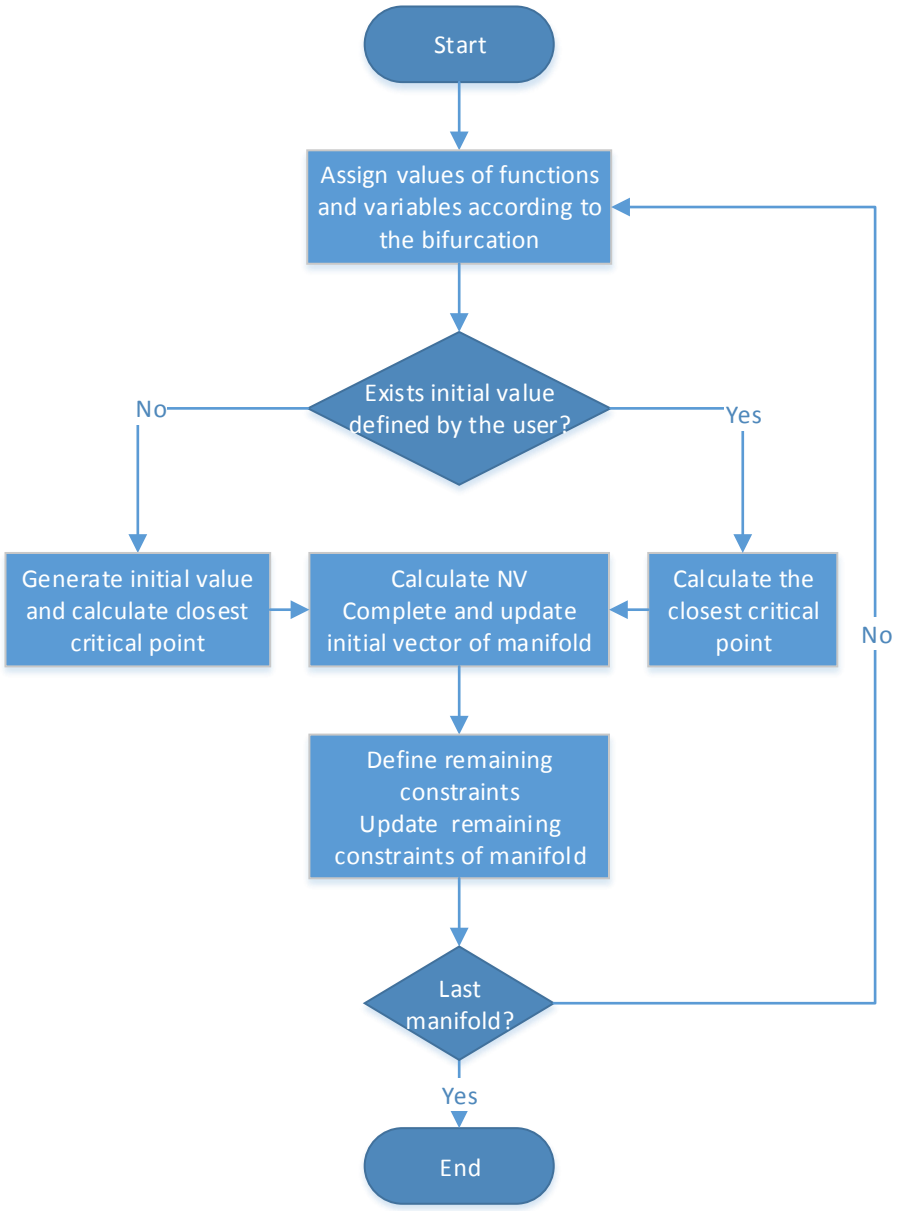


Figure 2.5: Flowchart of the loop inside the `fmincon` function.

During each iteration of the loop, one per manifold, the following actions are carried out. Firstly the functions of the augmented system and the normal vector system constraints are chosen according to the bifurcation among the generated functions by MAPLE. Taking into account the number of manifolds and the type of bifurcation, some variables are defined. Due to the use of these variables the following code can remain unchanged and accordingly it can be run automatically.

The next step is the calculation of the closest critical point to the point defined by the initial value of the uncertain parameters. There are two possible ways to calculate the closest critical point depending on the availability of an initial value defined to this end. If an initial value has been defined previously by the user, the closest critical point is calculated minimizing the distance d between the critical point and the initial point taking as initial value the initial value supplied by the user. When no initial value of the closest critical point is available, it must be generated. The initial value is generated pseudo-randomly by fixing the seed of the random number generator in order to achieve the same results every single time that a problem is optimized. A critical point is found taking into account the generated initial value by solving the equations of the manifold (1.14), contained in one of the generated functions. The closest critical point is calculated afterwards minimizing the distance d as in the previous case. The closest critical points calculated in this way must be compared with the existing critical points in order not to be repeated. This action is carried out to ensure that the critical points belong to different critical manifolds. If a critical point is repeated, two critical points belong to the same manifold an a manifold is not taken into account during the optimization. If a calculated critical point is considered as repeated a new initial point is generated. This process is repeated until a new closest critical point is obtained. The new closest critical points are stored in the property of the initial value in order to be used during later optimizations of the same problem. In this way the process of generating pseudo-random numbers is avoided and consequently the running time reduced. The process of finding a new closest critical point is outlined in Fig. 2.6

The normal vector corresponding to the critical point is calculated and jointly with the closest critical point associated variables stored as the initial value of the manifold. The initial distance between the critical point and the initial point d is also attached.

In this point the augmented system, the normal vector system and the connection constraints, i.e., the equality constraints (1.14), (1.15) and (1.16), are defined. At the end of each iteration, the inequality constraint of the manifold (1.17) is defined and combined with the existing inequality constraints. The equality constraints of the manifold are also added to the existing equality constraints, which in the first iteration are the steady state

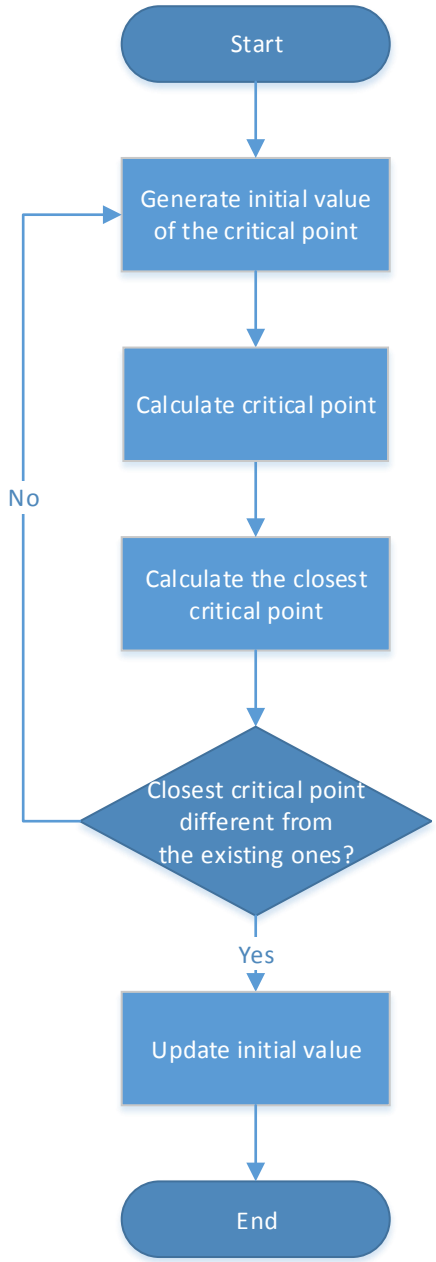


Figure 2.6: Calculation of a new closest critical point by generating the initial point pseudo-randomly.

constraints. The initial value of the iteration is updated with the segment of initial value of the present manifold.

In summary, during each iteration a segment of the initial value, corresponding to the manifold, is calculated, as well as the equality and inequality constraints. When this has

been done for all the manifolds, it exist a vector that contains the initial value except for the initial values of the dynamic variables and the uncertain parameters. In addition, the equality constraints corresponding to all manifolds are available in a variable and the inequality constraints in other one.

After the loop, as it was stated before, the equality constraints and the inequality constraints are concatenated. The initial value of the `fmincon` function is completed with the initial values of the uncertain parameters and the state variables, calculated by means of the steady state constraints. The initial value is saved in a property as well as the solution after it is calculated.

The functions generated by `gencode` are used here, in the `fmincon` function, to describe part of the equality constraints of the problem as well as to find the initial values as described before. In Sec. 2.3 was stated the number and the type of the input arguments of these functions, which must be properly taken into account when they are called. In general the number of arguments and the variables that must be supplied change between types of bifurcations. In order not to change the code between bifurcations, call the functions properly and keep the main code of the optimization as simple as possible, intermediate functions are used. These functions, two per bifurcation, adapt the input arguments supplied to the input arguments that the generated functions expect, that is to say the functions convert the supplied scalars into vectors when it is needed and also places the arguments in the right order. They are defined in the `fmincon` function file.

During the call to the `fmincon` function all the numerical analysis of the problem is performed automatically, both the calculation of the necessary initial values and the optimization of the problem itself. It is done in a cyclical process, repeated for each bifurcation, which allows the automatic resolution of the problem. The final solution of the problem is achieved easily without intermediate steps.

2.5 Visualize the results: `get` function

After the call of the `fmincon` function the results of the optimization are available in two different variables. As it was stated before, after calculating the optimum, it is stored in a property of the object, additionally when the `fmincon` function is called the value of the optimum is an output argument, therefore the external variable corresponding to the output argument contains the optimal value as well. Both the external variable and the property of the object are directly accessible and the value of them are exactly the same. For the purpose of facilitating the interpretation of the results, the `get` function is

defined.

While in the accessible variables, the optimal state, the optimal uncertain parameters and the normal vectors are contained in a single variable that can be unknown by the user, by means of the `get` function those values (the optimal state, the optimal uncertain parameters or any of the normal vectors) are supplied to the user separately as an output parameter. The different results that can be visualized using the `get` function are: the state variables, belonging to a critical point or the nominal point; the uncertain parameters, belonging to a critical point or the nominal point and the normal vector to any manifold. If the state variables or the uncertain parameters of the nominal point are provided by the `get` function those values are also saved in a property of the object, so they are accessible for a further use.

2.6 Properties of the class

Within an object, the data can be stored in a variable or in a class property. The variables are local variables of the methods and therefore they are not accessible from outside the method. When the value of a variable has to be used outside the function where it is defined, its value must be stored in a class property; there is also the possibility of defining the variable as global, but this is not a good practice.

Four different types of information are stored in properties in the defined object. The first type is the data supplied by the user by means of the `set` function. This information must be stored in a property due to its use in the generation of the code and in the optimization. During the definition of the problem, besides the data supplied by the user, certain information about the problem is self-generated in order to be used by the other functions; this information constitutes the second type of information. As it has been stated, the result of the optimization is saved as well in a property in order to be accessible by the user. And finally, the status variables of the object which indicate whether a parameter has been defined or not are also defined as a property. A more detailed explanation of the different data stored and the properties used for that purpose is given below. In addition, the different properties and their kind appear in Tab. 2.2.

The data supplied by the user consists of scalars, vectors, anonymous functions, strings or string arrays and option structures that are stored unchanged or adapted to its use in the specific property. This data remains visible for the user so the problem can be set out again if any mistake is detected. Two different types of properties that store this information can be distinguished: properties whose access is protected and properties

that are accessible by the user. Within the group of properties that define the problem and its access is protected are properties such as the number of state variables, uncertain parameters, manifolds, type of bifurcations or the initial values. The access of these properties is protected because at the same time that they are defined other properties must be defined in accordance to their values. In case that one of these properties had to be redefined, it must be done by means of the `set` function. On the contrary, properties that can be modified by the user without using the `set` function are: the name of the problem, the cost function, the steady state constraints, the boundaries, the options, the paths to the different directories and the value of sigma, as the case may be.

The self-generated information during the definition of the problem comprises the segments of code used in the `gencode` function, the default values of certain properties and the definition of the number of variables per bifurcation. The properties that store the information mentioned in first place remain hidden to the user. On the other hand, the number of variables per bifurcation are visible but inaccessible. Taking into account the number of variables per bifurcation, the optimal state, the optimal uncertain parameters and the normal vectors can be obtained from the property that stores the result of the optimization as explained in Sec. 3.4.

Both the result obtained after the optimization and the partial results stored by the `get` function consist of vectors. The properties that store this information as well as the property that stores the initial value of the optimization are properties that are visible by the user but not accessible for obvious reasons.

The status properties of the object show if the problem has been defined. These properties are defined when the `set` function is called. At the beginning of the `gencode` and `fmincon` functions these properties are used to verify if the problem is completely defined. In addition they are used within the `set` function when certain characteristics of the problem are defined. The status variables are used just internally to know which parameters have been defined and for that reason they remain hidden.

In Tab. 2.2 there is an overview of the different properties classified by their visibility and accessibility according to their use in the class.

2.7 Structure of the class file

The class definition file contains, as its name suggests, the definition of the class. In these files the properties are stated, as well as their attributes and their initial values. Not only the properties are stated in this file, but also the methods, the events and the

Table 2.2: Types of the properties of the object.

	Accessible	Non Accessible
Visible	• Name of the problem • Steady state constraints • Cost function • Sigma value • Lower and upper bounds • Options • Paths to directories	• Number of state variables • Uncertain parameters • Delays • Known parameters • Manifolds • Type of bifurcations • Initial values • Number of variables of the system • Calculated initial value and results
Non Visible		• State properties • Segments of code

enumerations. The present class does not contain any event or enumeration and therefore in the class file only the descriptions of the properties and the methods are contained.

The mentioned properties are defined in three different blocks, each of them with the same attributes of access and visibility. The default values of properties such as the directories are defined at this point, as well as the initial values of the status variables of the object, those which show if the problem is completely defined. On the other hand, all methods are contained in a single block of methods. In this block the only method that is defined completely is the constructor method, that has to be contained in the class definition file [22]. The other methods: the `set`, the `gencode`, the `fmincon`, and the `get` function are defined in separated files. The files that contain the methods are function files that must be contained in the same folder as the class definition file. In addition to be contained in the same folder as the class definition file, the mentioned methods have to be called from the class definition file.

According to the previous paragraph and the earlier sections, before the optimization in the file where the optimization is carried out, can be found the folder of the class, which contains its definition and the definition of the mentioned methods. After the optimization, in the same folder, in addition to the existing files before the optimization, there must be the object and the generated functions.

Chapter 3

Implementation

In the present chapter the implementation details of the resolution of the problem are stated. Every step in the resolution of the problem is explained considering the different available options. That includes how the optimization object must be used and a description of the necessary directories and complementary files that must be available. Within the description of the use of the object it is illustrated how the definition of the optimization object and its properties have to be done, that is the problem; some relevant aspects about the generation of the constraints functions; the optimization of the defined problem and the display and handling of the results. The resolution of the optimization problem presented in [5] is solved by means of the defined class with the code presented in the appendix 6.2.

3.1 Directories

The objective of this section is to state the necessary directories that must be defined before the problem. The referred directories comprise the directories of the MAPLE modules and the directories where the code is generated and stored.

There are two different directories regarding the MAPLE modules that must be supplied to the optimization object, and therefore exist before the object is defined. Those directories are the general directory of the MAPLE modules and the directory of the augmented system that must be contained in the general directory of the MAPLE modules. In addition to the directories supplied to the optimization problem, in the general directory of the MAPLE modules it must exist as well the directory of the auxiliary MAPLE functions. This can be visualized in Fig. 3.1, where the the bold directories are the directories that may be supplied to the `set` function.

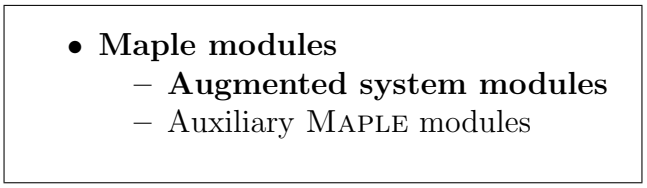


Figure 3.1: Necessary directories of the MAPLE Modules.

Three directories are necessary to generate and store the code: the general directory, the shared working directory and the generated constrains directory. The general directory must contain at least two folders: the shared working directory and the generated constraints directory. In the shared working directory, the constraints of the C-functions generated by MAPLE are stored, in this directory is also located the complementary C-files described in Sec. 3.3. After the three parts of the final functions are joined, the complete C-function of the augmented system constraint and the normal vector constraints are located in this folder. These functions are moved to the directory of the generated constraints and there, they are compiled and the MEX functions generated. The MEX functions are moved to the original working directory where the optimization is performed. This can be visualized in Fig. 3.2, where the the bold directories are the directories that may be supplied to the `set` function.

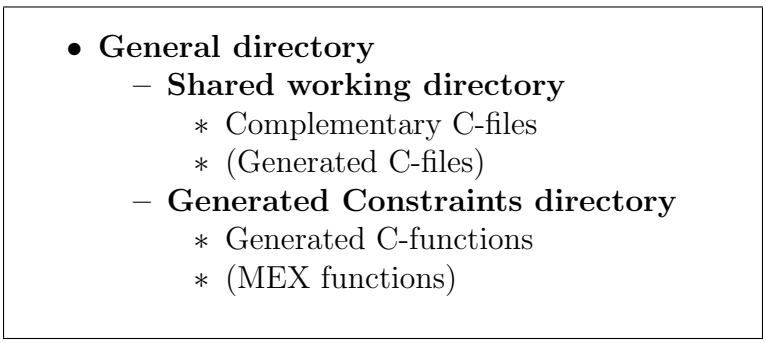


Figure 3.2: Necessary directories for the generated functions.

3.2 How to set the problem

In this section it is explained how the `set` function must be called in order to define the optimization problem properly. Through the following paragraphs different aspects that have to be taken into account are addressed. It is discussed which definitions of parameters can be avoided, in which order have to be defined the parameters and how can be defined the parameters. As it was mentioned in Sec. 2.2, the definition of the parameters of the problem by means of the `set` function, can be done in two different

ways: by supplying all data when the set function is called or via the command line; both options are described for all the different parameters. When it is possible, it will be stated as well how to modify directly the property that stores the data.

Among the seventeen different parameters that can be defined, only seven of them must be defined in order to carry out the optimization or generate the constraint functions. The essential parameters to be defined are: the cost function, the steady state constraints, the dynamic variables, the parameters, the delay differential equations, the expressions of the delays and the bifurcations. If any of the previous parameters were not defined before the generation of the functions or the optimization, these actions could not be carried out. A message would we displayed if an attempt to generate the functions or optimize the problem is made before these parameters are defined.

The definition of the rest of the parameters can be postponed or omitted. The behaviour of the optimization function if any of the optional parameters is omitted is explained for each parameter. Special attention should be paid to the parameter sigma.

As stated in Sec. 2.2, when certain parameters are defined, besides the specific property or properties which store the data of the parameter, other properties may change. If the definition of a parameter involves the use of a property that must be set before, during the definition of another parameter, the second parameter must be defined before the first one. It is described now the order in which the parameters of the problem must be defined. It is not a strict order but the precedence of some definitions over others.

The first definitions that must be done are the definitions of the dynamic variables; the parameters (known and uncertain) and the delays. Once that the dynamic variables are defined the DDEs can be defined. After the definition of the number of parameters and dynamic variables the type of bifurcation can be stated and therefore the number of variables involved in the bifurcation. The knowledge of the number of variables of each bifurcation allows to define the cost function, the steady state constraints and the initial values. The upper and lower boundaries can be defined at the same time as the bifurcations. The names of the variables, if needed, can be defined after the dynamic variables, the parameter and the delays. Figure 3.3 collects this information .

In addition, there are certain parameters that can be stated at any time because neither their definition needs any other information nor the information generated during their definition is used in the definition of other parameters. These parameters are the directories, the name of the problem, the options of the numerical analysis functions and the value of sigma. Although the value of sigma is necessary in the modified Fold and modified Hopf bifurcations, it is not necessary to define it because a default value is given

when it is not defined.

Figure 3.3 summarizes the order in which the different parameter of the optimization problem must be defined. It can be observed in this figure which parameters can be defined at any time and which of them are followed by others or following others. The parameters in bold and with an asterisk have to be defined in all the problems.

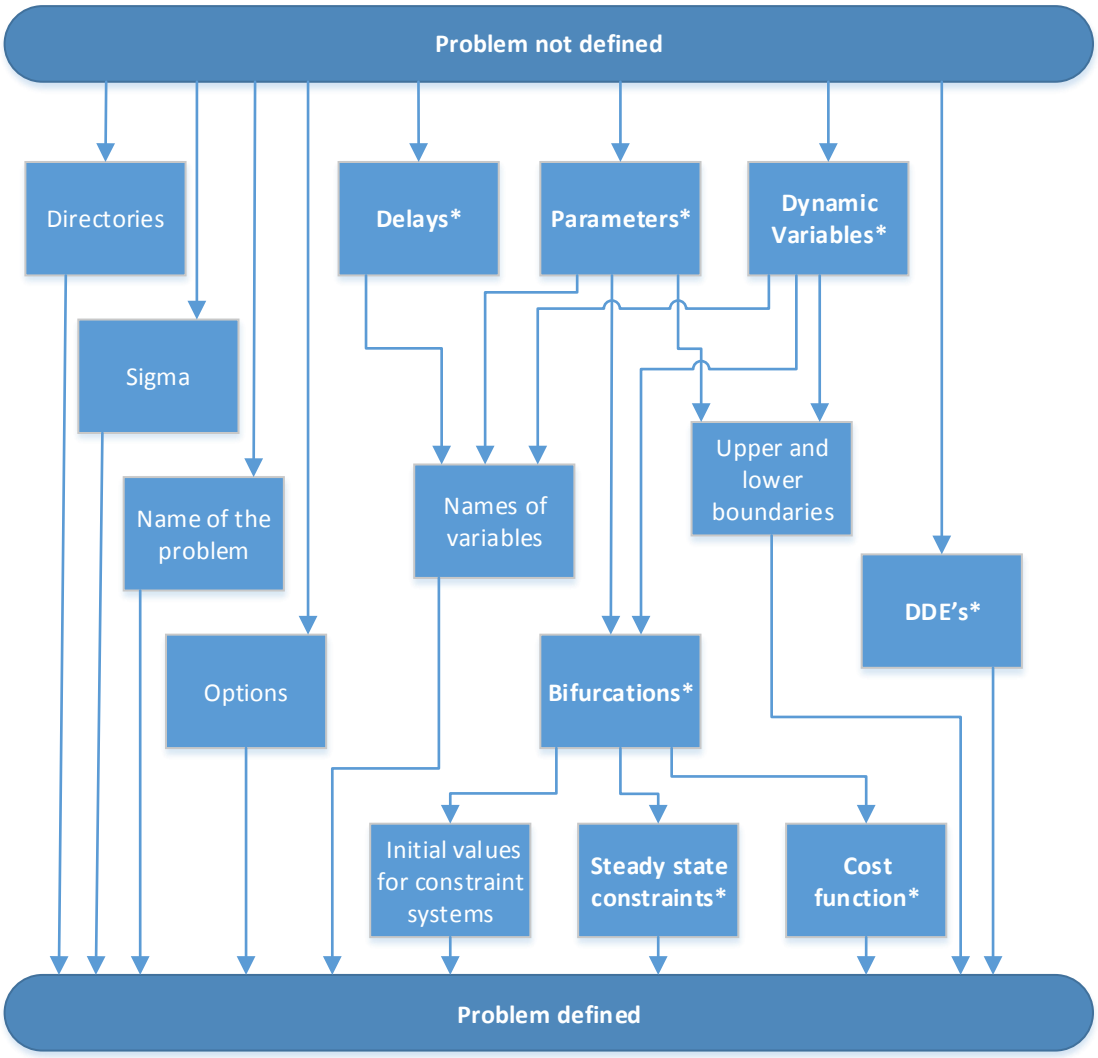


Figure 3.3: Order of precedence of the definition of the different parameters. The parameters in bold and with an asterisk are necessary for the generation of the code and the optimization of the problem.

A description of the definitions of the different parameters is given below:

- **Dynamic variables**. The definition of the number of dynamic variables is an

unavoidable step of the definition of the problem because the number of dynamic variables is used later. For this reason the number of dynamic variables must be stated at the beginning of the definition of the problem. The number of dynamic variables must be supplied to the function via the command line, when asked, or when the `set` function is called. The number of dynamic variables is the only value that the set functions expects and it must be contained in a scalar numerical variable.

- **Parameters**. In order to define the known parameters and the uncertain parameters of the problem the `set` function has to be used. The definition of the parameters is slightly different depending on the use of the `set` function. If all the data concerning the parameters is supplied to the set function when it is called, the data must be supplied in the following way: after stating the object of the optimization and the parameter to set (parameters), the number of known parameters must be supplied in a numerical scalar variable; a vector containing the values of these parameters is the fourth input argument of the function; the fifth and sixth input arguments are the number of uncertain parameters and their initial values respectively, supplied in the same way as the third and fourth arguments.

  In case that the definition of the parameters is done via the command line, the inputs of the `set` function when it is called are only two: the object of the optimization and the parameter to set (parameters). Then a massage that asks for the number of known parameters is displayed in the command line. After the number of known parameters are set, the values of the different known parameters are requested individually through the command line. The number of uncertain parameters and their initial values are requested and supplied in the same way after the known parameters.

- **Expressions of the delays**. The expressions of the delays are defined after the statement of the number of delays. If the expressions are defined using just the `set` function, they have to be contained in a cell array. Each of the arrays contained in the cell array must be the expression of a delay. The `set` function ought to be called with four input arguments: object, name of parameter, number of delays and their expressions, if the delays are defined directly. By means of the command line the different expressions are defined separately after the function asks for them. In this way the number of input arguments of the `set` function is two, since the number of delays is requested after the call.

- **Bifurcations**. In order to define the bifurcations, the number and type of them have to be supplied to the function. The number of them and the names of the

types of bifurcations are supplied to the `set` function as the third and fourth input arguments. The data has to be contained in a scalar and a cell array respectively. If the bifurcations are defined via the command line, firstly the number of bifurcations is provided and then separately the type of each bifurcation. The types of the bifurcations are arrays that must match the defined types of bifurcations defined so far (fold, hopf, modFold and modHopf). The aspects to bear in mind when a new bifurcation is needed are explained in detail in Sec. 3.5.

- **Delay differential equations**. The definition of the delay differential equations is very similar to the definition of the expressions of the delays. The only difference is that the number of delay differential equations has not to be stated because is the number of dynamic variables.

- **Cost function**.The cost function must be set as an anonymous function if it is supplied as an input argument of the `set` function. The indexes of the state variables go from x(1) to x(n) and the indexes of the uncertain parameters go from x(n+1) to x(n+m), where n is the number of state variables and m the number of uncertain parameters. A string that contains the definition of an anonymous function is expected when the cost function is set via the command line. In this case the indexes of the dynamic variables and the uncertain parameters are displayed on the screen when the set function asks for the cost function.

- **Steady state constraints**. The definition of the steady state constraints implies the definition of different anonymous functions. The indexes of the state variables and the uncertain parameters corresponds to the indexes of those variables used to define the cost function. In this case the third input of the `set` function is expected to be a vector of anonymous functions, whereas the data supplied via the command line has to be a cell array which contains the definitions of the different steady state constraints.

- **Initial values for augmented systems**. The initial augmented system values define the initial points used to find the initial critical points. They are, therefore, vectors whose length is defined by the number of variables of the augmented system corresponding to each bifurcation of the problem. If the initial augmented system values are supplied as an argument of the `set` function, a matrix (AxB) containing all the initial values is expected; where A is the number of bifurcations and B the maximum number of variables of all the augmented systems. The number of variables of the augmented system is stored in an visible property. Independent vectors containing the different initial values have to be supplied when the definition is made via the command line. When each of the initial values is requested the length

of the vector is provided as well. To know more about the properties see Sec. 2.6.

If any of the possible initial values is set, it is considered that the whole matrix is initialized and, therefore, the optimization of the problem is carried out without the pseudo-random generation of the critical points. Since the initial values are initialized after defining the types of the bifurcations, the initial values that are not defined will take the default value, a vector of zeros. In contrast to the previous parameters, this one and the following parameters can be omitted from the definition of the problem.

- **Initial normal vector value**. The initial normal vectors are the initial values that would be taken into account to calculate the normal vectors to the closest critical points. In this case the length of the vectors depends on the number of variables of the normal vector system. The number of variables of each normal vector system is stored, as it is the number of variables of the augmented system, in a visible property. The process of defining the initial normal vectors does not differ from the definition of the initial augmented system values except for the length of the vectors. This length is shown when the definition of the initial values is made via the command line.

- **Lower boundary of the variables and parameters**. A lower feasibility boundary of the dynamic variables and uncertain parameters can be assigned. In case no lower feasibility boundary is assigned the default lower feasibility boundary of the solution will be $-\infty$. The lower bound of the dynamic variables and the uncertain parameters consist of two vectors that have to be supplied separately as the forth input argument of the `set` function or via the command line. The third input argument states if it is the boundary of the dynamic variables x or of the uncertain parameters α. As previously mentioned, this is an optional step of the definition of the problem. If any of the boundaries is not defined, the default value $-\infty$ is used. The lower boundaries can be easily modified by acceding the property that stores the vector. This feature applies to all remaining parameters.

- **Upper boundary of the variables and parameters**.An upper feasibility boundary of the dynamic variables and uncertain parameters can be assigned as the lower boundary. In case no upper feasibility boundary is assigned the default value will be ∞. Both bounds can be modified at any time without the use of the `set` function. The definition of the upper boundaries of the dynamic variables and the uncertain values is analogous to the lower boundary

- **Names of the variables**. When the complexity of the problem increases and

the original names of the variables must be used they can be defined by means of this function. The order of the variables must be: state variables, uncertain parameters, known parameters and delay variables. The names of the variables must be contained in a cell array. Internally the actual names of the variables are assigned to the supplied names so that the definition of the equations can be carried out. The definition of the actual names of the variables has to be done using the `set` function.

- **Options of the fsolve function and the Matlab fmincon function**. In case that specific options in the numerical optimization are needed, the options of the `fsolve` and the `fmincon` solvers can be changed, otherwise the default optimization options of the solvers will be used. The properties that store the options of the numerical analysis can be modified through the `set` function or by acceding the specific properties.

- **Sigma**. The value of sigma must be introduced as a scalar either when the function is called or via the command line. A default value of 0 is considered if it is omitted.

- **Directories**. The paths to the different directories where the code is located can be assigned in case the default directories were not right. The use of the function is optional. The paths which can be assigned are:

  - MAPLE modules
  - Augmented system
  - Working directory
  - Shared directory
  - Constraint code

  The five different directories are supplied as strings.

- **Name of the problem**. In order to name the different files generated for each bifurcation a name of the problem can be supplied. Although the whole resolution of the problem can be performed without the definition of any problem name, it is recommendable to define the name of the problem, because when more than one problem is defined the name of the problem will denote if the constraint code of that specific problem is available. In the case the constraint code of the problem is available the generation code step can be avoided and thus the running time improved. The name of the problem can be assigned by acceding its property or by supplying it as a input argument of the `set` function.

Table 3.1: Arguments 3-6 of the `set` function.

Parameter	Input 3	Input 4	Input 5	Input 6
Dynamic variables	Number			
Parameters	Number	Values	Number	Values
Expressions of the delays	Number	Expressions		
Bifurcations	Number	Types		
DDE's	Expressions			
Cost function	Function			
Steady state constraints	Function			
Initial Aug System Value	Initial values			
Initial NV system Value	Initial values			
Lower bound	x/alpha	Boundary		
Upper bound	x/alpha	Boundary		
Names of the variables	Names			
Options `fsolve`	`fsolve` options			
Options `fmincon`	`fmincon` options			
Sigma	Value			
Directories	Name	Path		
Name of the problem	Name			

From the previous paragraphs it can be deducted that the number of input arguments of the `set` function depends on the parameter to be defined. However, the first two input arguments remain unchanged; the first input argument is the name of the object of the optimization and the second one is the parameter to be defined. In general, if the parameter is defined via the command line, those are the only arguments that must be supplied when the `set` function is called, except for some parameters specified before. The table 3.1 summarizes the input arguments of the `set` function which change with the parameter to define.

In summary, the definition of the optimization problem is a flexible process that can be adapted to the known data and the familiarity of the user with the use software.

3.3 Generation of the constraints and optimization of the problem

In the present section some implementation details of the `gencode` and `fmincon` functions are addressed. The necessary conditions that must be met in order to call the `gencode` function are stated as well as the necessity of the generation of the constraint functions when changes in the definition of the problem are made. In addition, the actions that are

performed when the `gencode` function is called are described before some characteristics of the `fmincon` function are explained.

As mentioned in Sec. 2.3, in order to define the constraint functions, the problem must be previously defined. The problem is completely defined if the following parameters are stated: the dynamic variables, the parameters, the delays, the types of the bifurcations, the delay differential equations, the cost function and the steady state constraints. The definition of the cost function and the steady state constraints are not strictly necessary to the generation of the constraint functions but they are required in order to define the whole problem at the same time.

If any of the mentioned parameters change, the expressions of the generated functions change and therefore they must be generated again. In this case, the name of the problem ought to be changed or the old functions deleted; otherwise, the `gencode` function would consider that the functions of the problem already exist and they would not be generated.

If the functions have to be generated the `gencode` function is run until the end. Briefly `gencode` generates for each different bifurcation a MAPLE syntax file that is run; after that, two C-files with the constraints of the augmented system and the normal vector system are generated and later modified; lastly the C-files are compiled separately into callable from MATLAB files. A more detailed explanation of the actions carried out during this function is given below and schematically in Fig. 3.5.

The MAPLE syntax file can be divided in the five different parts according to their purpose, these parts are presented now an can be visualized in 3.4.

- **Header**: in this part the paths to the MAPLE modules that will be used further on are defined.
- **Definition of the system**: the definition of the system consist of setting the names of the dynamic variables of the system, the names of the uncertain parameters and their initial values, the definition of the expressions of the delay differential equations, the setting of the names of the delayed variables and their expressions and lastly the definition of the known parameters and their values. This is possible due to the information provided and generated during the definition of the problem.
- **Creation of the system**: a normal vector system is defined by means of the existing MAPLE modules. As said in the description of the bifurcations in Sec. 1.1, the normal vector system depends on the system itself and on the type of bifurcation, for this reason MAPLE has to be called as many times as different bifurcations exist in the problem. The specific system of each bifurcation is created when its MAPLE

```
# initialization
 restart;
#define paths
 _ModulesDirectory:="/home/users/dmo/Dokumente/MapleModules":
#load modules
 read(cat(_ModulesDirectory, "/Aux/Aux.mpl")):
 read(cat(_ModulesDirectory, "/AugSys2/AugSys2.mpl")):
# define system
 Sys["DynVars"]:=[x1, x2];
 Sys["Parameters"]:=[ alpha1=5, alpha2=2];
 Sys["AEs"]:=[];
 Sys["ODEs"]:=[ `x1'` = alpha1*x2  -alpha2*x1 -alpha1*exp(-
alpha2*tau[1])*x2tau1, `x2'` = alpha1*exp(-alpha2*tau[1])*x2tau1-
p1*x2^2];
 Sys["DelVars"]:=[x1tau1, x2tau1];
 Sys["AlgVars"]:=[];
 Sys["ExplicitAEs"]:=[ p1=1, p2=1, p3=0.5];
 Sys["Delays"]:=[ tau[1]=p2+p3*(1-exp(-x2))];
# insert fixed parameters
Sys:=Aux:-SystemClasses:-subsExplicitAEsIntoDAESys(Sys);
 # look for errors
  Aux:-SystemClasses:-listOfErrorsInDDESys(Sys, strict);
 # define normal vector system
  AugSys:=AugSys2:-SdDelayBif:-ModFoldNV:-
CreateModFoldNVSys(Sys,[ alpha1, alpha2],-0.1):-getSys();
# pick relevant equations of Normal Vectors System
 manifoldEq:=[]:
 for i from 1 by 1 to 5 do
 manifoldEq:=[op(manifoldEq),(rhs(AugSys["Equations"][i]))]:
 end do:
# create frame for code generation
 Procedure4CodeGen:=proc(x1, x2,w, alpha1, alpha2 )
 m;
 end proc;
 ManifoldEquation:=subs([m=manifoldEq],eval(Procedure4CodeGen));
# generate C code
 CodeGeneration:-
C(ManifoldEquation,returnvariablename="residuum",defaulttype=nume
ric,output="Manifold.c",deducetypes=false);
# pick relevant equations of Normal Vectors System
 manifoldEq:=[]:
 for i from 1 by 1 to nops(AugSys["Equations"]) do
 manifoldEq:=[op(manifoldEq),(rhs(AugSys["Equations"][i]))]:
 end do:
# create frame for code generation
 Procedure4CodeGen:=proc(x1, x2,w,v,g1,u,r, alpha1, alpha2)
 m;
 end proc;
 ManifoldEquation:=subs([m=manifoldEq],eval(Procedure4CodeGen));
# generate C code
 CodeGeneration:-
C(ManifoldEquation,returnvariablename="residuum",defaulttype=nume
ric,output="NV.c",deducetypes=false);
```

Figure 3.4: Structure of the MAPLE file. Orange box: header, blue box: definition of the system, purple box: creation of the system, green box: generation of manifold code, red box: generation of normalvector code.

modules are called.

- **Generation of the manifold code**: the code called to generate the code of the manifold takes into account the number of equations that define the augmented system and the auxiliary variables involved, these aspects are described in Tab. 1.1 included in Sec. 1.1. After this part of the code is run, a C-file which contains the definition of the equations of the manifold is available.
- **Generation of the normal vector system code**: the result of this part of the MAPLE code is a C-file which is equivalent to the previous C-file described. This file contains the equations of the normal vector system. In order to generate it, the auxiliary variables of the normal vector system are required, as the auxiliary variables of the augmented system in the file described before. These variables can be found in Tab. 1.2 included in Sec. 1.1.

The newly generated C-files, two per bifurcation, which contain the equations of the augmented system and the normal vector system, are adapted to be called and used by MATLAB. This consists of complementing the generated C-files with the necessary code which enables to evaluate the functions of the normal vector and augmented system taking into account the values of the involved variables. The final C-files are compiled and converted to MEX functions in order to be called by Matlab. These functions have as input arguments the values of the different variables and as output arguments the result of the evaluation of the functions which describe the specific system.

The previously mentioned C-files which are compiled into callable from MATLAB files, two of them per bifurcation, consist of three different parts generated separately. The first part is common in both the augmented system constraints and the normal vector system constraints, it contains necessary library to compile the code. The second part of the C-files is generated by MAPLE automatically and it contains the definition of the constraints (augmented system or normal vector system). Lastly, the third part of the file verifies if the number of arguments of the function and the number of arguments supplied when it is called coincide. Figure 3.6 represents an example of this type of files.

In addition to be generated separately, the three parts of the C-files are created in three different ways. Because the first part of the C-files does not change, it is stored in the shared working directory. The second part is generated automatically by MAPLE after the specific code for that problem an that bifurcation is run, the MAPLE file described before. The last part of the C-files is generated by MATLAB directly. The purpose of this part of the code is to verify that the number of input arguments supplied when the function is called is right; it also assigns the supplied values to their variables and call

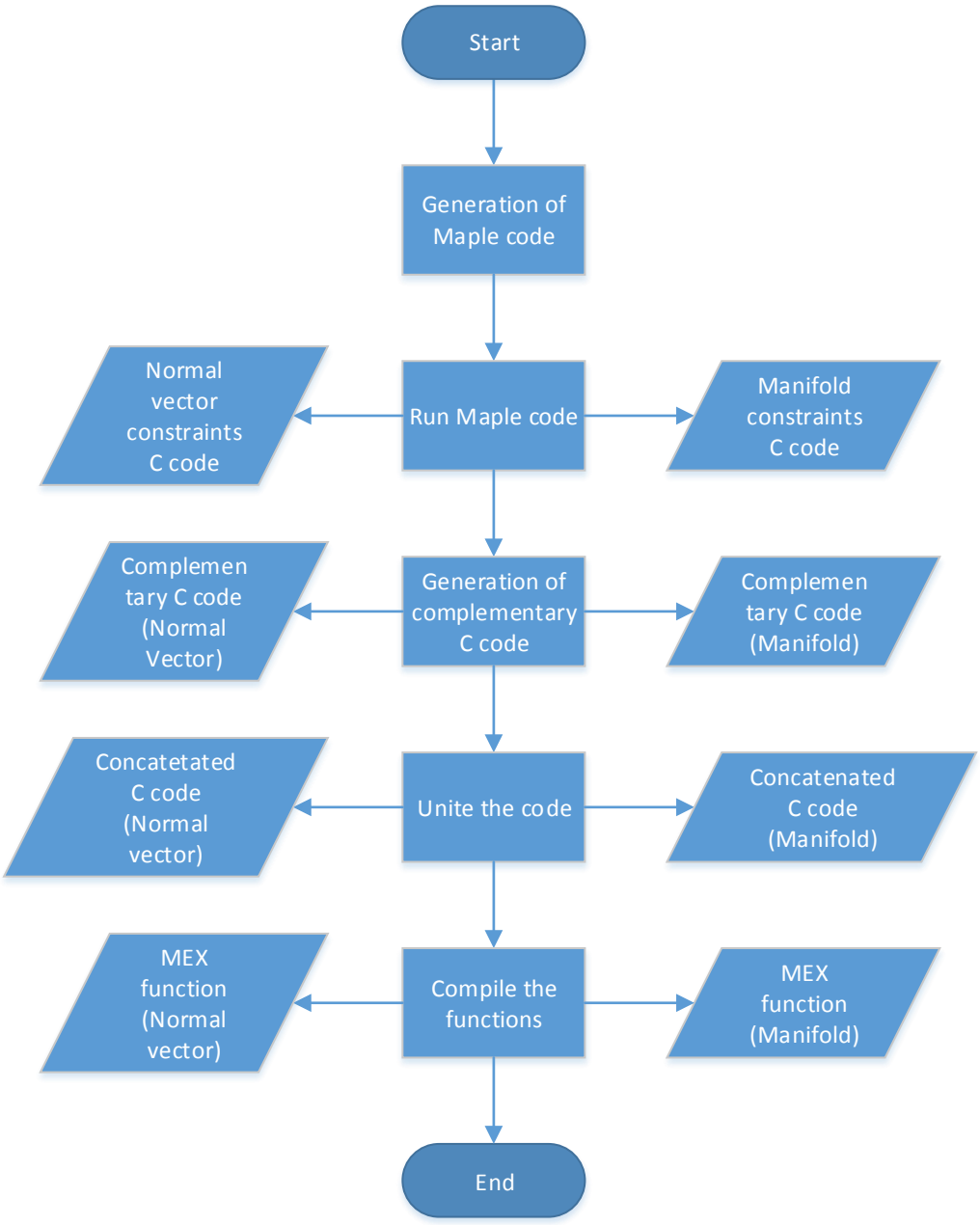


Figure 3.5: Flowchart of the generation of the MEX files.

the computational routine. This requires the knowledge of certain information concerning the system, such as: number and name of dynamic variables, parameters and auxiliary variables and number of equations of the system.

The three parts of the files are put together and compiled into binary MEX files in order to be callable from MATLAB. Once the MEX files are generated, they are moved to the

```
#include "mex.h"
```

```
#include <math.h>
void ManifoldEquation (
  double x1,
  double x2,
  double *w1,
  double alpha1,
  double alpha2,
  double residuum[5])
{
  residuum[0] = alpha1 * x2 - alpha2 * x1 - alpha1 * exp(-alpha2 * (0.15e1
- 0.5e0 * exp(-x2))) * x2;
  residuum[1] = alpha1 * exp(-alpha2 * (0.15e1 - 0.5e0 * exp(-x2))) * x2 -
x2 * x2;
  residuum[2] = -alpha2 * w1[0];
  residuum[3] = (alpha1 + 0.5e0 * alpha1 * alpha2 * exp(-x2) * exp(-alpha2
* (0.15e1 - 0.5e0 * exp(-x2))) * x2) * w1[0] + (-0.5e0 * alpha1 * alpha2 *
exp(-x2) * exp(-alpha2 * (0.15e1 - 0.5e0 * exp(-x2))) * x2 - 0.2e1 * x2) *
w1[1] - alpha1 * exp(-alpha2 * (0.15e1 - 0.5e0 * exp(-x2))) * w1[0] +
alpha1 * exp(-alpha2 * (0.15e1 - 0.5e0 * exp(-x2))) * w1[1];
  residuum[4] = pow(w1[0], 0.2e1) + pow(w1[1], 0.2e1) - 0.1e1;
}
```

```
/* The gateway function */
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 /* check for proper number of arguments */
 if(nrhs!=5) {
     mexErrMsgIdAndTxt("MyToolbox:popfoldFoldManiPop:nrhs","5 inputs
required (some of them are vectors).");
 }
 if(nlhs==0) {
     mexErrMsgIdAndTxt("MyToolbox:popfoldFoldManiPop:nlhs","Please define
an output!");
}
 if(nlhs!=1) {
     mexErrMsgIdAndTxt("MyToolbox:popfoldFoldManiPop:nlhs","One output
required.");
}
 /* get the values of the inputs*/
 double x1 = mxGetScalar(prhs[0]);
 double x2 = mxGetScalar(prhs[1]);
 double *wPointer = mxGetPr(prhs[2]);
 double alpha1 = mxGetScalar(prhs[3]);
 double alpha2 = mxGetScalar(prhs[4]);
 /* create the output matrix */
 plhs[0] = mxCreateDoubleMatrix(1,(mwSize)5,mxREAL);
 /* get a pointer to the real data in the output matrix */
 double *residuumPointer = mxGetPr(plhs[0]);
 /* call the computational routine */
 ManifoldEquation(x1, x2,wPointer, alpha1, alpha2,residuumPointer);
}
```

Figure 3.6: Structure of the C-files. Purple box: libraries, blue box: constraint code, orange box: code for the call of the function.

working directory and an attempt to generate the functions of the next bifurcation is done. As said before, the generation of the functions is avoided if the functions for that bifurcation have been already generated.

If the previous conditions stated above are met, the generation of the functions can be carried out, this is, the `gencode` function can be called. In order to call the `gencode` function and generate the code the only input argument that must be supplied is the name of the object that contains the optimization problem. While MAPLE is generating the constraints, the related code is displayed as well as the possible errors. At the end the generated functions are moved to the directory where the optimization is carried out.

As it happens at the generation of the constraints, the problem must be completely defined before its optimization. No more parameters than the previously mentioned are required in order to optimize the problem. A warning message is displayed and the optimization stopped if any of the required parameters has not been defined yet.

During the call of the `fmincon` function the MEX functions generated by `gencode` function must be contained in the folder where the optimization object is defined. To avoid future errors, it is verified that these functions are in the mentioned file. If the functions are not found on the folder, a message that suggests the generation of the code is displayed.

After fulfilling these two conditions, the `fmincon` function can be called. The only input argument that must be supplied to the function is the name of the object that contains the definition of the problem. If no initial points to look for the initial critical points were defined before the optimization, initial points are generated to this end as described in Sec. 2.4. The use of pseudo-random numbers to generate the initial points is announced during the optimization of the problem. The random numbers that led to the initial critical points are displayed after their calculation and the closest critical points are stored as the initial points in order to be used in a further optimization and avoid the generation of random numbers.

3.4 Visualize the results

The results achieved after the optimization are stored in a property of the optimization object so they can be visualize at any time after the optimization. In this section it is explained what is actually stored in the property and how the data can be obtained from the property.

The variable that is stored in the property of the optimization problem after the resolution

is the vector that contains the result obtained by the MATLAB function `fmincon`. The values stored in this vector comprise the values of the dynamic variables and uncertain parameters at the optimal point, as well as the values of the variables that define the normal vectors to the different manifolds. The variables that define the normal vectors to each manifold are the variables which were stated in Tab. 1.2: dynamic variables and uncertain parameters of the critical points, the auxiliary variables of the augmented and the normal vector system and the normal vector, and the distance d, described in (1.16) and (1.17), between the critical point and the critical manifold.

The specific variables contained in the solution vector, and therefore in the property, and their exact position in the vector are described now. The vector of the solution can be divided into two different parts: the first part, that describes the optimal point and the second part, that describes the normal vectors. The first part can also be split itself into two parts: the values of the dynamic variables at the very beginning and the values of the uncertain parameters. On the other hand the second part of the vector of the solution can be split into as many parts as bifurcations has the problem. Each of they contain the values of the variables stated in Tab. 3.2 referred to a critical point of a manifold or its normal vector.

Table 3.2: Variables that define a normal vector and their order in the vector of the solution.

Bifurcation	Variables
Fold	x, α, w, r, d
Hopf	$x, \alpha, \Omega, w_1, w_2, v_1, v_2, \gamma_1, \gamma_2, u, r, d$
Modified Fold	$x, \alpha, w, v, g, u, r, d$
Modified Hopf	$x, \alpha, \Omega, w_1, w_2, v_1, v_2, \gamma_1, \gamma_2, u, r, d$

Besides the solution vector, as an output parameter of the defined `fmincon` function can be found the output parameters of the MATLAB function `fmincon` [25]: the cost function value at the solution, the reason `fmincon` stopped, information about the optimization process, Lagrange multipliers at the solution, gradient at the solution and the approximate Hessian.

The different variables stored in the solution vector, and in the property of the solution, can be obtained taking into account the order that has been described above and the number of variables per bifurcation which is stored in a accessible property as well. It is also possible to access to some of these values by means of the `get` function; how is it done is explained below.

The `get` function is a single-output function that supplies the value of the specified variable in its input arguments. The possible variables to be shown are: the dynamic variables at

the optimum and at any of the critical points, the uncertain parameters at the optimum and at any of the critical points and the different normal vectors. These variables are specified in the input parameters of the `get` function as it follows.

- The first input argument of the `get` function is the object of the optimization.
- The second parameter is the name of the variables: dynamic variables, uncertain parameters or normal vector.
- The third input argument for the dynamic variables and uncertain parameters discerns between optimal values and critical values. The third input argument for the normal vector indicates the manifold.
- The forth input argument is used to indicate the manifold of the critical point when the dynamic variables or the uncertain parameters have to be shown.

If the user accesses the dynamic variables or the uncertain parameters of the optimal point, the values of these variables are stored in properties so these values can be used later.

3.5 Definition of new types of bifurcations

Up to this point, in this chapter, it has been explained how the optimization problem can be carried out: the definition of the problem, the generation of the constraints, the resolution of the problem and the visualization of the results. However only four different possible bifurcations have been considered in the resolution of the problem. In this section, the changes that must be performed in the code in order to include more bifurcations are explained. It must be noticed that, due to the structure of the existing code, the addition of a new bifurcation to be considered does not change the process of the resolution but adds more information which the program itself must take into account during the resolution. That means that the programmer in charge of the inclusion of new bifurcations does not need to modify the general operation of the code, nor even know how the program works. And for the final user the problem is solved in the same way but it includes more applications.

The changes that must be performed in the code in order to add a new bifurcation consist of the inclusion of a new option to be taken into account when the existing bifurcations are considered. When a new bifurcation is described, the variables that change from one bifurcation to another are stated. As said in Sec. 2.3, according to the bifurcation, the generation of the code changes, as well as the resolution of the optimization. For

this reason specific changes have to be done in the `gencode` and the `fmincon` function. However, the `gencode` and the `fmincon` function are not the only functions that must be modified in order to include the new bifurcation; the `set` function is to be modified because the number of variables of each bifurcation, which is used by the `gencode` and the `fmincon` function, is stated then. In addition to the definition of the changing variables, the new bifurcation has to be taken into account when verifications of the existence of the code are carried out. It is explained below the necessary changes that must be performed in these three functions if a new bifurcation has to be taken into account.

The `set` function is where less changes have to be performed. As stated before, the new definition of the number of variables of the augmented system and the normal vector system must be included. That is the only change in the `set` function when a new bifurcation is included. On the opposite side it is the `gencode` function, where several variables have to be defined according to the new bifurcation. Among these variables are the number of equations of the augmented and normal vector system generated by MAPLE, the inputs that certain procedures of MAPLE require, the names of the specific MAPLE modules, the definition of the auxiliary variables of the bifurcation and the names of the files generated. In the `fmincon` function it is necessary to define the names of the functions to be called during the optimization, this is the functions of the constraints.

The names of the functions of the new bifurcation have to be defined in the `fmincon` function. In addition to them, the functions have to be created as well. The mentioned functions are the functions described at the end of Sec. 2.4, which call the functions generated by MAPLE with the proper arguments, taking into account order, number and type. And finally, It has not been mentioned yet but it is obvious that the MAPLE modules of the new bifurcation have to be defined and contained in the supplied directories.

Chapter 4

Verification of the class and limitations

This chapter focuses on the validation of the defined class and the obtained results. The process of the validation has been affected by the way in which the class was created. The different bifurcations that can be taken into account during the optimization were programmed one after the other, therefore the objective of the validation of the class has consisted, loosely speaking, of verifying every moment that different problems with different kinds of bifurcations can be solved properly.

4.1 Verification of the class

The verification of a program is an iterative process which is performed as the different steps of the problem to be solved are programmed. The possible errors or malfunctions of the program are found easily if the verification is made as the program is written. In the process of verification of the present class, the different methods or functions described have been verified individually and finally jointly.

4.1.1 Verification of the `set` function

The verification of the `set` function has consisted of checking if the data provided by the user was stored properly and if the data generated by the set function and used by other functions had the right format or syntax according to its future use.

As mentioned in Sec. 2.2 the optimization problem can be defined in two different ways

using the `set` function: supplying the data via the command line or by means of the function solely. Regardless of the option which has been chosen, the values of the properties which store the data used by other functions or methods must be the same. In order to verify the `set` function, the definition of each of the possible parameters of the optimization has been checked taking into account the two possible options.

After programming the definition of one of the parameters of the problem, the `set` function was called and the programmed parameter specified. Different types of values were supplied, including those which are not supposed to be supplied, by this it was checked if the class can differentiate the supplied data and display warning messages when the supplied data does not fulfil the expected formats or values.

The format and syntax of the properties which are modified while a new parameter is defined were checked for each parameter of the problem. An special effort was made to ensure that the properties which are used when MAPLE is called to define part of the equality constraints of the problem had the MAPLE syntax expected by the `gencode` function.

4.1.2 Validation of the `gencode` function

Unlike the `set` function, the `gencode` function does not consist of different parts or options. The verification of the definition of the equality constraints by MAPLE has been performed following the next steps:

1. **Verification of the Maple code.** Firstly the MAPLE syntax code which is run in order to define the equations of the manifold and the normal vector system was written. The code was written taking into account the directories of the modules and the necessary input arguments of each module. The validation of this code was made by running it in MAPLE. After obtaining no errors during the execution, the MAPLE code was considered to be verified.

2. **Verification of the complementary C-code.** The verification of the complementary C-code which is part of the final C-files was made by checking the syntax of the generated code after its automatic generation.

3. **Compilation of the C-files.** The compilation of the final C-files also helped with the validation of the code showing possible errors. After correcting the errors which appeared, the `gencode` function could generate the manifold and normal vector system equations properly.

4. **Validation of the `gencode` function together with the `set` function.** Finally the data which defines the problem was substituted in the MAPLE and C-code files, validated in the steps 1 and 2 respectively, by the properties defined by the `set` function. Both functions (`set` and `gencode`)were run and it was checked if any errors appeared and if the final C-files which are generated were equal to the compiled previously validated files.

4.1.3 Verification of the `fmincon` function

The process of verification of the `fmincon` function not only consisted of the verification of the final result of a problem but also the intermediate steps which lead to the final solution. In the validation of the `fmincon` function, the calculation of the closest critical points, the calculation of the initial point of the optimization and the definition of the remaining constraints defined by (1.16) and (1.17) were taken into account.

Closest critical points

The process of finding the closest critical points to a given initial point by the generation of pseudo-random initial points was validated as explained below. In order to validate this process, the problem described by Otten and Mönnigmann [6] has been considered. The problem is stated in Tab. 4.1.

The mentioned process must ensure that the found closest critical points are indeed the closest ones and that both points belong to different critical manifolds. In Fig. 4.1, the critical manifolds of the problem described by Otten and Mönnigmann [6], as well as the calculated closest critical points are depicted. The asterisks represent the closest critical points calculated, α_{cc1} and α_{cc2}, the circle represents the uncertainty region in which the state is expected to vary and the point in the center, α_0, of the circle is the initial value of the uncertain parameters. It can be seen that each of the closest critical points, α_{cc1} and α_{cc2}, belongs to a different manifold and that those points are at the minimum distance to the initial point α_0.

Initial value for the optimization

The initial value of the optimization consists of the values of the variables stated in Tab. 1.2 and the variable of the distance to the critical points d, that satisfy (1.14) and (1.15) at the critical points, in this case α_{cc1} and α_{cc2}. After the calculation of the initial value of

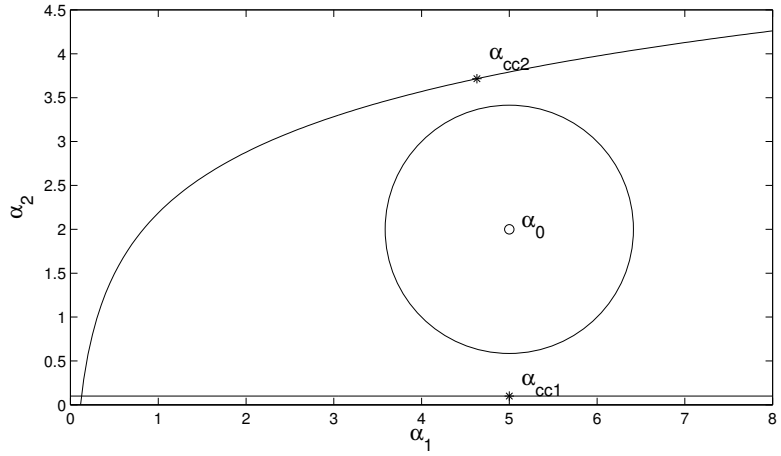


Figure 4.1: Calculated closest critical points, critical manifolds and initial point of the system presented in [6]. The asterisks represent the closest critical points and the circle the initial value of the parameters.

the optimization it was verified that the normal vectors to the manifolds and the distance of the initial point α_0 to them was right. The values of the normal vectors an distances to the manifolds, with which the initial value was compared, were calculated independently taking into account the initial point α_0 and the closest critical points α_{cc1} and α_{cc2}.

New constraints

The validation of the remaining constraints, the connection and the robustness constraint, defined by (1.16) and (1.17) respectively was made numerically, due to the fact that their expressions are known and easy to work with.

Verification of the whole `fmincon` function

The general verification of the `fmincon` function consisted of the resolution of three different problems in which three different bifurcations could be found, modified Fold bifurcation, modified Hopf bifurcation and Hopf bifurcation. Solving these problems also the `gencode` function was validated, the initial validation was made for the problem described in Tab. 4.1, that is modified Fold bifurcation. The resolution of a optimization problem with Fold bifurcations has not been possible because the MAPLE modules of this bifurcation have not been validated yet.

The problem described by Otten and Mönnigmann [6] was solved in order to completely validate the class for modified Fold bifurcations. The system presented by Otten and Mönnigmann is a population model which describes the number of juvenile and mature

individuals considering the reproduction rate of the mature population, the environmental limitations of the population and the mortality rate of juvenile individuals. The objective of the optimization problem is to minimize the percentage of juvenile population. The number of individuals, juvenile and mature, are the dynamic variables of the system, and the reproduction rate and the mortality of juvenile population are the uncertain parameters. The present problem is solved by means of the defined class an the results are compared with the results achieved by Otten and Mönnigmann [6]. In Tab. 4.1 are the information which is supplied to the `set` function and which describes the present problem.

Table 4.1: Parameters of the population model [6].

Number of dynamic variables: 2
Expressions of the delays: $\tau(x(t)) = p_2 - p_3 \exp(-x_2(t))$
Values of known parameters p_1: $1, 1, 0.5$
Initial values of uncertain parameters α_i: $5, 2$
Delay differential equations: $\dot{x}_1(t) = \alpha_1 x_2(t) - \alpha_2 x_1(t) - \alpha_1 \exp(-\alpha_2 \tau(x)) x_2(t - \tau(x))$ $\dot{x}_2(t) = \alpha_1 \exp(-\alpha_2 \tau(x)) x_2(t - \tau(x)) - p_1 x_2^2(t)$
Type of bifurcations: modified Fold, modified Fold
Cost function: $\Phi(x, \alpha^{(0)}) = \frac{x_1}{x_1 + x_2}$
Steady State constraints: $0 = \alpha_1 x_2 - \alpha_2 x_1 - \alpha_1 \exp(-\alpha_2 \tau(x)) x_2$ $0 = \alpha_1 \exp(-\alpha_2 \tau(x)) x_2 - p_1 x_2^2(t)$
Value of sigma (σ): -0.1

The solution obtained for the present problem is represented in Fig. 4.2, as well as the solution given by Otten and Mönnigmann [6]. The solutions are represented in a two-dimensional space defined by both uncertain parameters. Figure 4.2 shows the nominal optimal point of the population model generated with the newly defined class as a cross. A small circle represents the nominal point of this model according to Otten and Mönnigmann [6]. It can be noticed that both solutions are equivalent; therefore the programming of the modified Fold bifurcation can be considered to be right. In contrast to the initial situation depicted in Fig. 4.1, the distance d of the nominal point to the instability boundaries has been reduced to the minimum.

The second bifurcation which is verified is the modified Hopf bifurcation. The modified Hopf bifurcation is a generalized case of the Hopf bifurcation where the parameter σ, the real part of the eigenvalues of the system $\mathrm{Re}\{\lambda\} = \sigma$ may take any value different than zero [5]. Because the value of σ in the problem considered was zero, the modified

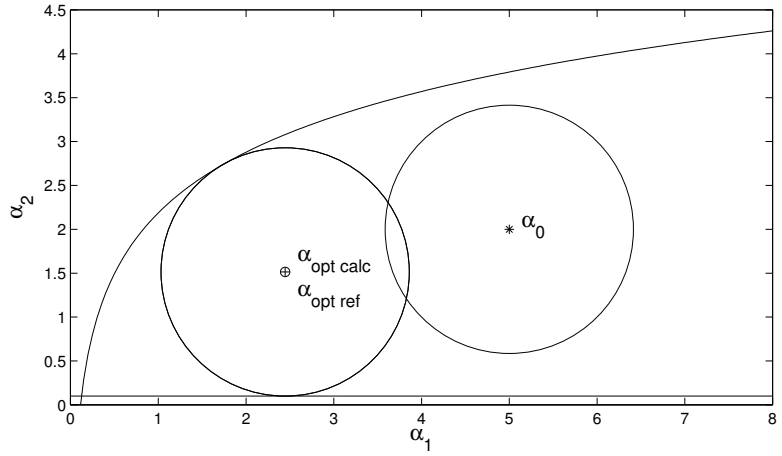


Figure 4.2: Solution of the population model presented in [6]. The robust optima are represented as well as the initial value and thier respectiv uncertainty regions.

Hopf bifurcations found in it correspond to Hopf bifurcations. The problem presented by Otten and Mönnigmann [5] is used with this purpose. A supply chain model that consists of three links: a distributor, a manufacturer an a supplier is described . These three links define three different independent stability boundaries in which six different uncertain parameters are defined. It is a model of six dynamic variables and seven uncertain parameters. The actual object of the optimization is to maximize the demand that can be handled by the supply chain model. Table 4.2 contains the information which is supplied to the `set` function and which describes the problem presented by Otten and Mönnigmann [5].

Both the solution obtained by means of the defined class and the solution achieved by Otten and Mönnigmann [5] can be seen in Fig. 4.3. In this figure it is represented the robust optimum in a plane which is parallel to the α_2-α_5 plane. The lower feasibility boundaries of α_2 and α_5 are also represented in the figure (dashed lines) as well as the stability boundary concerning the second stage of the supply chain model (the manufacturer) (continuous line) as done in by Otten and Mönnigmann [5]. The robust optimum calculated is represented with a small circle and the robust optimum taken as a reference [5] is represented with a cross. It can be seen that both points are coincident. As in the previous example it is also represented a circle that shows the uncertainty region. The position of the uncertainty region in the plane is fixed by the feasibility boundaries and the stability boundary concerning the second stage of the model; this means that both parameters α_2 and α_5 are critical parameters, which could be inferred by the fact that those parameters model the second stage of the supply chain model, the critical stage according to Otten and Mönnigmann [5].

As denoted by Otten and Mönnigmann [5], the stability boundaries of the model do not

Table 4.2: Parameters of the supply chain model [5] with modified Hopf bifurcations.

Number of dynamic variables: 6
Expressions of the delays: $\tau_1(x(t)) = \alpha_1$ $\tau_2(x(t)) = \alpha_1 + \alpha_4$ $\tau_3(x(t)) = \alpha_1 + \alpha_4 + \sqrt{x_1} + 1$ $\tau_4(x(t)) = \alpha_1 + p_{10}$ $\tau_5(x(t)) = \alpha_2$ $\tau_6(x(t)) = \alpha_2 + \alpha_5$ $\tau_7(x(t)) = \alpha_2 + \alpha_5 + \sqrt{x_3} + 1$ $\tau_8(x(t)) = \alpha_2 + p_{11}$ $\tau_9(x(t)) = \alpha_3$ $\tau_{10}(x(t)) = \alpha_3 + \alpha_6$ $\tau_{11}(x(t)) = \alpha_3 + \alpha_6 + \sqrt{x_5} + 1$ $\tau_{12}(x(t)) = \alpha_3 + p_{12}$
Values of known parameters p_i: $1, 1, 1, 0.1, 0.1, 0.1, 0.01, 0.01, 0.01, 11, 12, 13$
Initial values of uncertain parameters α_i: $3, 3, 3.5, 4, 4, 4, 10$
Delay differential equations[1]: $\dot{x}_1(t) = x_2$ $\dot{x}_2(t) = -\frac{1}{p_1}(p_4(x_{1\tau_3} - \alpha_7) + p_7(x_{1\tau_1} - x_{1\tau_2}) + \frac{1+\alpha_4 p_7}{p_{10}}(\alpha_7 - \alpha_7) + x_2)$ $\dot{x}_3(t) = x_4$ $\dot{x}_4(t) = -\frac{1}{p_2}(p_5(x_{3\tau_7} - x_{1\tau_5}) + p_8(x_{3\tau_5} - x_{3\tau_6}) + \frac{1+\alpha_5 p_8}{p_{11}}(x_{1\tau_5} - x_{1\tau_8}) + x_4)$ $\dot{x}_5(t) = x_6$ $\dot{x}_6(t) = -\frac{1}{p_3}(p_6(x_{5\tau_{11}} - x_{3\tau_9}) + p_9(x_{5\tau_9} - x_{5\tau_{10}}) + \frac{1+\alpha_6 p_9}{p_{12}}(x_{3\tau_9} - x_{3\tau_{12}}) + x_6)$
Type of bifurcations: modified Hopf, modified Hopf, modified Hopf
Cost function: $\Phi(x, \alpha^{(0)}) = -\alpha_7$
Steady State constraints: $0 = \alpha_1 x_2 - \alpha_2 x_1 - \alpha_1 \exp(-\alpha_2 \tau(x)) x_2$ $0 = \alpha_1 \exp(-\alpha_2 \tau(x)) x_2 - p_1 x_2^2(t)$
Value of sigma (σ): 0

influence each other and the the stability boundary of the second stage of the model determines the robust optimum. For these reasons, the values of the other uncertain parameters except for α_7, which defines the cost function, can differ from the values of the robust optimum proposed by Otten and Mönnigmann [5]. This aspect can be visualized in Fig. 4.4, where both robust optima, the calculated by means of the defined class and the optimum proposed by Otten and Mönnigmann [5], are represented by a circle and a cross respectively. The points are shown in a plane parallel to the α_3-α_5 plane, as well as the stability boundaries of the second and third stage (continuous lines) and the lower feasibility constraints of α_3 and α_5 (dashed lines). In this case, both robust optima are not coincident due to the fact that the parameter α_3 differs. The distance of the robust

[1] $x_{i\tau_k} = x_i(t - \tau_k)$

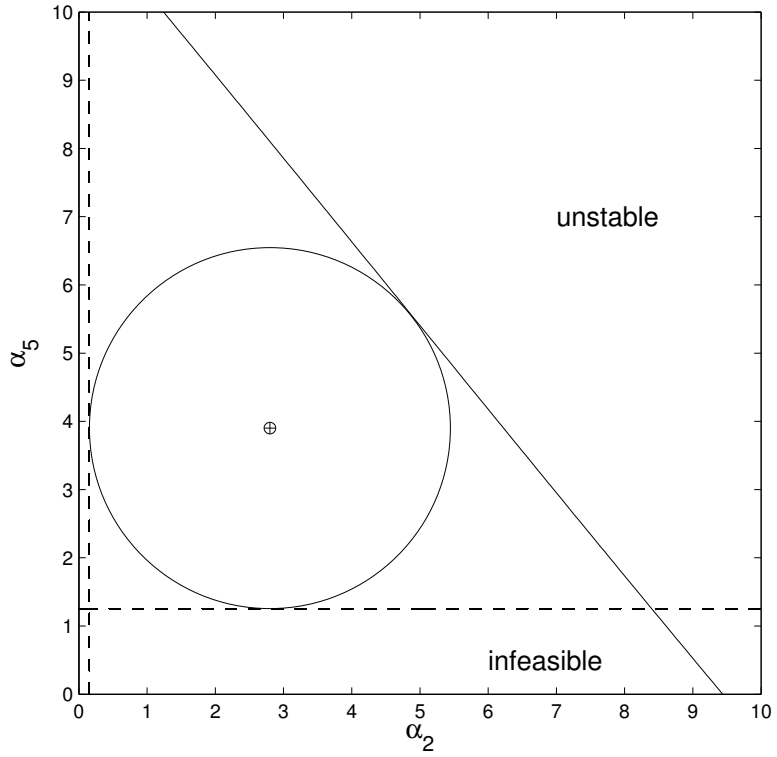


Figure 4.3: Robust optimum of the supply chain model [5] with modified Hopf bifurcations in a α_2-α_5 plane.

optima to the feasibility boundary of the parameter α_5 can not be reduced as denotes the uncertainty regions; a dotted line is used to represent the region of the reference, the uncertainty region of the calculated optimum is represented by a continuous line. Due to the chosen plane, it may seem that the distance to the stability constraint of the second stage can be reduced, however, this distance is reduced to the minimum if other plane is chosen as done by Otten and Mönnigmann [5]. It must be noticed that the distance of the point to the lower feasibility boundary of α_3 is not the minimum according to the robustness constraint defined by (1.17).

In Fig. 4.5 the nominal point of the supply chain model obtained by means of the class is represented by a cross and the nominal point of this model according to Otten and Mönnigmann [5] by a circle. These points are represented in a plane parallel to the α_2-α_7 plane. The cut of the feasibility boundary of α_2 and the stability boundary of the second stage with the plane are also represented. As in Fig. 4.3, both points are coincident because these parameters define the critical stability boundary and the cost function. As in Fig. 4.3 the distance of the robust optimum to the feasibility boundary of α_2 can not be reduced. Due to the chosen plane, it may seem that the distance to the stability constraint of the second stage can be reduced, however, this distance is reduced to the minimum if other plane is chosen as in [5].

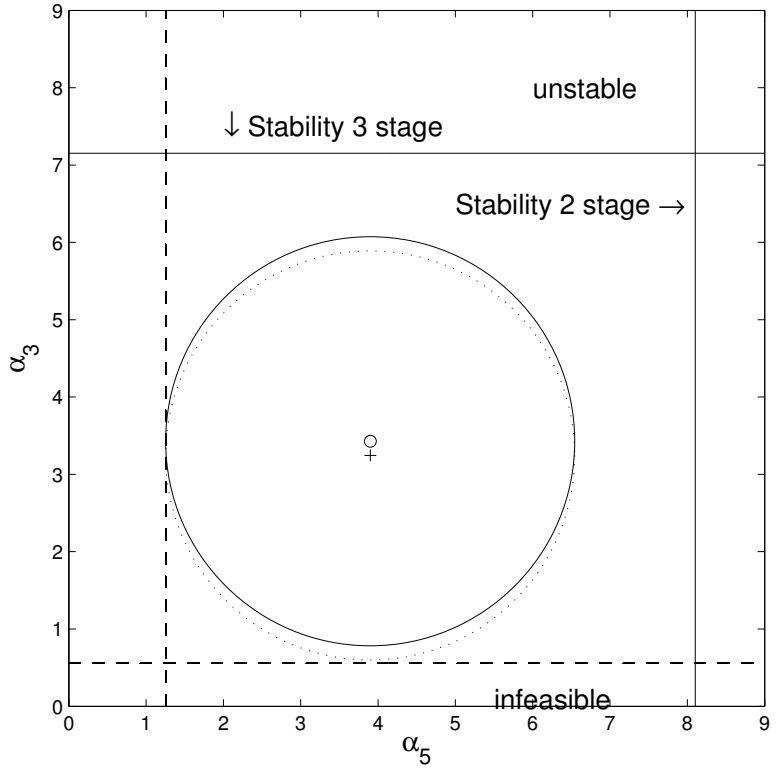


Figure 4.4: Robust optimum of the supply chain model [5] with modified Hopf bifurcations in a α_5-α_3 plane.

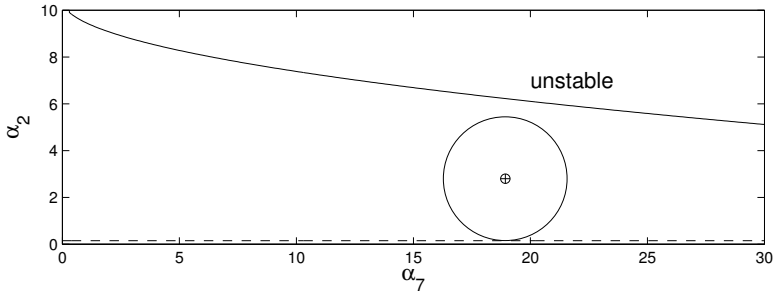


Figure 4.5: Robust optimum of the supply chain model [5] with modified Hopf bifurcations in a α_7-α_2 plane.

If the distances d of the robust optimum to each manifold are compared (4.1), it can be noticed that just one of the manifolds are critical and therefore the distance of the robust optimum point to this manifold is minimum. It can be seen that the distances to the first and third manifolds are longer than the distance to the second manifold, which is equivalent to $\sqrt{7}$, the minimum distance.

$$d_{nom} = \begin{bmatrix} 2.9144 \\ 2.6458 \\ 3.2079 \end{bmatrix} \tag{4.1}$$

Both robust optima can be numerically compared. The robust optimum of the authors Otten and Mönnigmann [5] is defined by (4.2) and the calculated optimum by (4.3). It can be noticed that the values of the parameters α_2, α_5 and α_7 are coincident as explained before and the values of the parameters concerning the first and third stage of the supply chain model α_1, α_3, α_4 and α_6 differ

$$\alpha_{\text{rob opt ref}} = \begin{bmatrix} 2.81514 \\ 2.8 \\ 3.24352 \\ 3.00219 \\ 3.9 \\ 2.81043 \\ 18.9267 \end{bmatrix} \tag{4.2}$$

$$\alpha_{\text{rob opt calc}} = \begin{bmatrix} 2.80525 \\ 2.80000 \\ 3.42755 \\ 3.07132 \\ 3.90000 \\ 2.76714 \\ 18.9268 \end{bmatrix} \tag{4.3}$$

The problem used in the verification of the Hopf bifurcation is also the problem presented by Otten and Mönnigmann [5], since the value of σ is zero and the modified Hopf bifurcation and the Hopf bifurcation are equivalent in this case. Table 4.3 contains the information supplied to the class by means of the `set` function.

The resolution of this optimization problem does not lead to the same results as the problem presented before in which three modified Hopf bifurcations can be found. Admitting that the solution of both problems must be the same, the intermediate steps of the resolution of both problems have been compared in order to find the point in which the optimizations differ.

After the generation of the .MEX functions which define the manifold and normal vector system constraints of the problem with the Hopf bifurcations, these equality constraints were numerically compared to the same constraints of the problem with the modified Hopf bifurcation. By the numerical evaluation of the constraints small differences were revealed, which can be caused by approximation errors. The `gencode` function was considered to

Table 4.3: Parameters of the supply chain model [5] with Hopf bifurcations.

Number of dynamic variables: 6
Expressions of the delays: $\tau_1(x(t)) = \alpha_1$ $\tau_2(x(t)) = \alpha_1 + \alpha_4$ $\tau_3(x(t)) = \alpha_1 + \alpha_4 + \sqrt{x_1} + 1$ $\tau_4(x(t)) = \alpha_1 + p_{10}$ $\tau_5(x(t)) = \alpha_2$ $\tau_6(x(t)) = \alpha_2 + \alpha_5$ $\tau_7(x(t)) = \alpha_2 + \alpha_5 + \sqrt{x_3} + 1$ $\tau_8(x(t)) = \alpha_2 + p_{11}$ $\tau_9(x(t)) = \alpha_3$ $\tau_{10}(x(t)) = \alpha_3 + \alpha_6$ $\tau_{11}(x(t)) = \alpha_3 + \alpha_6 + \sqrt{x_5} + 1$ $\tau_{12}(x(t)) = \alpha_3 + p_{12}$
Values of known parameters p_i: $1, 1, 1, 0.1, 0.1, 0.1, 0.01, 0.01, 0.01, 11, 12, 13$
Initial values of uncertain parameters α_i: $3, 3, 3.5, 4, 4, 4, 10$
Delay differential equations[2]: $\dot{x}_1(t) = x_2$ $\dot{x}_2(t) = -\frac{1}{p_1}(p_4(x_{1\tau_3} - \alpha_7) + p_7(x_{1\tau_1} - x_{1\tau_2}) + \frac{1+\alpha_4 p_7}{p_{10}}(\alpha_7 - \alpha_7) + x_2)$ $\dot{x}_3(t) = x_4$ $\dot{x}_4(t) = -\frac{1}{p_2}(p_5(x_{3\tau_7} - x_{1\tau_5}) + p_8(x_{3\tau_5} - x_{3\tau_6}) + \frac{1+\alpha_5 p_8}{p_{11}}(x_{1\tau_5} - x_{1\tau_8}) + x_4)$ $\dot{x}_5(t) = x_6$ $\dot{x}_6(t) = -\frac{1}{p_3}(p_6(x_{5\tau_{11}} - x_{3\tau_9}) + p_9(x_{5\tau_9} - x_{5\tau_{10}}) + \frac{1+\alpha_6 p_9}{p_{12}}(x_{3\tau_9} - x_{3\tau_{12}}) + x_6)$
Type of bifurcations: Hopf, Hopf, Hopf
Cost function: $\Phi(x, \alpha^{(0)}) = -\alpha_7$
Steady State constraints: $0 = \alpha_1 x_2 - \alpha_2 x_1 - \alpha_1 \exp(-\alpha_2 \tau(x)) x_2$ $0 = \alpha_1 \exp(-\alpha_2 \tau(x)) x_2 - p_1 x_2^2(t)$

work properly.

The intermediate steps of the newly defined `fmincon` function were compared as well. The comparison of the closest critical points showed an error on this step. The closest critical points were not totally coincident. The closest critical points found for the Hopf bifurcation differed from the closest critical points found for the modified Hopf bifurcation. Although the closest critical points calculated for the Hopf bifurcation were different between them, when they were used to find the robust optimum of the problem, two of the points belonged to the same manifold and therefore a normal vector was repeated. Without considering one of the manifolds, the solution achieved was not the expected one. According to this result it can be stated that the generation of the pseudo-random

[2] $x_{i\tau_k} = x_i(t - \tau_k)$

initial points is not precise enough.

4.1.4 Verification of the `get` function

The `get` function was easily verified comparing the values of the variables provided by it and the expected values of the variables, extracted from the solution available after the call of the `fmincon` function

4.2 Limitations of the automated initialization

As explained in Sec. 2.4 one of the previous steps to the optimization of the problem is the calculation of the initial value of the optimization. Loosely speaking, the calculation of the initial value of the optimization is done in two steps, the calculation of the closest critical points and the calculation of the normal vector according to these points. If no initial points are supplied by the user, the closest critical points are calculated in an iterative way in order to obtain different closest critical points. For a better understanding, the points which are calculated in each step of the calculation of the initial point of the optimization of the population model are represented in Fig. 4.6 and detailed explanation of the process is given below.

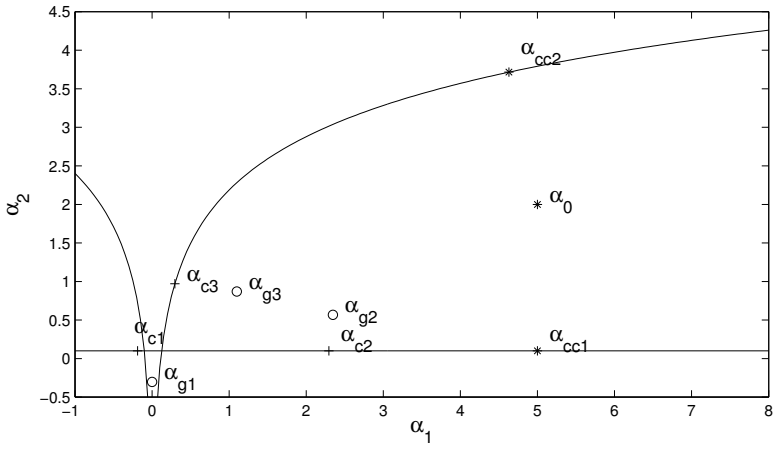


Figure 4.6: Points generated to calculate the closest critical point of the population model. The points generated pseudo-randomly in order to look for a critical point, α_{gi}, are represented by a circle, the critical points α_{ci} are represented by a cross and the closest critical points α_{cci} are represented by an asterisk, as well as the initial point α_0.

Firstly the point α_{g1} is generated pseudo-randomly in order to look for a critical point. The use of the point α_{g1} leads to the critical point α_{c1} which belongs to the first manifold.

Through α_{c1}, the first closest critical point of the problem α_{cc1} is found. In order to calculate the initial point of the optimization, the closest critical point of each manifold is needed; the closest critical point of the second manifold is pursued now. A new point is generated pseudo-randomly, in this case α_{g2}, which leads to α_{c2}. The point α_{c2} lies in the first manifold, therefore the closest critical point found is α_{cc1} again. In this point a new point is generated, α_{g3}, which leads to the critical point α_{c3}. The point α_{c3} lies in the second manifold and is used to find the closest critical point of this manifold α_{cc2}. The length of this process depends to a great extent on the problem and the generation of the points. The most relevant aspects that influence the duration of the process are presented below:

- **Number of manifolds**. As the number of manifolds increases, the number of closest critical points increases as well. The process is longer not only because of the number of critical points that must be found but also because the generated points may lead to a manifold whose critical point has been already found. As new closest critical points are calculated, the region of the space which leads to closest critical points of new manifolds becomes smaller. The number of points which have to be generated in order to obtain the last critical points is larger in the last manifolds. For instance, in the supply chain model proposed by Otten and Mönnigmann [5], the number of generated points to obtain the first critical point was 2, to obtain the second critical point four different points were generated and finally the third critical point was found after eight points were generated. Within the generated points are included points whose use does not allow the convergence of the numerical calculations.

- **Type of the bifurcations**. The same closest critical point can be found several times because the functions that are used to find them are common to the same type of bifurcation. Which makes possible to find points in different manifolds is the initial value generated. Therefore, if all the bifurcations of a problem are of the same type, the search of the initial point of the optimization takes longer than if the bifurcations are different.

- **The system**. The algorithm of the generation is common to all the problems. According to the region of the space in which the manifolds are, their critical points are found with a greater or lesser difficulty. The points are generated pseudo-randomly and further from the origin as the number of generated numbers increases. For this reason the manifolds which are further from the origin may involve a larger number of iterations for the generation of the initial points. It may also happen that the manifolds are close to each other and only critical points of one of the manifolds

are found.

The search of the closest critical points needed for the calculation of the initial value of the optimization is the step of the optimization which lasts longer when the initial values must be generated, specially if the number of manifolds increases. The circumstances which are presented above do not affect the calculation of the initial point of the optimization when an initial value is supplied initially by the user.

Chapter 5

Conclusion

A new class has been defined in order to perform the robust optimization of delayed systems. The proposed class can store the necessary data to define the problem, carry out the optimization and save the results. The object oriented environment allows to work with this kind of problems easily, decreasing the number of possible errors due to less interaction of the user with the mathematical software.

The robust optimization of the delayed systems is performed in four different steps: the definition of the problem, the generation of the constraints according to the normal vector method, the numerical optimization of the problem and the visualization of the results. From the users point of view, the most time consuming step is the definition of the problem, followed by the visualization of the results. This is due to the fact that the generation of the constraints and the optimization of the problem are done automatically using the information supplied by the user in the definition step.

The problem is now defined in a more user friendly way; there is not need to know the syntax of the different mathematical software because most the information is requested in numerical vectors except for the expressions of the DDEs and the types of the bifurcations. In addition, there is the possibility of defining the problem in a slower way in which the type of information that is required is specified. The definition step not only comprises the actual definition of the problem but also some characteristics of the optimization, such as the options of the numerical solvers or the definition of optional previous initial values for the optimization. Due to the actions carried out internally by means of the data supplied by the user, it is completely avoided to handle Maple code and, except for the the definition of the problem, Matlab code as well.

The generation of the constraint functions and the optimization are now automatic actions which are carried out without the intervention of the user. The necessary functions are

generated after the `gencode` function is called. These functions are ready to be used in the optimization without any changes. If an attempt to optimize the problem is done and the functions are not available, an error message is displayed as well as other necessary information is also missing.

The actual optimization of the problem can be carried out after the generation of the constraint functions just by calling the specific function. During this step, the remaining constraints, equations and inequations, are defined, and initial value of the optimization is found automatically. The calculation of the initial value of the optimization is the most critical step in the current object due to its influence on the final solution. The difficulty of finding the proper initial value of the optimization is due to the fact that the critical manifolds can be located in very different points of the space according to the problem. It has been probed that the method used to generate the initial point of the optimization when no initial values are supplied can provide valid initial values but it could be improved if more information concerning the manifolds were available. When points close to critical points in the different manifolds are supplied the optimization of the problem is performed properly as indicated in the previous chapter. The results achieved after the optimization are available in the object an can be displayed in a more intuitive way by means of the defined function.

The proper operation of the defined class has been validated for the modified Fold bifurcation, the Hopf bifurcation and the modified Hopf bifurcation. The Fold bifurcation is completely programmed but it has not been validated because the MAPLE modules which would generate its functions of constraints are not validated by now.

In contrast to the method used before the definition of the new class, the proposed class presents a the following characteristics. The difficulty of defining the problem and running its optimization has been reduced by the actions which are carried out internally during the definition. The MAPLE syntax remains inside the class code, avoiding its knowledge by the user. The necessary changes that have to be done according the bifurcation are done automatically, which means time saving and the reduction of errors. Loosely speaking, the user just has to define the problem.

The future lines of the present master thesis can be classified within the three following topics:

- **Improvement of the present optimization**. Here can be included the development of the algorithm through which the closest critical points are achieved. In order to this the software DDE-BIFTOOL can be very helpful.
- **Addition of new features**. Some useful features to be included are the plot of

the results, an interesting tool on the verification process; the verification of the Fold bifurcation and the facilitation of the definition of the expressions of the cost function, the steady state constraints, the delay differential equations and the delays. Other types of bifurcation could also be included.

- **Improvement of the handling**. The handling of the class and the visualization of the results can be improved building a MATLAB GUI, by means of which the definition of the problem can be easily done and the results showed after the optimization.

Chapter 6

Appendix

6.1 Augmented system and normal vector system expressions. Modified Fold bifurcation

The equation (6.1) describes the augmented system of a delayed system in which a modified Fold bifurcation can be found.

$$\left.\begin{array}{c} f(\tilde{x}^{(c)}, \tilde{x}^{(c)}, ..., \tilde{x}^{(c)}, \alpha^{(c)}) = 0 \\ \sigma w - J_0' w - \sum_{i=1}^{m} J_i' \exp(-\sigma \tau_i) w = 0 \\ w'w - 1 = 0 \end{array}\right\} G \tag{6.1}$$

$\tilde{x}^{(c)}, \alpha^{(c)}$ represent critical steady states and critical parameters respectively, σ is the real part of the eigenvalue of the problem which is considered, w is the eigenvector corresponding to the same eigenvalue, J_k, $k = 1, ..., m$ are the Jacobian matrices with respect to $x(t)$ and $x(t - \tau_k)$ respectively, τ_k, $k = 1, ..., m$ are the delays and m is the number of delays.

The equations that describe the normal vector system for a modified Fold bifurcations

are:

$$\left.\begin{array}{c} f(\tilde{x}^c, \tilde{x}^c, ..., \tilde{x}^c, \alpha^c) = 0 \\ \sigma w - J_0' w - \sum_{i=1}^{m} J_i' \exp(-\sigma\tau_i) w = 0 \\ w'w - 1 = 0 \end{array}\right\} G \tag{6.2}$$

$$\left.\begin{array}{c} \begin{bmatrix} \nabla_{\tilde{x}} f' & B_{12}^{exp} 0 \\ 0 & \sigma I - \sum_{i=1}^{m} J_i' \exp(-\sigma\tau_i) 2w \end{bmatrix} \kappa = 0 \\ \begin{bmatrix} \nabla_{\alpha} f' & -\sum_{i=1}^{m} \exp(-\sigma\tau_i) \nabla_{\alpha}(w' J_i) & 0 \end{bmatrix} \kappa - r = 0 \end{array}\right\} H \tag{6.3}$$

where $\tilde{x}^{(c)}, \alpha^{(c)}, \sigma, w, J_k$ and m are as in (6.1), $\nabla_{\tilde{x}} f$ is the Jacobian of the delay differential equations with respect to a steady state $\tilde{x}, \kappa$ collects auxiliary variables, $\nabla_{\tilde{\alpha}} f$ is the Jacobian of the delay differential equations with respect to$\tilde{\alpha}, r$ is the normal vector and B_{12}^{exp} is defined by (6.4)

$$B_{12}^{exp} = -\sum_{i=1}^{m} \exp(-\sigma\tau_i) \nabla_{\tilde{x}}(w' J_i) + \sigma \sum_{i=1}^{m} w' J_i \exp(-\sigma\tau_i) \nabla_{\tilde{x}} \tau_i \tag{6.4}$$

being $\nabla_{\tilde{x}} \tau_i$ the derivatives with respect to $\tilde{x}$ of the expressions of the delays.

6.2 Resolution of a problem by means of the defined class

The problem here solved is the problem described in Tab. 4.2 and presented by Otten and Mönnigmann [5].

```
%% Definition of the problem
% Definition of the object
SCoptimization=OptimProb();

% Name of the problem
set(SCoptimization,'name','SCoptim')

% Number of dynamic variables
set(SCoptimization,'dynVars',6)
```

```
10
11 % Number of delays and expressions
12 tausExpr={'tau[ 1]=hD1',...
13     'tau[ 2]=hD1+hP1',...
14     'tau[ 3]=hD1+hP1+sqrt(x1)+1',...
15     'tau[ 4]=hD1+hA1',...
16     'tau[ 5]=hD2',...
17     'tau[ 6]=hD2+hP2',...
18     'tau[ 7]=hD2+hP2+sqrt(x3)+1',...
19     'tau[ 8]=hD2+hA2',...
20     'tau[ 9]=hD3',...
21     'tau[10]=hD3+hP3',...
22     'tau[11]=hD3+hP3+sqrt(x5)+1',...
23     'tau[12]=hD3+hA3'};
24 set(SCoptimization,'delays',12,tausExpr)
25
26 % Number of known parameters and values and number of uncertain
       parameters and values
27 p=[1,1,1,0.1,0.1,0.1,0.01,0.01,0.01,11,12,13];
28 alpha=[3,3,3.5,4,4,4,10];
29 set(SCoptimization,'parameters',12,p,7,alpha)
30
31 % Expressions of the delayed differental equations
32 ddesexps={'`x1'`' = x2',...
33     '`x2'`' = -1/T1*(ai1*( x1Del3 -d     ) + aWIP1*(x1Del1-
         x1Del2) + (1+hP1*aWIP1)/hA1* (d -d          ) + x2)',...
34     '`x3'`' = x4',...
35     '`x4'`' = -1/T2*(ai2*( x3Del7-x1Del5) + aWIP2*(x3Del5-
         x3Del6) + (1+hP2*aWIP2)/hA2*(x1Del5- x1Del8) + x4)',...
36     '`x5'`' = x6',...
37     '`x6'`' = -1/T3*(ai3*(x5Del11-x3Del9) + aWIP3*(x5Del9-
         x5Del10) + (1+hP3*aWIP3)/hA3*(x3Del9-x3Del12) + x6)'};
38 set(SCoptimization,'dDEs',ddesexps)
39
40 % Number bifurcations and type
41 set(SCoptimization,'bifurcations',3,{'modHopf','modHopf','
       modHopf'})
```

```
42
43 % Cost function
44 costF=@(x)(-x(6+7));
45 set(SCoptimization,'costFunc',costF);
46
47 % Steady state constraints
48 SSConstr=@(x)(x(1:6)-[0,x(6+7),0,x(6+7),0,x(6+7)]);
49 set(SCoptimization,'sSConstraints',SSConstr);
50
51 % Optional parameters
52 % Lower and upper boundaries
53 set(SCoptimization,'ub',[Inf,Inf,Inf,Inf,Inf,Inf,1e6],'alpha')
54 set(SCoptimization,'ub',[Inf,Inf,Inf,Inf,Inf,Inf],'x')
55 set(SCoptimization,'lb',[2.8, 2.8, 3.2, 2.75, 3.9, 2.65, sqrt(7)
    ],'alpha')
56 set(SCoptimization,'lb',[-Inf,-Inf,-Inf,-Inf,-Inf,-Inf],'x')
57
58 % Numerical options
59 OptionsFS=optimoptions('fsolve','Algorithm','levenberg-marquardt
    ','Display','off','MaxIter',1e6,'TolFun',1e-15,'TolX',1e-12);
60 OptionsOpt=optimoptions('fmincon','Algorithm','active-set','
    TolCon',1e-12,'TolFun',1e-12,'MaxFunEvals',20000, 'display','
    off');
61 set(SCoptimization,'optionsFS',OptionsFS)
62 set(SCoptimization,'optionsOpt',OptionsOpt)
63
64 % Names of the variables
65 acctualNames={'x1','x2','x3','x4','x5','x6',...
66 'hD1','hD2','hD3','hP1','hP2','hP3','d',...
67 'T1','T2','T3','ai1','ai2','ai3','aWIP1','aWIP2','aWIP3','hA1','
    hA2','hA3',...
68 'x1Del1', 'x2Del1', 'x3Del1', 'x4Del1', 'x5Del1', 'x6Del1',...
69 'x1Del2', 'x2Del2', 'x3Del2', 'x4Del2', 'x5Del2','x6Del2',...
70 'x1Del3','x2Del3','x3Del3','x4Del3','x5Del3','x6Del3',...
71 'x1Del4','x2Del4','x3Del4','x4Del4','x5Del4','x6Del4',...
72 'x1Del5','x2Del5','x3Del5','x4Del5','x5Del5','x6Del5',...
73 'x1Del6','x2Del6','x3Del6','x4Del6','x5Del6','x6Del6',...
```

```
'x1Del7','x2Del7','x3Del7','x4Del7','x5Del7','x6Del7',...
'x1Del8','x2Del8','x3Del8','x4Del8','x5Del8','x6Del8',...
'x1Del9','x2Del9','x3Del9','x4Del9','x5Del9','x6Del9',...
'x1Del10','x2Del10','x3Del10','x4Del10','x5Del10','x6Del10',...
'x1Del11','x2Del11','x3Del11','x4Del11','x5Del11','x6Del11',...
'x1Del12','x2Del12','x3Del12','x4Del12','x5Del12','x6Del12'};
set(SCoptimization,'nameVars',acctualNames)

% Initial value of the variables of the augmented system which
    describes the manifold
set(SCoptimization,'initialAug',SCinitialAug)

% Initial value of the variables of the normalvector system
set(SCoptimization,'initialNV',SCinitialNV)


%% Generation of the code
gencode(SCoptimization)


%% Optimization of the problem
[xopt,Jopt,exitflag,output,lambda,gradient] = fmincon(
    SCoptimization);


%% Visualization of the results
% Robust optimum state variables
[x_rob_opt]=get(SCoptimization,'x','nominal');

% Robust optimum uncertain parameters
[alpha_rob_opt]=get(SCoptimization,'alpha','nominal');

% The state variables of the closest critical point in the first
    manifold
[x_crit_1]=get(SCoptimization,'x','critical',1);

% The uncertain parameters of the closest critical point in the
```

```
    second manifold
[alpha_crit_2]=get(SCoptimization,'alpha','critical',2);

% The normal vector to the first manifold
[NV1]=get(SCoptimization,'NV',1);
```

Bibliography

[1] Elliott, T. R., and Luyben, W. L. Capacity-based economic approach for the quantitative assessment of process controllability during the conceptual design stage. *Industrial & Engineering Chemistry Research 34*, 11 (November 1995), 3907–3915.

[2] Kastsian, D., and Mönnigmann, M. Impact of delay on robust stable optimization of a cstr with recycle stream. In *10th IFAC International Symposium on Dynamics and Control of Process Systems* (December 2013), vol. 10, pp. 433–438.

[3] Mönnigmann, M. *Constructive Nonlinear Dynamics for the Design of Chemical Engineering Processes*. PhD thesis, RWTH Aachen University, 2004.

[4] Mönnigmann, M., and Marquardt, W. Normal vectors on manifolds of critical points for parametric robustness of equilibrium solutions of ODE systems. *Journal of Nonlinear Science 12*, 2 (May 2002), 85–112.

[5] Otten, J., and Mönnigmann, M. Robust optimization of delay differential equations with state and parameter dependent delays. In *Proceedings of the 55th IEEE Conference on Decision and Control* (2016). Accepted.

[6] Otten, J., and Mönnigmann, M. Robust steady state optimization with state dependent delays. In *Proceedings of the 13th IFAC Workshop on Time Delay Systems* (June 2016), vol. 49, pp. 47–52.

[7] Sipahi, R., Niculescu, S.-I., Abdallah, C. T., Michiels, W., and Gu, K. Stability and stabilization of systems with time delay. *IEEE Control Systems Magazine 31(1)* (February 2011), 38–65.

[8] The MathWorks, Inc. Attribute specification. `http://uk.mathworks.com/help/matlab/matlab_oop/specifying-attributes.html`, Accessed August 1, 2016.

[9] The MathWorks, Inc. Class attributes. `http://uk.mathworks.com/help/matlab/matlab_oop/class-attributes.html`, Accessed August 1, 2016.

[10] The MathWorks, Inc. Classdef block. http://uk.mathworks.com/help/matlab/matlab_oop/classdef-block.html, Accessed August 1, 2016.

[11] The MathWorks, Inc. Event and listener concepts. http://uk.mathworks.com/help/matlab/matlab_oop/events-and-listeners--concepts.html#bq8nr3l, Accessed August 1, 2016.

[12] The MathWorks, Inc. How to use properties. http://uk.mathworks.com/help/matlab/matlab_oop/how-to-use-properties.html, Accessed August 1, 2016.

[13] The MathWorks, Inc. Properties with constant values. http://uk.mathworks.com/help/matlab/matlab_oop/properties-with-constant-values.html, Accessed August 1, 2016.

[14] The MathWorks, Inc. Property attributes. http://uk.mathworks.com/help/matlab/matlab_oop/property-attributes.html, Accessed August 1, 2016.

[15] The MathWorks, Inc. Property set methods. http://uk.mathworks.com/help/matlab/matlab_oop/property-set-methods.html, Accessed August 1, 2016.

[16] The MathWorks, Inc. Role of classes in matlab. http://uk.mathworks.com/help/matlab/matlab_oop/classes-in-the-matlab-language.html, Accessed August 1, 2016.

[17] The MathWorks, Inc. Save and load process for objects. http://uk.mathworks.com/help/matlab/matlab_oop/understanding-the-save-and-load-process.html#br0li_j, Accessed August 1, 2016.

[18] The MathWorks, Inc. Subclass multiple classes. http://uk.mathworks.com/help/matlab/matlab_oop/subclassing-multiple-classes.html#breg88p, Accessed August 1, 2016.

[19] The MathWorks, Inc. How to use methods. http://uk.mathworks.com/help/matlab/matlab_oop/how-to-use-methods.html, Accessed August 2, 2016.

[20] The MathWorks, Inc. Method attributes. http://uk.mathworks.com/help/matlab/matlab_oop/method-attributes.html, Accessed August 2, 2016.

[21] The MathWorks, Inc. Methods and functions. http://uk.mathworks.com/help/matlab/matlab_oop/specifying-methods-and-functions.html#responsive_offcanvas, Accessed August 2, 2016.

[22] The MathWorks, Inc. Methods in separate files. http://uk.mathworks.com/help/matlab/matlab_oop/methods-in-separate-files.html, Accessed August 2, 2016.

[23] The MathWorks, Inc. Overload functions in class definitions. http://uk.mathworks.com/help/matlab/matlab_oop/overloading-functions-for-your-class.html, Accessed August 2, 2016.

[24] The MathWorks, Inc. Overview events and listeners. http://uk.mathworks.com/help/matlab/matlab_oop/learning-to-use-events-and-listeners.html, Accessed August 2, 2016.

[25] The MathWorks, Inc. fmincon. http://uk.mathworks.com/help/optim/ug/fmincon.html?s_tid=srchtitle#responsive_offcanvas, Accessed August 3, 2016.

[26] The MathWorks, Inc. Object-oriented programming in matlab. http://uk.mathworks.com/discovery/object-oriented-programming.html?s_tid=srchtitle, Accessed July 4, 2016.

[27] The MathWorks, Inc. Why use object-oriented design. http://uk.mathworks.com/help/matlab/matlab_oop/why-use-object-oriented-design.html, Accessed July 4, 2016.