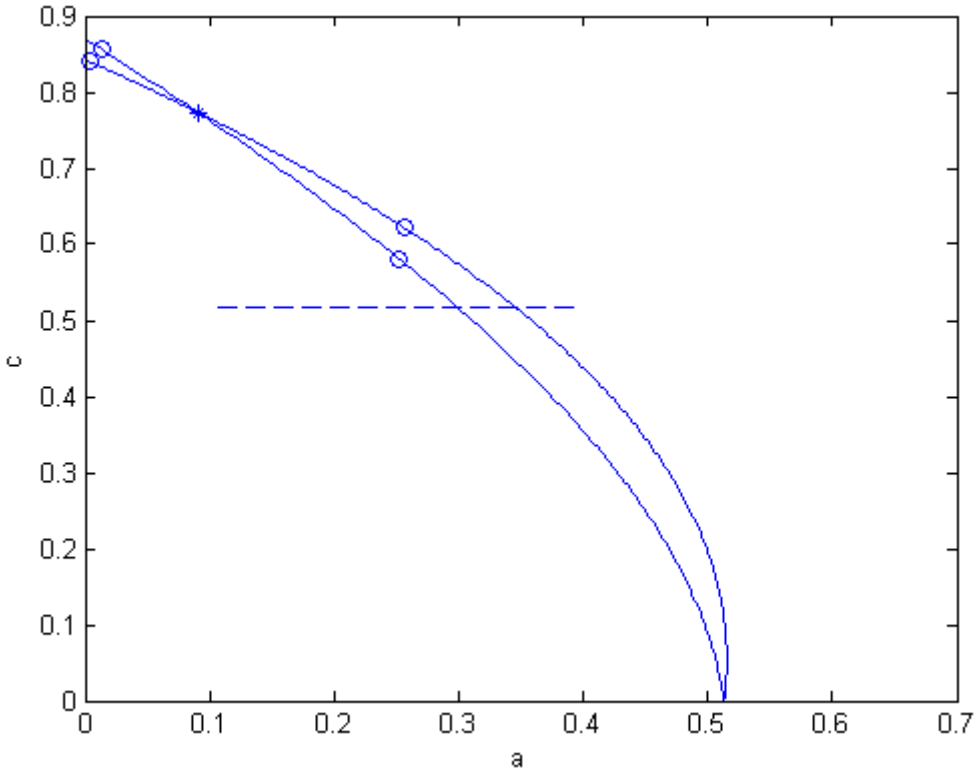
Normal form computations for Delay Differential Equations in DDE-BIFTOOL

Bram Wage
Universiteit Utrecht

Written under the supervision of Yuri Kuznetsov
Second examiner: Sjoerd Verduyn Lunel

August 2014



Abstract

Delay Differential Equations (DDEs) appear in many applications, including neuroscience, ecology, and engineering. The analysis of one- and two-parameter families of bifurcations is based on computing normal forms of ODEs without delays describing the dynamics on center manifolds. We give an overview of so-called sun-star calculus of dual semigroups necessary to derive symbolic formulas for the critical normal form coefficients for the Hopf, Generalized Hopf, Zero-Hopf and Double Hopf bifurcations. We then discuss their implementation in the Matlab package DDE-BIFTOOL. Additionally, detection of these bifurcations was implemented. We demonstrate the new features by detecting bifurcations and computing their normal form coefficients in several DDE models.

Contents

Chapter 1

Introduction

1.1 Bifurcations and Delay Differential Equations

In science, many phenomena can be modelled by Ordinary Differential Equations (ODEs). An ODE is given by

$$\dot{x}(t) = f(t, x(t), \alpha),$$

where x is a function $\mathbb{R} \to \mathbb{R}^n$, f is a function $\mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, α is a parameter in $\mathbb{R}^m$, and the dot represents the derivative with respect to t. In many applications, the ODE is autonomous:

$$\dot{x}(t) = f(x(t), \alpha), \tag{1.1}$$

i.e. it doesn't depend explicitly on time. (This is the case when the system that is being modelled is free of external forcing.)

As a quick recap, we provide a short description of the analysis of bifurcations for ODEs, using the Hopf bifurcation as an example.

When continuously changing the parameter, the phase portrait of the ODE may undergo a qualitative change. This is called a **bifurcation**. The type of bifurcation is determined by the spectrum of the derivative Df of f. A Hopf bifurcation takes place if an eigenvalue pair crosses the imaginary axis, see Figure 1.1. If a system of the form (1.1) undergoes a Hopf

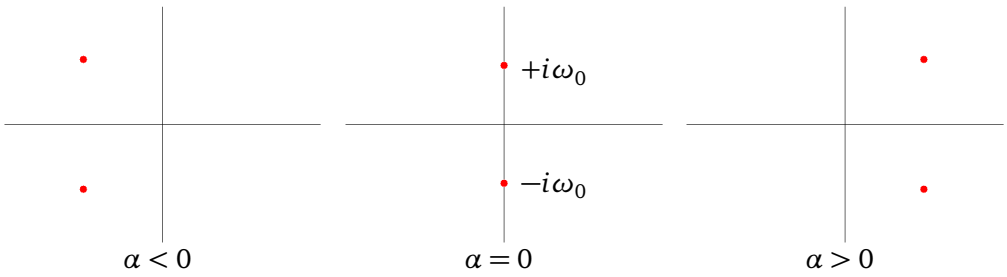


FIGURE 1.1: *At a Hopf bifurcation, a complex eigenvalue pair crosses the imaginary axis.*

bifurcation, then near the origin, it's locally topologically equivalent to the system

$$\dot{\rho} = \rho\left(\beta(\alpha) + L_1(\alpha)\rho^2\right),$$
$$\dot{\varphi} = 1.$$

Its complex form is

$$\dot{z} = z\left(\beta(\alpha) + i + L_1(\alpha)|z|^2\right).$$

The parameter $L_1(\alpha)$ is the **First Lyapunov Coefficient**. The **critical** normal form coefficient $L_1(0)$ determines the direction of the bifurcation.

Example 1.1. We consider the system

$$\ddot{x} + \dot{x}^3 - 2\alpha\dot{x} + x = 0.$$

Setting $y = -\dot{x}$, we get a system of two differential equations, namely

$$\begin{cases} \dot{x} &= -y, \\ \dot{y} &= x - 2\alpha y - y^3, \end{cases}$$

The system undergoes a Hopf bifurcation at $\alpha = 0$. For the above system, we have $L_1(0) = -\frac{3}{8}$, so at criticality, the system is equivalent to

$$\dot{z} = z\left(\beta(0) + i - \frac{3}{8}|z|^2\right).$$

Figure 1.2 shows the change of the phase portrait as function of the parameter in the two cases. If $L_1 < 0$, a stable focus changes into a stable cycle and if $L_1 > 0$, an unstable focus changes into an unstable cycle.

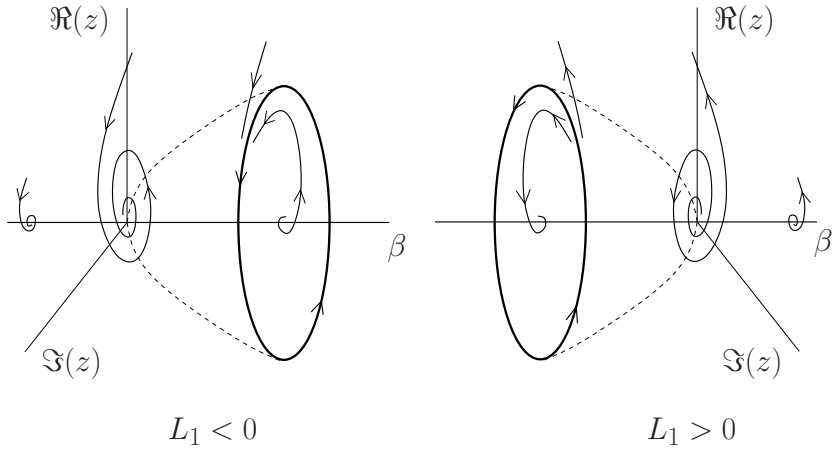


FIGURE 1.2: *A supercritical Hopf bifurcation for $L_1 < 0$ (left) and a subcritical Hopf bifurcation for $L_1 > 0$ (right).*

It is well-known that "simple" bifurcations can already occur in low-dimensional systems of ODEs: one dimension suffices for fold and two dimensions suffice for Hopf bifurcations. Of course, these bifurcations also occur in higher dimensional settings. It's a remarkable fact that in this case, these bifurcations occur in "essentially" the same way as in the lower dimensional case. For a generic n-dimensional system exhibiting a fold or Hopf bifurcation, there exist certain parameter-dependent invariant manifolds, respectively one- and two-dimensional, called **center manifolds**. On these manifolds, the bifurcation takes place, while the behaviour off the manifolds is somehow "trivial". (See [11], Chapter 5.)

For instance, the center manifold for the Hopf bifurcation is twodimensional. What it looks like for a threedimensional system can be seen in Figure 1.3. We see that the bifurcation "happens" on the twodimensional surface; all other orbits simply converge to the center manifold.

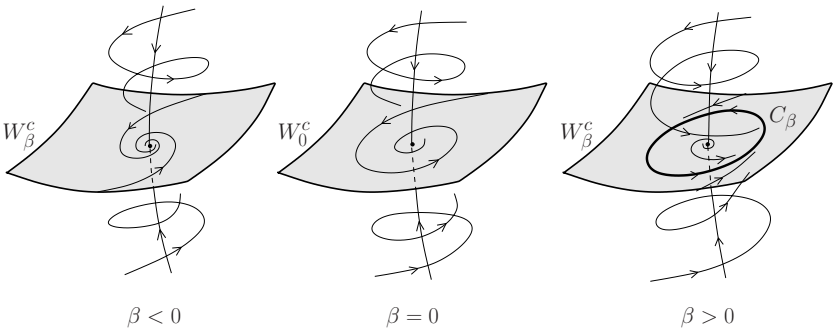


FIGURE 1.3: *The Hopf bifurcation in a threedimensional system takes place on a twodimensional center manifold.*

If a system has more than one parameter, then starting at a Hopf bifurcation point, it is possible to vary a second parameter to obtain an entire *branch* of Hopf bifurcations. As both parameters vary, so-called codimension 2 bifurcations can occur. If L_1 becomes exactly 0, we have a Generalized Hopf or Bautin bifurcation; if in addition to the imaginary pair, a real eigenvalue crosses the imaginary axis, we have a Zero-Hopf or Fold-Hopf bifurcation; and if a second complex pair of eigenvalues crosses the imaginary axis, we have a Double Hopf or Hopf-Hopf bifurcation. These bifurcations also take place on center manifolds and have normal forms.

In this thesis, we investigate a broader class of differential equations, namely Delayed Differential Equations (DDEs). These can be used to model a broader spectrum of phenomena. In a DDE, this derivative may depend on the entire "history" of x. A non-rigorous definition of an (autonomous) DDE might therefore read as follows: a DDE is an equation of the form

$$\dot{x}(t) = f(x_t, \alpha),$$

where x_t represents the trajectory of the solution in the past.

Example 1.2. Suppose that our DDE reads

$$\dot{x}(t) = ax(t-1).$$

In other words, the derivative depends only on the value of x one second ago (if that's the unit of t). Now suppose $x(t) \in \mathbb{R}$. Then if an initial condition $\phi : [-h, 0] \to \mathbb{R}$ is given, we can solve the DDE explicitly by integration for $t < 1$:

$$x(t) = a \int_{\vartheta=0}^{t} \phi(\vartheta - 1)\, ds + C.$$

Now, in DDEs depending on a parameter, there are bifurcations as well. They also take place on center manifolds and they have normal forms, of which the critical normal form coefficients can be computed. The difference with the ODE case is that it takes more advanced

mathematical machinery to describe the bifurcations and compute the coefficients. Therefore, the first aim of this thesis is to present an overview of the relevant theory.

1.2 Numerical bifurcation analysis of DDEs

The second, and main aim of this thesis lies in the automatic computation of the critical normal form coefficients. For ODEs, there is standard software available to plot orbits, make bifurcation diagrams and compute the normal form coefficients (e.g. MATCONT). The software for DDEs is in a less polished state. It can be roughly divided into two types: some focus on plotting orbits (otherwise known as integration or simulation) and others focus on bifurcation analysis, i.e. computing bifurcation curves. We are primarily interested in the second type; simulation of DDEs is robustly done by Matlab's internal `dde23` routine. We provide a short overview of existing bifurcation analysis software here. A compilation of many DDE software packages can be found in [12].

XPP (X-window PhasePlane, see [6], [5]) by Bard Ermentrout is a software package that can analyze ODEs, DDEs and other dynamical problems. It is primarily intended to run simulations, but it can also be used to compute equilibria, their stability, and their invariant manifolds. It can handle DDEs with several discrete delays. The author has combined its functionality with AUTO, which is a bifurcation analysis tool for ODEs and maps.

TRACE-DDE (Tool for Robust Analysis and Characteristic Equations of Delay Differential Equations, see [1] and [2]), by D. Breda, S. Maset and R. Vermiglio, is a Matlab GUI package for numerical stability analysis of linear autonomous systems of DDEs with several discrete and/or distributed delays. It allows for the numerical computation of the characteristic roots and it can perform two-parameter robust stability analysis, producing so-called stability charts, i.e. sets of asymptotically stable/unstable regions in the parameters plane.

KNUT (see [13]) is a program that can continue periodic orbits in DDEs with time-dependent discrete delays. It can detect codimension 1 bifurcations (branching points, Neimark-Sacker, period doubling and fold) and continue bifurcation curves emanating from them. Equilibria of autonomous systems can also be continued as constant periodic orbits with a small, fixed period. In this case the Hopf bifurcation is detected as Neimark-Sacker bifurcation, and the periodic solution, emanating from the Hopf point, can be continued by switching to the periodic solution branch. The software uses orthogonal collocation to discretize periodic orbits. For continuation, the pseudo arclength method is used. Codimension-one bifurcations are computed using test functionals and the arising bordered sparse linear systems are solved using the BEMW method developed by Govaerts and Pryce [9].

Lastly, we have a Matlab package called DDE-BIFTOOL (see [4]). Version 2.00 of this program is able to produce bifurcation diagrams of DDEs, including the continuation of steady state, fold and Hopf points, as well as periodic and homoclinic orbits. However, it is not able to *detect* these bifurcations.

As one can see, KNUT and DDE-BIFTOOL are the programs of choice for bifurcation analysis in the spirit of MATCONT. Both have drawbacks: KNUT can only artificially continue equilibria and DDE-BIFTOOL can't detect bifurcations. What both can't do is detect codimension 2 bifurcations and compute normal form coefficients associated with any bifurcation. They also don't have the built-in ability to run simulations.

For this thesis, we have expanded DDE-BIFTOOL. We have added the functionality mentioned

above for the detection of Hopf bifurcations and its corresponding codimension 2 bifurcations. We have not added the same functionality for the fold and cusp bifurcations, because all relevant bifurcation can be obtained from analyzing the system without delays as a standard ODE.

1.3 Structure of this thesis

In the first three chapters, we will focus on the theory of DDEs. We start out with the functional analytic background of DDEs in Chapter 2. We then turn our attention to the analysis of bifurcations in DDEs in Chapter 3. In Chapter 4, we describe what the normal form of a DDE looks like and how to obtain its coefficients, and present a concrete derivation of the Hopf critical normal form coefficients. This chapter ends formulas that are nearly concrete enough to implement into a software package.

In the last two chapters, we focus on the numerics and DDE-BIFTOOL. In Chapter 5, we describe the new features of DDE-BIFTOOL and how they were implemented. In closing, Chapter 6 contains some example systems and the code that was used to analyze them. These chapters are intended as an addition to the already available DDE-BIFTOOL manual (so a nodding acquaintance with the use of DDE-BIFTOOL is presumed).

Chapter 2

Delay Differential Equations and Sun-Star calculus

In this chapter, we present the functional analytic background needed to understand the analysis of Delay Differential Equations. Proofs of the Theorems and Propositions can be found in Appendix II of [3].

2.1 Definition of a DDE

Naturally, we begin with a definition of a Delay Differential Equation (DDE). We start with a precise definition of "the past":

Definition 2.1. The **history** of a function $x : \mathbb{R} \to \mathbb{R}^n$ at time t is the function x_t given by

$$x_t : [-h, 0] \to \mathbb{R}^n, \quad x_t(\vartheta) \equiv x(t + \vartheta),$$

for $h \in \mathbb{R}$.

As you can see, the parameter h is in general not specified, but for a given discussion, it is assumed to be fixed.

We can now formally define a DDE.

Definition 2.2. An (autonomous) **Delay Differential Equation** (DDE) is an equation of the form

$$\dot{x}(t) = f(x_t, \alpha) \quad (t \geqslant 0), \tag{2.1}$$

where $x : [-h, \infty) \to \mathbb{R}$, $f : C([-h, 0], \mathbb{R}^n) \times \mathbb{R}^m \to \mathbb{R}^n$, $\alpha \in \mathbb{R}^m$, the dot represents the derivative with respect to t, and x_t is the history of x. Given an initial condition $\phi \in C([-h, 0], \mathbb{R}^n)$, a function $x : [-h, t_0) \to \mathbb{R}^n$ is a solution of the DDE if $x \in C([-h, t_0)) \cap C^1([0, t_0))$, it satisfies (2.1) on $[0, t_0)$, and

$$x_0 = \phi.$$

Note that in Definition 2.2, we assume x_t to be continuous. Other conditions on the continuity and differentiability could be imposed as well (e.g. C^1).

Intuitively, the general procedure of solving a DDE now works as follows. Given an initial condition $\phi : [-h, 0] \to \mathbb{R}^n$, we use the DDE (2.1) to *extend* this function along the real axis for $t > 0$. However, we do not need to do this all at once; it suffices to know how to extend the solution a little bit, say until $t_0 > 0$. If this extension is successful, we can shift the part of the solution on $[-h + t_0, t_0]$ back to $[-h, 0]$ and repeat the extension process. See Figure 2.1 for a visualization of the procedure.

This way, we get an *orbit in a space of functions*, namely $C([-h, 0])$: for every $t \geqslant 0$, there is a point in the function space, representing the shifted-back function at time t. This is why we turn to the functional analysis of infinite-dimensional Banach spaces to analyze DDEs.

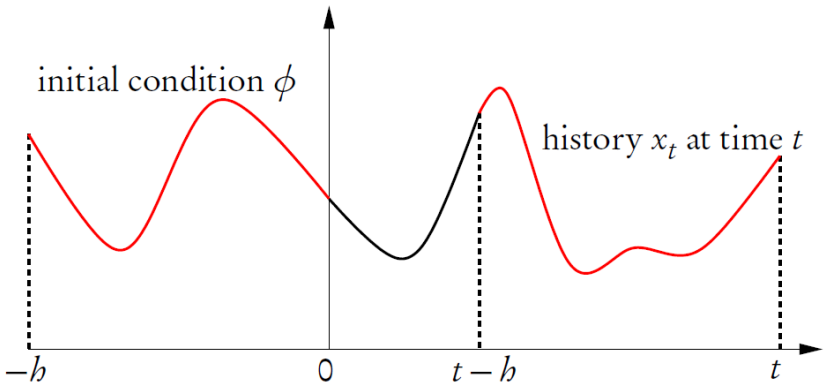


FIGURE 2.1: *The extension procedure of solving a DDE.*

2.2 Semigroups and generators

We analyze the existence and uniqueness of solutions to DDEs in the framework of functional analysis, more specifically the theory of strongly continuous semigroups and their generators. It will turn out that the spectrum of the generator of the semigroup corresponding to the DDE determines the nature of the bifurcations.[1]

Definition 2.3. Let X be a complex Banach space and let, for each $t \geqslant 0$, $T(t) : X \to X$ be a bounded linear operator. Then the family $\{T(t)\}_{t \geqslant 0}$ is called a **strongly continuous semigroup**, or a C_0**-semigroup**, if the following three properties hold:

(1) $T(0) = I$;

(2) $T(t)T(s) = T(t+s)$, for $t, s \geqslant 0$;

(3) for all $x \in X$, $\lim_{t \downarrow 0} \|T(t)x - x\| = 0$.

The interpretation of these operators is that a specific $T(t)$ maps an initial condition to the solution at time t. This is why it's also commonly called a **semiflow**. Properties (1) and (2)

[1] In fact, the derivative matrix in an ODE setting is just a specific instance of a generator.

are what makes $\{T(t)\}_{t\geqslant 0}$ into a semigroup. Property (3), the strong continuity, is of another category: this ensures that orbits $\{T(t)x : t \geqslant 0\}$ are continuous with respect to the (norm) topology on X.

Definition 2.4. The **infinitesimal generator** A of $\{T(t)\}_{t\geqslant 0}$ is defined by

$$\mathcal{D}(A) = \left\{ x : \lim_{h\downarrow 0} \frac{1}{h}(T(h)x - x) \text{ exists} \right\},$$
$$Ax = \lim_{h\downarrow 0} \frac{1}{h}(T(h)x - x).$$

So A is simply the derivative of $T(t)$ at $t = 0$. In general, A is an unbounded operator, which is why we have to specify its domain so awkwardly.

The idea is that A captures the differential problem (the DDE, in our case) and that the corresponding semigroup $T(t)$ supplies solutions to the problem. In systems of ODEs, A is simply a matrix and the semigroup is the matrix exponential $T(t) = e^{tA}$. When working with the abstract objects, this intuition comes in handy.

Proposition 2.5 lists a few properties of a semigroup and its generator. Note that the integrals here are Riemann integrals (generalized to obtain values in a Banach space).

Proposition 2.5 ([3], Appendix II, Propositions 1.1, 1.4, 1.5, 1.6)**.** Let $\{T(t)\}_{t\geqslant 0}$ be a strongly continuous semigroup and A its infinitesimal generator. Then the following properties hold.

(1) $t \mapsto T(t)x$ is continuous from $\mathbb{R}_+$ into X;

(2) for any $x \in X$ and arbitrary $t > 0$,

$$\int_0^t T(s)x\,ds \in \mathcal{D}(A) \quad \text{and} \quad A\left(\int_0^t T(s)x\,ds\right) = T(t)x - x;$$

(3) for arbitrary $t > 0$, $\mathcal{D}(A)$ is $T(t)$-invariant and for $x \in \mathcal{D}(A)$,

$$\lim_{h\downarrow 0} \frac{d}{dt} \int_t^{t+h} T(s)x\,ds = AT(t)x = T(t)Ax;$$

(4) $\mathcal{D}(A)$ is dense in X;

(5) A is closed.

2.3 Dual spaces and adjoint semigroups

Recall that, given a Banach space X, the **dual space** X^* is the set of all continuous linear functionals $X :\to \mathbb{R}$. We use a special notation for the application of a functional $x^* \in X^*$ to

an element $x \in X$, namely the **pairing**

$$\langle x^*, x\rangle \equiv x^*(x).$$

X^* is itself a Banach space, equipped with the norm

$$\|x^*\| = \sup_{\|x\|\leqslant 1} |\langle x^*, x\rangle|.$$

The dual space gives rise to the notion of an adjoint operator.

Definition 2.6. If A is a densely defined operator $X \to X$, then the **adjoint operator** $A^* : X^* \to X^*$ is defined as follows: the domain is given by

$$\mathcal{D}(A^*) = \{x^* \in X^* : \exists\, y^* \in X^* \text{ such that } \langle x^*, Ax\rangle = \langle y^*, x\rangle, \text{ for all } x \in \mathcal{D}(A)\}$$

and the action by

$$A^* x^* = y^*.$$

The notation should remind the reader of a Hilbert space H, where we have $H^* = H$, the pairing is the inner product, and the adjoint is simply defined by $\langle A^* x, y\rangle = \langle x, Ay\rangle$. As in a Hilbert space, it also holds that $\|A\| = \|A^*\|$.

When X is a function space, working with the pairing directly is often very inconvenient, and thinking about X^* may become confusing because its members are functions working on functions. It would be more convenient if we could think of the elements of X^* as functions as well and of the pairing $\langle\cdot,\cdot\rangle$ as an operation combining a function from X and a "function" from X^* (reinforcing the inner product intuition). In fact, this is possible: if $X = C([-h, 0], \mathbb{C})$ (as will be the case in our applications), then we can identify X^* with a space of **normalized bounded variation** functions on $\mathbb{R}$:

$$\mathrm{NBV} = \left\{ \begin{array}{c} f : \mathbb{R} \to \mathbb{R} : \sup_{P(t)} \sum_{j=1}^{N} \left| f(\sigma_j) - f(\sigma_{j-1}) \right| \text{ is bounded in } t, \\ f(\vartheta) = 0 \text{ for } \vartheta \leqslant 0, \quad f(\vartheta) = f(h) \text{ for } \vartheta \geqslant h \end{array} \right\},$$

where $P(t)$ denotes a partition $0 = \sigma_0 < \sigma_1 < \cdots < \sigma_N = t$ of $[0, t]$. Note that f needs to be defined only on $[0, h]$: it is 0 on $(-\infty, 0]$ and equal to $f(h)$ on $[h, \infty)$. (This is simply a technical trick.)

If we adopt this convention, the pairing between $f \in X^*$ and $\phi \in X$ can also be written as

$$\langle f, \phi\rangle = \int_0^\infty df(\vartheta)\, \phi(-\vartheta),$$

where the integral is a **Riemann-Stieltjes integral**. For more information about Riemann-Stieltjes integrals, one can have a look at Appendix I of [3]. Luckily, in the same work it is shown that one can also evaluate these integrals as Lebesgue integrals, interpreting $df(\vartheta)$ as a measure corresponding to the function f. We will show an example later.

Generally, ϕ will only be defined on $[-h, 0]$, which renders the limit ∞ occurring in the integral potentially dangerous, but since f is constant outside of $[0, h]$, this does not influence the value of the integral. (Another technical trick.)

For other spaces X, other representations for X^* and the pairing are possible. We reproduce the very nice compilation Sebastiaan Janssens has made in [10] here as Table 2.1.

The pairing between the original and the dual space also leads to a new form of convergence, namely weak convergence and weak* convergence. Actually, to do things thoroughly, we should start with the weak topology on X and the weak* topology on X^*. To avoid an abstract discussion on this topic, we simply make the following definition.

Definition 2.7. A function $f : X \to X$ is said to **converge weakly** to $y \in X$, or to have the **weak limit** y, as $x \to x_0$ if

$$\lim_{x \to x_0} \langle x^*, f(x) \rangle = y \quad \text{for all} \quad x^* \in X^*.$$

This is also denoted by weak-$\lim_{x \to x_0} f(x) = y$ or $f(x) \rightharpoonup y$ as $x \to x_0$.

A function $f : X^* \to X^*$ is said to have the **weak* limit** $y^* \in X^*$ as $x^* \to x_0^*$ if

$$\lim_{x^* \to x_0^*} \langle f(x^*), x \rangle = y^* \quad \text{for all} \quad x \in X.$$

This is also denoted by weak*-$\lim_{x^* \to x_0^*} f(x^*) = y^*$ or $f(x^*) \overset{*}{\rightharpoonup} y^*$ as $x^* \to x_0^*$.

Whenever we refer to properties such as weak-dense, weak*-closed et cetera, you can take the normal definition of the property and replace the normal limit by the appropriate weak limit.

Returning to semigroups, we can ask the question whether a given strongly continuous semigroup $\{T(t)\}_{t \geqslant 0}$ on X gives rise to an adjoint strongly continuous semigroup $\{T^*(t)\}_{t \geqslant 0}$ on X^*, where $T^*(t) \equiv [T(t)]^*$.[2] The answer, unfortunately, is a partial no.

Theorem 2.8 ([3], Appendix II, Theorem 3.5)**.** Suppose $\{T(t)\}_{t \geqslant 0}$ is a strongly continuous semigroup on X and let $\{T^*(t)\}_{t \geqslant 0}$ be its adjoint group on X^*. Then the following is true:

(1) $\{T^*(t)\}_{t \geqslant 0}$ is a weak* continuous semigroup, i.e.

   (a) $T^*(0) = I$,

   (b) $T^*(t+s) = T^*(t) + T^*(s)$,

   (c) $t \mapsto T^*(t)x^*$ is weak* continuous from $\mathbb{R}_+$ into X^*.

(2) A^* is the weak* generator of $\{T^*(t)\}_{t \geqslant 0}$, i.e.

$$\text{weak}^*\text{-}\lim_{h \downarrow 0} \frac{1}{h}(T^*(h)x^* - x^*) = y^*$$

   if and only if $x^* \mathcal{D}(A^*)$ and $y^* = A^* x^*$.

(3) $\mathcal{D}(A^*)$ is $T^*(t)$ invariant and $A^* T^*(t) = T^*(t) A^*$ on $\mathcal{D}(A^*)$.

[2] It will turn out that we need this property to analyze DDEs and their bifurcations, although at this point in time, it seems unclear why anyone should care.

In order to get a good theory of DDEs, we would like the adjoint semigroup to be strongly continuous as well. We can achieve something of the kind by restricting the domain.

2.4 The sun subspace and the sun-star dual

In general, the adjoint semigroup of a strongly continuous semigroup on X is not strongly continuous on the dual space X^*. However, there might be a *subspace* of X^* on which it *is* strongly continuous. This leads to the following definition:

Definition 2.9 ([3], Appendix II, Definition 3.7)**.** Suppose $\{T(t)\}_{t\geqslant 0}$ is a strongly continuous semigroup on a Banach space X and let $\{T^*(t)\}_{t\geqslant 0}$ be its adjoint group on X^*. We define the **sun dual** $X^\odot \subset X^*$ (with respect to $\{T(t)\}_{t\geqslant 0}$) by

$$X^\odot := \left\{ x^* \in X^* : \lim_{h\downarrow 0} \|T^*(h)x^* - x^*\| = 0 \right\}$$

and call the semigroup given by the restricted operators

$$T^\odot(t) := T^*(t)|_{X^\odot}, \quad t \geqslant 0$$

the **sun dual semigroup**.

The next logical question is: is this a useful definition? Isn't $X^\odot$ very small or even emtpy, and what about the generator of the sun dual semigroup? The following theorem shows that $X^\odot$ is in fact well-behaved, quite large, and leads to a "nice" generator $A^\odot$.

Theorem 2.10 ([3], Appendix II, Proposition 3.8 and Theorem 3.10)**.** Suppose $X^\odot$ is the sun dual space for a strongly continuous semigroup $\{T(t)\}_{t\geqslant 0}$ on X. Then the following properties hold:

(1) $X^\odot$ is a norm-closed, $T^*(t)$-invariant subspace of X^*;

(2) $X^\odot = \overline{\mathcal{D}(A^*)}$ (the closure of the domain of A^*);

(3) The generator of $\left\{T^\odot(t)\right\}_{t\geqslant 0}$, denoted by $A^\odot$, is the part of A^* in $X^\odot$, i.e.

$$\mathcal{D}\left(A^\odot\right) = \left\{x^* \in \mathcal{D}(A^*) : A^*x^* \in X^\odot\right\}, \quad \text{and} \quad A^\odot x^* = A^*x^* \quad \text{for} \quad x^* \in \mathcal{D}\left(A^\odot\right).$$

Now we're going to perform a trick of which the significance (and usefulness) will become apparent later. The reasoning goes like this: if we have $\left\{T^\odot(t)\right\}_{t\geqslant 0}$ on $X^\odot$, we have a strongly continuous semigroup on a Banach space. So we can take the adjoint semigroup again, leading to a semigroup $\left\{T^{\odot *}(t)\right\}_{t\geqslant 0}$ on $X^{\odot *}$. However, as we have seen, this semigroup isn't necessarily strongly continuous. But we know the solution to this problem already: we take the sun dual again! This leads to a strongly continuous semigroup $\left\{T^{\odot\odot}(t)\right\}_{t\geqslant 0}$ on a corresponding double sun dual space $X^{\odot\odot}$.

Space	Representation	Pairing
X X^*	$\phi \in C([-h,0],\mathbb{R}^n)$ $f \in \mathrm{NBV}\big([0,h],\underline{\mathbb{R}}^n\big)$	$\langle f,\phi\rangle = \int_0^h df(\vartheta)\,\phi(-\vartheta)$
$X^\odot$ $X^{\odot *}$	$(c,g) \in \underline{\mathbb{R}}^n \times L^1\big([0,h],\underline{\mathbb{R}}^n\big)$ $(a,\phi) \in \mathbb{R}^n \times L^\infty([-h,0],\mathbb{R}^n)$	$\langle(\alpha,\phi),(c,g)\rangle = c\alpha + \int_0^h g(\vartheta)\,\phi(-\vartheta)\,d\vartheta$
X $X^\odot$	$\phi \in C([-h,0],\mathbb{R}^n)$ $(c,g) \in \underline{\mathbb{R}}^n \times L^1\big([0,h],\underline{\mathbb{R}}^n\big)$	$\langle(c,g),\phi\rangle = c\phi(0) + \int_0^h g(\vartheta)\,\phi(-\vartheta)\,d\vartheta$

TABLE 2.1: *Representations for the abstract spaces X, X^*, $X^\odot$ and $X^{\odot *}$ for the case of the semigroup $\{T(t)\}_{t\geqslant 0}$ associated with the linear equation (2.4). If a space is underlined, its components should be thought of as row vectors. Also indicated are the dual pairings used in the computation of normal form coefficients. (This is Table 2.1 in [10].)*

Of course, one would expect to get back in the original space after taking the dual twice. In general, this is not the case; spaces for which we do have $X^{**} = X$ are called reflexive. In this case, after taking the dual for the second time, we end up in the (potentially) smaller space $X^{\odot *}$. There is a natural embedding $j : X \to X^{\odot *}$ and we certainly have $j(X) \subset X^{\odot\odot}$, but in general this need not be an equality. If it is, we reserve a special name for it:

Definition 2.11 ([3], Appendix II, Definition 3.20)**.** Suppose $X^{\odot\odot}$ is the double sun dual of X, with respect to some strongly continuous semigroup $\{T(t)\}_{t\geqslant 0}$. Then X is called **$\odot$-reflexive** (i.e. **sun-reflexive**) with respect to $\{T(t)\}_{t\geqslant 0}$ if $j(X) = X^{\odot\odot}$, where j is the embedding of X into $X^{\odot *}$.

Now we have defined all relevant spaces in the sun-star calculus, we can list their representations as spaces of functions. See Table 2.1 (due to [10]).

2.5 The shift semigroup

Up till now, our discussion on the various adjoint semigroups was rather abstract. We're going to see them in action now, resulting from the analysis of a simple system. To reiterate: solving a DDE amounts to extending the initial value a tiny bit, shifting the result back to $[-h,0]$, and repeat the process. The formalism of the sun and sun-star dual groups will help us do this in an elegant fashion, allowing us to treat the shifting and the extension seperately, their properties being captured in different operators. In fact, the problem below will be all that's needed to understand the shifting part.

We choose $X = C([-h,0],\mathbb{C})$ and we study the **trivial DDE**

$$\begin{cases} \dot{x}(t) = 0 & \text{for } t \geqslant 0, \\ x(\vartheta) = \phi(\vartheta) & \text{for } -h \leqslant \vartheta \leqslant 0, \end{cases} \tag{2.2}$$

where $\phi \in X$ is some initial condition. We can explicitly determine the solution of this equation: it is

$$x(t) = \begin{cases} \phi(t) & \text{for } -h \leqslant t \leqslant 0, \\ \phi(0) & \text{for } t \geqslant 0, \end{cases}$$

i.e. it extends ϕ in a continuous way to the whole of $[-h, \infty)$ with a constant function.

Now, this equation gives rise to a strongly continuous semigroup $\{T_0(t)\}_{t\geqslant 0}$, which can also be given an explicit formula:

$$x_t = (T_0(t)\phi)(\vartheta) = \begin{cases} \phi(t+\vartheta) & \text{if } -h \leqslant t+\vartheta \leqslant 0, \\ \phi(0) & \text{if } t+\vartheta \geqslant 0. \end{cases}$$

So the operator $T_0(t)$ maps the initial state ϕ at $t = 0$ onto the state x_t at time t. This group is called the **shift semigroup**.

Of course, we can now explicitly determine the generator A_0 as well.

Lemma 2.12 ([3], Chapter II, Lemma 2.1)**.** The generator of $\{T_0(t)\}_{t\geqslant 0}$ is given by

$$\mathcal{D}(A_0) = \left\{\phi : \dot{\phi} \in C([-h,0],\mathbb{C}),\ \dot{\phi}(0) = 0\right\}, \quad \text{and} \quad A_0\phi = \dot{\phi}. \tag{2.3}$$

Equation (2.3) poses a fundamental problem: *the rule for extending the initial condition beyond* $[-h, 0]$ *is incorporated into the domain of* A_0 *instead of into its action*, the rule being that $\dot{\phi}(0) = 0$. In fact, we will see that for a *general* DDE problem (i.e. (2.1)), all information about the extension is contained in the domain, and the actual *action* of the generator is simply $A\phi = \dot{\phi}$.

This is not very nice to work with, of course. Fortunately, it turns out that on $X^{\odot *}$, we can capture both the extension and the shifting part in the action of the generator! This is why we need the whole sun-star machinery.

So our next aim is to find $A^{\odot *}$, and see what it looks like for the shift semigroup. We start with determining A^* explicitly. Using the pairing from Table 2.1, we see that

$$\begin{aligned} \langle f, T_0(t)\phi\rangle &= \int_0^\infty df(\vartheta)\,(T_0(t)\phi)(-\vartheta) = \int_0^t df(\vartheta)\,\phi(0) + \int_t^\infty df(\vartheta)\,\phi(t-\vartheta) \\ &= f(t)\phi(0) + \int_0^\infty f(t+\sigma)\,\phi(-\sigma)\,d\sigma. \end{aligned}$$

This leads to the following conclusion.

Lemma 2.13. The adjoint shift semigroup $\left\{T_0^*(t)\right\}_{t\geqslant 0}$ is given by

$$\left(T_0^*(t)f\right)(\vartheta) = f(t+\vartheta) \quad \text{for } \vartheta > 0.$$

The domain and action of A_0^* can be explicitly computed as well. This calculation is rather involved, so we only state the result. In a sense, A_0^* returns the derivative of its operand.

Theorem 2.14 ([3], Chapter II, Theorem 5.1)**.** The domain $\mathcal{D}\left(A_0^*\right)$ of the generator of the adjoint shift semigroup $\left\{T_0^*(t)\right\}_{t\geqslant 0}$ consists of those functions f for which

$$f(\vartheta) = f\left(0^+\right) + \int_0^\vartheta g(\sigma)\,d\sigma,$$

where $g \in \mathrm{NBV}$ with $g(h) = 0$ and $f\left(0^+\right) \equiv \lim_{\vartheta\downarrow 0} f(\vartheta)$. For $f \in \mathcal{D}\left(A_0^*\right)$, we have

$$A_0^* f = g.$$

The sun dual $X^\odot$ is now simply the closure of $\mathcal{D}\left(A_0^*\right)$ (see Proposition 2.10). This leads to the following result.

Theorem 2.15 ([3], Chapter II, Theorem 5.2)**.** The sun dual of $X = C([-h,0],\mathbb{C})$ with respect to the shift semigroup is given by

$$X^\odot = \left\{ \begin{array}{c} f \in \mathrm{NBV} : f(t) = c + \displaystyle\int_0^t g(\vartheta)\,d\vartheta \text{ for } t > 0, \\ \text{where} \quad c \in \mathbb{C} \quad \text{and} \quad g \in L^1 \text{ with } g(\vartheta) = 0 \text{ a.e. for } \vartheta \geqslant h \end{array} \right\}$$

As was already visible in Table 2.1, in light of Theorem 2.15, elements of the space $X^\odot$ are completely specified by giving a $c \in \mathbb{C}$ and a $g \in L^1$ satisfying $g(\vartheta) = 0$ almost everywhere for $\vartheta \geqslant h$. In other words:

Proposition 2.16. The space $X^\odot$ (with respect to the shift semigroup) is *isometrically isomorphic* to $\mathbb{C} \times L^1([0,h],\mathbb{C})$, if we equip this space with the norm

$$\|(c,g)\| = |c| + \|g\|_{L^1}.$$

Again, to make sense of various integrals, we adopt the convention that L^1 functions on $[0,h]$ are extended to (h,∞) by zero.

Knowing $X^\odot$, it is possible to determine a concrete formula for the sun dual shift semigroup $\left\{T_0^\odot(t)\right\}_{t\geqslant 0}$ and its generator $A_0^\odot$. Recall that AC, appearing in the description of the domain of $A_0^\odot$, is the space of **absolutely continuous functions**.

Theorem 2.17 ([3], Chapter II, Theorem 5.3)**.** Using the representation of Proposition 2.16, the action of the sun dual shift semigroup $\left\{T_0^\odot(t)\right\}_{t\geqslant 0}$ is given by

$$T_0^\odot(t)(c,g) = \left(c + \int_0^t g(\sigma)\,d\sigma,\ G\right), \quad \text{where} \quad G(\vartheta) = g(t+\vartheta).$$

The domain of the generator can be explicitly written as

$$\mathcal{D}\left(A_0^{\odot}\right) = \left\{ f : f(t) = c + \int_0^t g(\vartheta)\, d\vartheta \text{ for } c \in \mathbb{C} \text{ and } g \in \mathrm{AC}(0,h), g(\vartheta) = 0 \text{ for } \vartheta \geqslant h \right\},$$

and in this representation, we have

$$A_0^{\odot} f = g.$$

In the alternative representation, we have

$$\mathcal{D}\left(A_0^{\odot}\right) = \{(c,g) : c \in \mathbb{C} \text{ and } g \in \mathrm{AC}(0,h), g(\vartheta) = 0 \text{ for } \vartheta \geqslant h\} \quad \text{and}$$
$$A_0^{\odot}(c,g) = \left(g\left(0^+\right), \dot{g}\right).$$

This concludes the analysis of the relevant sun dual objects; we can now turn our attention to the sun-star dual objects, on the space $X^{\odot *}$. For this, it is necessary to have an alternative representation of $X^{\odot *}$ as well.

Proposition 2.18. The space $X^{\odot *}$ (with respect to the shift semigroup) is isometrically isomorphic to $\mathbb{C} \times L^{\infty}([0,h],\mathbb{C})$, if we equip this space with the norm

$$\|(\alpha,\phi)\| = \sup\left\{|\alpha|, \|\phi\|_{\infty}\right\}.$$

Theorem 2.19 ([3], Chapter II, Theorem 5.5)**.** Using the representation of Proposition 2.18, the action of the sun-star dual shift semigroup $\left\{T_0^{\odot *}(t)\right\}_{t \geqslant 0}$ is given by

$$T_0^{\odot *}(t)(\alpha,\phi) = \left(\alpha, \phi_t^{\alpha}\right),$$

where

$$\phi_t^{\alpha}(\vartheta) \equiv \begin{cases} \phi(t+\vartheta) & \text{if } t+\vartheta \leqslant 0, \\ \alpha & t+\vartheta > 0. \end{cases}$$

The generator is given by

$$\mathcal{D}\left(A_0^{\odot *}\right) = \{(\alpha,\phi) : \phi \text{ is Lipschitz-continuous with Lipschitz constant } \alpha\}, \quad \text{and}$$
$$A_0^{\odot *}(\alpha,\phi) = \left(0, \dot{\phi}\right).$$

In words, $T_0^{\odot *}(t)$ first extends ϕ by the constant value α for $\vartheta > 0$ and then shifts it over a time t. As you can see, we have solved the big problem that was present on X: the domain of $A_0^{\odot *}$ no longer refers to any specifics of the DDE, only its action does.

We're almost there now. Having all the objects on the sun-star dual space, we now turn our attention to the double sun objects.

Theorem 2.20. The double sun dual $X^{\odot\odot}$ with respect to the shift semigroup $\{T_0(t)\}_{t\geqslant 0}$ is given by

$$X^{\odot\odot} = \overline{\mathcal{D}\left(A_0^{\odot *}\right)} = \{(a,\phi) : \phi \in L^\infty([-h,0],\mathbb{C}) \cap C([-h,0],\mathbb{C}), \phi(0) = \alpha\}.$$

Note that X is sun-reflexive: a function $\phi \in X = C([-h,0],\mathbb{C})$ is represented by the couple $(\phi(0),\phi)$ in $X^{\odot *}$, and therefore it is in $X^{\odot\odot}$ as well. We could write $j(\phi) = (\phi(0),\phi)$ and $X^{\odot\odot} = j(X)$, but it is more convenient to drop the j altogether. Therefore, in the remainder of this thesis, we will mix and match the two representations.

In this section, we have merely taken the shift semigroup $\{T_0(t)\}_{t\geqslant 0}$, corresponding to the trivial DDE (2.2), defined on X, and by a long-winded functional analytic process turned it into a copy of itself, $\left\{T_0^{\odot\odot}(t)\right\}_{t\geqslant 0}$ defined on $X^{\odot\odot} = X$. But we have gained something new: we can think of $X^{\odot\odot}$ as being embedded in a larger space $X^{\odot *}$, on which we have a more general semigroup $\left\{T_0^{\odot *}(t)\right\}_{t\geqslant 0}$. In the next section, we will show that it is worthwhile to have $X^{\odot *}$ when working with a general DDE.

2.6 Sun-star calculus for linear DDEs

We now turn our attention to more general DDEs, namely linear ones. If a DDE is given by

$$\dot{x}(t) = Lx_t,$$

with L a linear function $C([-h,0]) \to \mathbb{R}^n$, it's possible to rewrite this as

$$\dot{x}(t) = \int_0^h d\zeta(\vartheta)\, x(t-\vartheta). \tag{2.4}$$

This is also called a **retarded functional delay equation** (or RFDE). Here, ζ is an $n \times n$ matrix valued function whose entries belong to NBV. Using Table 2.1, we can also write

$$\dot{x}(t) = \langle \zeta, x_t \rangle_n.$$

We now wish to cast this equation into our sun-star framework. As we have stressed before, we should regard the process of finding a global solution as an alternation of extending the current solution and shifting it back to $[-h,0]$. In the previous section, we have seen that the shifting is done by the shift semigroup. Therefore, it's natural to do the following:

$$\frac{d}{dt}x_t = A_0^{\odot *}x_t + Bx_t = \left(A_0^{\odot *} + B\right)x_t, \tag{2.5}$$

where $B : X \to X^{\odot *}$ is defined by

$$B\phi = \left(\langle \zeta, \phi \rangle_n, 0\right).$$

This gives us a differential equation for elements of $X^{\odot *}$, but we can still think of them as being elements of X.

It remains to be proven that $A_0^{\odot *} + B$ is indeed the generator of a strongly continuous semigroup which solves (2.4). We do this by introducing a so-called **variation of constants formula** (2.6).

Theorem 2.21 ([3], Chapter III, Theorem 2.4)**.** Let X be a sun-reflexive Banach space with respect to a strongly continuous semigroup $\{T_0(t)\}_{t\geqslant 0}$. Let $B : X \to X^{\odot *}$ be a bounded operator. There exists a unique strongly continuous semigroup $\{T(t)\}_{t\geqslant 0}$ such that

$$T(t)\phi = T_0(t)\phi + \int_0^t T_0^{\odot *}(t-\tau)B\,T(\tau)\phi\,d\tau =: T_0(t)\phi + U(t)\phi. \tag{2.6}$$

It holds that $\lim_{t\downarrow 0} \|U(t)\| = 0$ and hence $\lim_{t\downarrow 0} \|U^*(t)\| = 0$.

Of course, we think of $\{T_0(t)\}_{t\geqslant 0}$ as the shift semigroup, but the theorem holds for general semigroups. The integral in Theorem 2.21 can be thought of as a *perturbation* of the action of the original semigroup. This perturbation is given the name U.

We now want to determine the generator of $\{T(t)\}_{t\geqslant 0}$, called A, in terms of A_0 and B. From the behaviour of U^* as $t \downarrow 0$, we can see that the map $t \mapsto T^*(t)x^*$ is norm continuous at $t = 0$ if and only if $t \mapsto T_0^*(t)x^*$ is norm continuous at $t = 0$. And hence:

Lemma 2.22 ([3], Chapter III, Lemma 2.6)**.** The subspace $X^\odot$, defined as the sun dual with respect to the unperturbed semigroup $\left\{T_0^*(t)\right\}_{t\geqslant 0}$, is also the subspace of strong continuity for the perturbed semigroup $\{T^*(t)\}_{t\geqslant 0}$, i.e. it is also the sun dual with respect to the perturbed semigroup. In particular, $X^\odot$ is invariant under T^*.

So the perturbation leading us from T_0 to T doesn't change the underlying spaces. This is good to know!

Now, note that since $B : X \to X^{\odot *}$, we have $B^* : X^{\odot **} \to X^*$. Of course, the space $X^{\odot **}$ doesn't seem very nice to work with. Luckily, $X^\odot \subset X^{\odot **}$ (as in general, we can think of a space as being embedded in its second dual), so we can also consider a restricted version of B^*, namely $B^*|_{X^\odot} : X^\odot \to X^*$. In the following, we will use this restricted version and simply denote it by B^*.

In Theorem 2.23, we collect all results we need to know.

Theorem 2.23 ([3], Chapter III, Corollaries 2.8, 2.9, 2.12 and 2.13, Theorem 2.10)**.** If A is the generator of the perturbed semigroup $\{T(t)\}_{t\geqslant 0}$, the following hold:

(1) $\mathcal{D}(A^*) = \mathcal{D}\left(A_0^*\right)$ and $A^* = A_0^* + B^*$.

(2) The strongly continuous semigroup $\left\{T^\odot(t)\right\}_{t\geqslant 0}$ is generated by the operator $A^\odot$ defined by

$$\mathcal{D}\left(A^\odot\right) = \left\{x^\odot \in \mathcal{D}(A^*) : \left(A_0^* + B^*\right)x^\odot \in X^\odot\right\}, \quad \text{with} \quad A^\odot = A_0^* + B^*.$$

(3) The space X is sun-reflexive with respect to the perturbed semigroup.

(4) $\mathcal{D}\left(A^{\odot *}\right)=\mathcal{D}\left(A_0^{\odot *}\right)$ and $A^{\odot *}=A_0^{\odot *}+B$, i.e. $A^{\odot *}(\alpha,\phi)=\left(\langle\zeta,\phi\rangle_n,\dot{\phi}\right)$.

(5) The strongly continuous semigroup $\{T(t)\}_{t\geqslant 0}$ is generated by the operator A defined by

$$\mathcal{D}(A)=\left\{x\in\mathcal{D}\left(A_0^{\odot *}\right):\left(A_0^{\odot *}+B\right)x\in X\right\},\quad \text{with}\quad A=A_0^{\odot *}+B.$$

So, this theorem tells us that indeed, if we have an operator B defining an RFDE as in (2.5), there is a strongly continuous semigroup $\{T(t)\}_{t\geqslant 0}$ solving the equation, having a generator $A=A_0^{\odot *}+B$.

Let's go back to the more concrete setting in which $\{T_0(t)\}_{t\geqslant 0}$ is the shift semigroup, $X=C([-h,0],\mathbb{C}^n)$, $X^*=\mathrm{NBV}([0,h],\mathbb{C}^n)$, and $X^{\odot *}=\mathbb{C}^n\times L^\infty([-h,0],\mathbb{C}^n)$. We take B as in 2.6, i.e. $B\phi=(\langle\zeta,\phi\rangle_n,0)$, where ζ is a matrix valued NBV function.

Theorem 2.24 ([3], Chapter III, Theorem 4.1)**.** Let, with $\{T_0(t)\}_{t\geqslant 0}$ and B as defined above, $\{T(t)\}_{t\geqslant 0}$ be the semigroup defined by the abstract integral equation (2.6). Let $x(\cdot;\phi)$ be the solution of the RFDE

$$\dot{x}(t)=\int_0^h d\zeta(\vartheta)\,x(t-\vartheta),\quad t\geqslant 0, \tag{2.7}$$

with initial condition

$$x(\vartheta)=\phi(\vartheta),\quad -h\leqslant\vartheta\leqslant 0.$$

Then,

$$T(t)\phi=x_t(\cdot;\phi).$$

In other words, if we have (2.7) instead of $\dot{x}(t)=0$, we should put ζ in an operator B and get the semigroup $\{T(t)\}_{t\geqslant 0}$ corresponding to equation (2.6). This will give us the solution to our original DDE 2.7.

Chapter 3

Bifurcations of DDEs

3.1 Linearization near an equilibrium

As we have recalled in the Introduction, when studying an ordinary nonlinear differential equation $\dot{x}(t) = f(x, \alpha)$, we can deduce most relevant properties of the phase portrait by looking at the spectrum of the linearization $Df(x, \alpha)$.

For a DDE, it is possible to use similar techniques. We already mentioned that we need to look at the spectrum of the generator, but this is in general a rather complex object. We will therefore work towards a more straightforward way to obtaining the spectrum. We start by describing what a linearization looks like for DDEs. (For more information, see [10], Chapter 2.)

Suppose we have a general DDE of the form

$$\dot{x}(t) = f(x_t, \alpha). \tag{3.1}$$

We will split it into a linear and a non-linear part in the following way. Suppose we have a stationary solution $x^0(t, \alpha_0) \equiv C$ of (3.1) for a certain parameter value α_0 (this where a bifurcation takes place), so that $f(x_t^0, \alpha_0) = 0$.[1] By a change of coordinates it can always be arranged that $x_t^0 \equiv 0$. We now employ a simple rewriting trick to get

$$\begin{aligned}\dot{x}(t) &= D_1 f(0, \alpha)\, x_t + (f(x_t, \alpha) - D_1 f(0, \alpha)\, x_t) \\ &= \int_0^h d\zeta(\vartheta, \alpha)\, x_t(-\vartheta) + (f(x_t, \alpha) - D_1 f(0, \alpha)\, x_t), \\ &= \int_0^h d\zeta(\vartheta, \alpha)\, x_t(-\vartheta) + g_f(x_t, \alpha),\end{aligned} \tag{3.2}$$

i.e. we define

$$g_f(\phi, \alpha) = f(\phi, \alpha) - D_1 f(0, \alpha)\phi.$$

Here, $\zeta(\cdot, \alpha)$ denotes the $\mathrm{NBV}([0, h], \mathbb{R}^n)$ representation of the partial derivative $D_1 f(0, \alpha)$.

So now we have reduced the problem to the DDE

$$\dot{x}(t) = \int_0^h d\zeta(\vartheta, \alpha)\, x_t(-\vartheta) + g(x_t, \alpha) \quad (t \geqslant 0), \tag{3.3}$$

[1] So, x^0 is constant on its entire domain $[-h, \infty)$, meaning that its history x_t^0 is constant on $[-h, 0]$. The function f only operates on the history.

i.e. (2.4) with an extra general perturbation g. Here, $g : X \times \mathbb{R}^m \to \mathbb{R}^n$ is assumed to be of class C^k for sufficiently high k and is supposed to satisfy

$$g(0,\alpha_0) = 0, \quad D_1 g(0,\alpha_0) = 0 \tag{3.4}$$

for a fixed $\alpha_0 \in \mathbb{R}^m$.

To describe the solutions to this equation, we need some additional machinery. First, we need the standard basis vectors e_j of $\mathbb{R}^n$ $(j = 1,\dots,n)$. Second, we introduce vectors $r_j^{\odot *} \in X^{\odot *}$ by putting

$$r_j^{\odot *} \equiv (e_j, 0) \qquad (j = 1,\dots,n)$$

(remember that we use the representation in Table 2.1). Using this notation, we can define a C^k-smooth mapping $R : X \times \mathbb{R}^m \to X^{\odot *}$, called **the nonlinearity**, by

$$R(\phi,\alpha) \equiv \sum_{j=1}^{n} \left(g(\phi,\alpha) + \int_0^h (d\zeta(\vartheta,\alpha) - d\zeta(\vartheta,\alpha_0))\,\phi(-\vartheta) \right)_j r_j^{\odot *}, \tag{3.5}$$

where the subscript j refers to the j-th component of the vector between the large parenthesis. Because $r_j^{\odot *} = (e_j, 0)$, the (finite-dimensional) range of R is contained in the linear span of the $\mathbb{R}^n$-compontent of $X^{\odot *}$. Also note that our assumptions (3.4) entail that

$$R(0,\alpha_0) = 0, \qquad D_1 R(0,\alpha_0) = 0.$$

The kernel $\zeta(\cdot,\alpha_0) \in \mathrm{NBV}([0,h],\mathbb{R}^n)$ defines a linear DDE (of the form (3.3) with $g = 0$). Let $\{T(t)\}_{t\geqslant 0}$ be the corresponding semigroup of solution operators. We consider the parameter-dependent nonlinear abstract integral equation

$$u(t) = T(t)\phi + \int_0^t T^{\odot *}(t-\tau) R(u(\tau),\alpha)\,d\tau, \tag{3.6}$$

where $\phi \in X$ is given and the integral must be interpreted as a weak*-integral (with values in X). Solutions of (3.6) are continuous functions $u : [0, t_+) \to X$, where we can take $t_+ = \infty$. They have a one-to-one correspondence to solutions of (3.3) in the following way: if $x(\cdot,\alpha,\phi) : [-h,\infty) \to \mathbb{R}^n$ solves (3.3) with initial condition $x_0 = \phi$, then the function defined by

$$u(t,\alpha,\phi) = x_t(\cdot,\alpha,\phi) \qquad (t \geqslant 0)$$

uniquely solves (3.6). Conversely, if $u(\cdot,\alpha,\phi)$ is a solution of (3.6) then the function $x(\cdot,\alpha,\phi) : [-h,\infty) \to \mathbb{R}^n$ defined by

$$x_0 \equiv \phi, \quad x(t,\alpha,\phi) \equiv u(t,\alpha,\phi)(0) \quad \text{for all } t \geqslant 0$$

uniquely solves (3.3) with initial condition ϕ.

3.2 The spectrum of the generator

As we have seen, there is a strongly continuous semigroup $\{T(t)\}_{t\geqslant 0}$ yielding solutions to the linear part of (3.2), with generator A. As promised, the spectrum $\sigma(A)$ of this generator determines the phase portrait of the DDE. In this section, we present a method to extract the spectrum without having direct access to A.

First, we recap some terminology about spectra of linear operators on Banach spaces (due to Chapter IV.2 in [3]).

Definition 3.1. Let $L : \mathcal{D}(L) \to X$ be a linear operator with domain $\mathcal{D}(L)$ in a complex Banach space X. A complex number λ belongs to the **resolvent set** $\rho(L)$ of L if and only if the **resolvent operator** $(zI - L)^{-1}$ exists and is bounded, i.e.

(1) $\lambda I - L$ is injective,

(2) $\mathcal{R}(\lambda I - L) = X$,

(3) $(\lambda I - L)^{-1}$ is bounded.

The **spectrum** $\sigma(L)$ is by definition the complement $\mathbb{C} \setminus \rho(L)$.

The point spectrum $\sigma_p(L)$ is the set of those $\lambda \in \mathbb{C}$ for which $\lambda I - L$ is not one-to-one, i.e. $L\phi = \lambda\phi$ for some $\phi \neq 0$. One then calls λ an **eigenvalue** and ϕ an **eigenvector** corresponding to λ.

The null space $\mathcal{N}(\lambda I - L)$ is called the **eigenspace** corresponding to λ and its dimension the **geometric multiplicity** of λ. The **generalized eigenspace** $\mathcal{M}_\lambda(L)$ is the smallest closed linear subspace that contains all $\mathcal{N}\left((\lambda I - L)^j\right)$ for $j = 1, 2, \ldots$ and its dimension $M(L;\lambda)$ is called the **algebraic multiplicity** of λ. If, in addition, λ is an isolated point in $\sigma(L)$ and $M(L;\lambda)$ is finite, then λ is called an **eigenvalue of finite type**. When $M(L;\lambda) = 1$ we say that λ is a **simple eigenvalue**.

Note that closed operators automatically fulfil the third condition in Definition 3.1 (by the Closed Graph Theorem). The spectrum of compact operators consists of eigenvalues of finite type only.

Working with A directly is not very convenient, but fortunately, there is a much simpler object that can give us all necessary information about $\sigma(A)$.

Definition 3.2. The matrix-valued function $\Delta : \mathbb{C} \to \mathbb{C}^{n\times n}$ defined by

$$\Delta(z) = zI - \int_0^h e^{-z\vartheta}\, d\zeta(\vartheta) \tag{3.7}$$

is called the **characteristic matrix** of equation 3.3.

In concrete cases (the ones we will be applying DDE-BIFTOOL to), the integral usually becomes a finite sum, as Example 3.3 shows us.

Example 3.3. Suppose we have the DDE

$$\dot{x}(t) = x(t) - x(t - \tfrac{1}{2}).$$

We can write this as

$$\dot{x}(t) = \int_0^h d\zeta(\vartheta)\, x(t - \vartheta)$$

if we treat $d\zeta$ as a combination of Dirac measures:

$$d\zeta(\vartheta) = \delta(\vartheta) - \delta(\vartheta - \tfrac{1}{2}).$$

This means that (with $I = I_1 = 1$)

$$\Delta(z) = z - \int_0^h e^{-z\vartheta}\, d\zeta(\vartheta) = z - \int_0^h e^{-z\vartheta}\delta(\vartheta) + \int_0^h e^{-z\vartheta}\delta(\vartheta - \tfrac{1}{2}) = z - 1 + e^{-\frac{1}{2}z}.$$

In the next theorem, we summarize the main results about the spectrum of A and its relation to Δ.

Theorem 3.4 ([10], Theorem 2.3)**.** Let $(A, \mathcal{D}(A))$ be the generator of the semigroup $\{T(t)\}_{t\geqslant 0}$ corresponding to the linear part of (3.2), with $\alpha = \alpha_0$.

(1) $\sigma(A) = \sigma(A^*) = \sigma\left(A^{\odot}\right) = \sigma\left(A^{\odot *}\right)$, and these spectra consist solely of eigenvalues of finite type.

(2) The characteristic matrix $\Delta(z)$ is holomorphic, and $\lambda \in \sigma(A)$ if and only if $\det \Delta(\lambda) = 0$. In this case, the order of λ as a root of $\det \Delta$ equals the algebraic multiplicity of λ as an eigenvalue and the dimension of the nullspace $\mathcal{N}(\Delta(\lambda))$ is equal to the geometric multiplicity of λ as an eigenvalue.

(3) The generalized eigenspaces corresponding to λ are given by the nullspaces

$$\mathcal{N}\left((\lambda I - A)^{k_\lambda}\right) = \mathcal{N}\left((\lambda I - A^{\odot *})^{k_\lambda}\right),$$
$$\mathcal{N}\left((\lambda I - A^*)^{k_\lambda}\right) = \mathcal{N}\left((\lambda I - A^{\odot})^{k_\lambda}\right),$$

where k_λ is the order of λ as a pole of the map $z \mapsto \Delta(z)^{-1}$.

The transcendental equation $\det \Delta(z) = 0$ is called the **characteristic equation**. The eigenvectors of A can be obtained from the characteristic matrix as well. We will only need eigenvectors for the special cases when $\lambda \in \sigma(A)$ is simple or when λ is a double eigenvalue. First, we treat the simple case.

Lemma 3.5 ([3], Chapter IV, Theorems 5.5 and 5.9 and Corollary 5.12.)**.** Let λ be a simple eigenvalue of A. If the non-zero column vector q is a right null vector of $\Delta(\lambda)$ (i.e. $\Delta(\lambda)q = 0$), then

$$\phi = \vartheta \mapsto e^{\lambda\vartheta} q$$

is an eigenvector of A corresponding to λ. Furthermore, if the non-zero row vector p is a left null vector of $\Delta(\lambda)$ (i.e. $p\Delta(\lambda) = 0$, then $\phi^{\odot} \in X^*$ given by the NBV-representation

$$\phi^{\odot}(\vartheta) = p\left(I + \int_0^{\vartheta}\int_{\sigma}^{h} e^{\lambda(\sigma - s)} d\zeta(s)\, d\sigma\right)$$

is an eigenvector of A^* corresponding to λ. Finally,

$$\left\langle \phi^{\odot}, \phi \right\rangle = p\Delta'(\lambda)q \neq 0,$$

where $\Delta'(\lambda)$ denotes the derivative of $z \mapsto \Delta(z)$ at $z = \lambda$.

If λ is a double eigenvalue, we will need the notion of a Jordan chain, which is a generalization of the concept of an eigenvector.

Definition 3.6 ([10], Definition 2.6)**.** A sequence of column vectors $q_0, q_1, \ldots, q_{k-1}$ in $\mathbb{R}^n$ is called a **right Jordan chain** for $\Delta(z)$ at $z = \lambda$ if $q_0 \neq 0$ and

$$\Delta(z)\left(q_0 + (z-\lambda)q_1 + \cdots + (z-\lambda)^{k-1}q_{k-1}\right) = \mathcal{O}\left((z-\lambda)^k\right) \text{ as } z \to \lambda.$$

The number k is called the **rank** of the chain. Similarly, a sequence of row vector $p_0, \ldots, p_{k-1}$ in $\mathbb{R}^n$ is called a **left Jordan chain** of rank k for $\Delta(z)$ at $z = \lambda$ if $p_0 \neq 0$ and

$$\left(p_{k-1} + (z-\lambda)p_{k-2} + \cdots + (z-\lambda)^{k-1}p_0\right)\Delta(z) = \mathcal{O}\left((z-\lambda)^k\right) \text{ as } z \to \lambda.$$

In the special case that λ is a double eigenvalue of A of geometric multiplicity one (i.e. the dimension of $\mathcal{N}(\lambda I - A)$ is 1), there exists an eigenvector $\phi_0 \in \mathcal{D}(A)$ and a generalized eigenvector $\phi_1 \in \mathcal{D}(A)$ such that

$$A\phi_0 = \lambda\phi_0, \quad A\phi_1 = \lambda\phi_1 + \phi_0.$$

There also exists an eigenvector $\phi_1^{\odot} \in \mathcal{D}(A^*)$ and a generalized eigenvector $\phi_0^{\odot} \in \mathcal{D}(A^*)$ such that

$$A^*\phi_1^{\odot} = \lambda\phi_1^{\odot}, \quad A^*\phi_0^{\odot} = \lambda\phi_0^{\odot} + \phi_1^{\odot}.$$

Using the characteristic matrix, we can find explicit eigenfunctions.

Lemma 3.7 ([10], Lemma 2.7)**.** Let λ be an eigenvalue of A with geometric multiplicity one and algebraic multiplicity two. Let

$$\{q_0, q_1\} \in \mathbb{R}^n, \quad \{p_1, p_0\} \in \mathbb{R}^n$$

be right and left Jordan chains of $\Delta(z)$ of rank two at $z = \lambda$. Then the column vector valued functions

$$\phi_0 = (\vartheta \mapsto e^{\lambda\vartheta}q_0), \quad \phi_1 = \left(\vartheta \mapsto e^{\lambda\vartheta}(\vartheta q_0 + q_1)\right)$$

are an eigenvector and a generalized eigenvector for A corresponding to λ and the row vector valued functions

$$\phi_1^{\odot} = (0, g_1) \text{ where } g_1(\vartheta) = p_1\left(I + \int_0^{\vartheta}\int_{\sigma}^{h} e^{\lambda(\sigma - s)}d\zeta(s)\,d\sigma\right),$$

$$\phi_0^\odot = (0, g_0) \text{ where } g_0(\vartheta) = p_0\left(I + \int_0^\vartheta \int_\sigma^h e^{\lambda(\sigma-s)} d\zeta(s)\, d\sigma\right) + p_1\left(I + \int_0^\vartheta \int_\sigma^h e^{\lambda(\sigma-s)}(\sigma - s) d\zeta(s)\, d\sigma\right)$$

are an eigenvector and a generalized eigenvector for A^* corresponding to λ. Moreover, the following identities hold:

$$\begin{aligned}
\left\langle \phi_0^\odot, \phi_0 \right\rangle &= p_0\Delta'(\lambda)q_0 + \frac{1}{2!}p_1\Delta''(\lambda)q_0, \\
\left\langle \phi_1^\odot, \phi_1 \right\rangle &= p_1\Delta'(\lambda)q_1 + \frac{1}{2!}p_1\Delta''(\lambda)q_0, \\
\left\langle \phi_1^\odot, \phi_0 \right\rangle &= p_1\Delta'(\lambda)q_0, \\
\left\langle \phi_0^\odot, \phi_1 \right\rangle &= p_0\Delta'(\lambda)q_1 + \frac{1}{2!}p_0\Delta''(\lambda)q_0 + \frac{1}{2!}p_1\Delta''(\lambda)q_1 + \frac{1}{3!}p_1\Delta'''(\lambda)q_0.
\end{aligned}$$

3.3 The center manifold

We now know where to look to detect bifurcations (i.e. the spectrum of the generator via the characteristic matrix). We now turn our attention to the center manifold on which the bifurcations occur.

In the case of DDEs, the proofs of the properties of the center manifold are harder than in the case of ODEs, and some details are different, but the general idea remains true. As in the ODE case, the center manifold can be described as a graph of a function on a certain subspace of X, and we start by describing this subspace.

Definition 3.8 ([10], Section 2.3)**.** Let $(A, \mathcal{D}(A))$ be the generator of the semigroup $\{T(t)\}_{t\geqslant 0}$ corresponding to the linear part of (3.2), with $\alpha = \alpha_0$. Then the **center subspace** X_0 is defined as the direct sum

$$X_0 = \bigoplus \{\mathcal{M}_\lambda : \lambda \in \sigma(A) \cap i\mathbb{R}\}.$$

Here $\mathcal{M}_\lambda$ is the generalized eigenspace corresponding to λ and $i\mathbb{R}$ is the imaginary axis.

So in words, X_0 consists of the generalized eigenspaces of all eigenvalues of A on the imaginary axis. We can see that X_0 is finite-dimensional by the first statement of Theorem 3.4.

Theorem 3.9 ([3], Theorem IX.5.3, Corollary IX.7.10, Section IX.8)**.** Let $P_0 \in \mathcal{L}(X)$ be the spectral projection of X onto X_0 and denote its extension to $X^{\odot *}$ with range X_0 by $P_0^{\odot *} \in \mathcal{L}(X^{\odot *}, X)$.

For $\delta > 0$ sufficiently small, there exists a C^k-smooth injection $\mathcal{C}_\delta : X_0 \to X$ such that its image $\mathcal{W}_\delta^c = \mathcal{C}_\delta(X_0)$, called the (local) **center manifold**, has the following properties:

(1) $\mathcal{W}^c_\delta$ is conditionally locally forward-invariant, in the following sense. If $\phi \in X_0$ and $\sup\{\|u(t, C_\delta(\phi)) : t \in [0,T]\|\} \leqslant \delta$, then $u(t, C_\delta(\phi)) = C_\delta(P_0 u(t, C_\delta(\phi)))$ for all $t \in [0,T]$.

(2) $\mathcal{W}^c_\delta$ contains all solutions of (3.6) (with $\alpha = \alpha_0$ that are defined for all time and satisfy $\sup\{\|u(t,\psi) : t \in \mathbb{R}\|\} \leqslant \delta$.

(3) $\mathcal{W}^c_\delta$ contains the origin, since $C_\delta(0) = 0$ and it is tangent to X_0 there, i.e. $DC_\delta(0)\phi = \phi$ for all $\phi \in X_0$.

(4) If $\psi \in \mathcal{W}^c_\delta$ and $u_\delta(\cdot,\psi)$ exists for all time, then $y(t) \equiv P_0 u(t,\psi) \in X_0$ satisfies the ordinary differential equation

$$\dot{y}_\delta(t) = A y_\delta(t) + P_0^{\odot *} R(y_\delta(t)) \qquad (t \in \mathbb{R}).$$

If we study only the local dynamics, i.e. inside a small ball $B_\delta(0)$ centered around the origin, we may drop the subscript and write $\mathcal{W}^c$ and $\mathcal{C}$ instead of $\mathcal{W}^c_\delta$ and $\mathcal{C}_\delta$. If $\sigma(A)$ does not contain points in the open right half-plane (with real part > 0), then $\mathcal{W}^c$ is conditionally locally exponentially stable, implying that if a solution that lies in $B_\delta(0)$ for all time is locally exponentially stable within $\mathcal{W}^c$, then it is locally exponentially stable in X.

We saw in statement (4) of Theorem 3.9 that the projection of a solution onto X_0 satisfies a certain ODE on X_0. Near the origin, a solution also satisfies an ODE on the center manifold $\mathcal{C}$.

Proposition 3.10 ([10], Proposition 2.11)**.** Let $\psi \in \mathcal{W}^c$ and suppose that the solution $t \mapsto u(t,\psi)$ of (3.6) with $\alpha = \alpha_0$ exists as a map from $\mathbb{R}$ to X and lies in $B_\delta(0)$ for all $t \in \mathbb{R}$. Then $u(t,\psi) \in \mathcal{W}^c$ for all $t \in \mathbb{R}$ and $u(t,\psi)$ is differentiable with respect to t and satisfies

$$\frac{du(t,\psi)}{dt} = A^{\odot *} u(t,\psi) + R(u(t,\psi)) \qquad \text{for all } t \in \mathbb{R}. \tag{3.8}$$

We now have covered enough theory to start describing normal forms of DDEs.

Chapter 4

Normal forms of DDEs

Just as for ODEs, we can find normal forms for bifurcations in DDEs and compute their critical coefficients. We first sketch the general procedure and then present a concrete derivation for the Hopf case. Finally, we list the formulas for the critical normal form coefficients for the codimension 2 bifurcations.

4.1 The generic critical normal form

There exist several ways to arrive at the normal form coefficients. We follow a method that was first used by Kuznetsov ([11], Chapter 8) for ODEs and then applied to DDEs in [10]. Do note that this method yields only the **critical normal form coefficients**, i.e. the value of the normal form coefficient at criticality, when the bifurcation occurs. We first explain, following [10], what is actually meant by a normal form coefficient in a DDE context.

We continue in the setting of Chapter 3 and consider the general DDE

$$\dot{x}(t) = f(x_t, \alpha) \tag{4.1}$$

and its rewritten form

$$\dot{x}(t) = \int_0^h d\zeta(\vartheta, \alpha)\, x_t(-\vartheta) + g_f(x_t, \alpha).$$

Suppose that at the **critical parameter value** $\alpha = \alpha_0 = 0$ the zero function is a stationary solution of (4.1), i.e.

$$f(0,0) = 0.$$

Let $(A, \mathcal{D}(A))$ be the generator of the semigroup $\{T(t)\}_{t\geqslant 0}$ solving the linear DDE associated with the linearized equation

$$\dot{x}(t) = D_1 f(0, \alpha_0) x_t$$

and suppose that one of the bifurcation conditions in Tables 4.1 or 4.2 is satisfied and A has no other eigenvalues on the imaginary axis.

If the above conditions are met, there exists a non-trivial center subspace X_0 of finite dimension n_c, spanned by some basis Φ consisting of eigenvectors and (in the case of Bogdanov-Takens) generalized eigenvectors corresponding to the eigenvalues of A that lie on the imaginary axis. (These eigenvectors are actually eigenfunctions, of course.) Tangent to X_0 there

Bifurcation	Condition
Hopf	$\lambda_{1,2} = \pm i\omega_0,\ \omega_0 > 0$
Fold	$\lambda_1 = 0$

TABLE 4.1: *Bifurcation conditions for codimension-one bifurcations.*

Bifurcation	Condition
Cusp	$\lambda_1 = 0,\ b = 0$
Bogdanov-Takens	$\lambda_1 = \lambda_2 = 0$
Bautin (generalized Hopf)	$\lambda_{1,2} = \pm i\omega_0,\ \omega_0 > 0,\ l_1(0) = 0$
Fold-Hopf	$\lambda_1 = 0,\ \lambda_{2,3} = \pm i\omega_0,\ \omega_0 > 0$
Double Hopf	$\lambda_{1,4} = \pm i\omega_1,\ \lambda_{2,3} = \pm i\omega_2,\ \omega_{1,2} > 0$

TABLE 4.2: *Bifurcation conditions for codimension-two bifurcations.*

exists a local center manifold $\mathcal{W}^c_\delta$. We now look at equation (3.6) for $\alpha = \alpha_0$, replicated here for convenience as (4.2).

$$u(t) = T(t)\phi + \int_0^t T^{\odot *}(t-\tau)R(u(\tau), \alpha_0)\, d\tau. \tag{4.2}$$

We consider solutions $u : [0, t_+) \to X$ that are defined and lie on $\mathcal{W}^c_\delta$ for all (positive and negative) time. Let $y(t)$ be the projection of $u(t)$ onto X_0, i.e. y is a function $[0, t_+) \to X_0$. Then $y(t)$ can be expressed uniquely relatively to the basis of eigenfunctions Φ. Denote $y(t)$'s coordinates with respect to Φ by the coordinate vector $z(t)$. Then $z(t)$ satisfies an ODE admitting an expansion of the form

$$\dot{z}(t) = \sum_{|\nu|=1}^{N} \frac{1}{\nu!} g_\nu z^\nu(t) + \mathcal{O}\left(\|z(t)\|^{N+1}\right) \quad \text{for all } t \in \mathbb{R}. \tag{4.3}$$

(See Figure 4.1 for a visualization of the situation.)

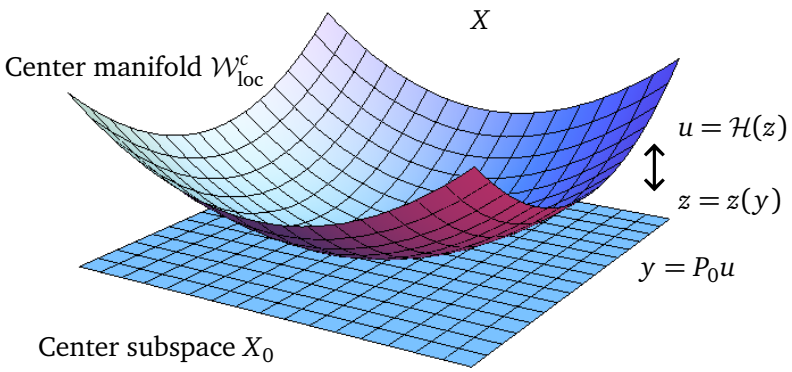


FIGURE 4.1: *The vector z is the coordinate vector of the center manifold with respect to the basis on X_0.*

This is then our generic critical normal form, with unknown critical normal form coefficients g_ν. The symbol ν denotes a multi-index[1] of length n_c and the series is supposed to be truncated after some sufficiently high order N.

[1] A multi-index of length n is a vector $\nu \in \mathbb{N}_0^n$. When summing over a multi-index at a step $|\nu| := \nu_1 + \cdots + \nu_n = k$, one adds one term for every multi-index satisfying $|\nu| = k$, following the convention $\nu! := \nu_1 \cdots \nu_n$ and $z^\nu = z_1^{\nu_1} \cdots z_n^{\nu_n}$, where $z \in \mathbb{R}^n$. So, assuming $n = 2$, the first few terms of (4.3) read: $\dot{z}(t) = g_{01}z_2(t) + g_{10}z_1(t) + \frac{1}{2}g_{02}z_2(t)^2 + g_{11}z_1(t)z_2(t) + g_{20}z_1(t)^2 + \cdots$

4.2 A general method to derive specific normal forms

We now present a way to actually determine the critical normal forms and their coefficients for a given bifurcation, based on [10] and [14].

First, we return to the linearity give in (3.5). At the critical parameter value $\alpha = \alpha_0$ $(= 0)$, the linear part vanishes, and we simply have

$$R(\phi) = \sum_{j=1}^{n} g_j(\phi) r_j^{\odot *} \quad \text{with} \quad r_j^{\odot *} = (e_j, 0) \in X^{\odot *}.$$

Notice that we have suppressed the dependence on α altogether. Now suppose f is sufficiently smooth. Then we can expand the nonlinearity as

$$R(\phi) = \frac{1}{2!} B(\phi, \phi) + \frac{1}{3!} C(\phi, \phi, \phi) + \mathcal{O}\left(\|\phi\|^4\right), \tag{4.4}$$

where

$$\begin{aligned} &B \in L_2\left(X, X^{\odot *}\right), \qquad B(\phi_1, \phi_2) := D^2 R(0)(\phi_1, \phi_2), \\ &C \in L_3\left(X, X^{\odot *}\right), \qquad C(\phi_1, \phi_2, \phi_3) := D^3 R(0)(\phi_1, \phi_2, \phi_3), \end{aligned}$$

i.e. B and C are symmetric bounded multilinear forms from X to $X^{\odot *}$.

These multilinear forms are given by derivatives of f. For two arbitrary elements $\xi_1, \xi_2 \in X$, we have

$$\begin{aligned} B(\xi_1, \xi_2) &= \sum_{j=1}^{n} \left[D^2 g(0)(\xi_1, \xi_2) \right]_j r_j^{\odot *}, \\ &= \sum_{j=1}^{n} \left[D^2 f(0)(\xi_1, \xi_2) \right]_j r_j^{\odot *} \\ &= \left[D^2 f(0)(\xi_1, \xi_2) \right] r^{\odot *}. \end{aligned} \tag{4.5}$$

This is because we still have $g(\phi) = f(\phi) - D_1 f(0)\phi$. The last line should be interpreted as an 'inner-like' product of $D^2 f(0)(\xi_1, \xi_2) \in \mathbb{R}^n$ with $r^{\odot *} \equiv (r_1^{\odot *}, \ldots, r_n^{\odot *})$. Analogously, $C(\xi_1, \xi_2, \xi_3) = D^3 f(0)(\xi_1, \xi_2, \xi_3) r^{\odot *}$ and so forth. So in general, we have

$$R(\phi) = \sum_{j>1}^{N} \frac{1}{j!} D^j f(0)(\overbrace{\phi, \ldots, \phi}^{j \text{ times}}) r^{\odot *} + \mathcal{O}\left(\|\phi\|^{N+1}\right). \tag{4.6}$$

Recall from Theorem 3.9 that the local center manifold $\mathcal{W}_\delta^c$ is given as the image of a C^k-smooth injection $\mathcal{C}_\delta : U \subset X_0 \to X$, where U is some open ball around the origin of X_0 and $\mathcal{W}_\delta^c$ is tangent to X_0 there. This mapping can be expanded as well.

Definition 4.1. Let $\mathcal{W}_\delta^c = \mathcal{C}_\delta(U)$ be the local center manifold with dimension n_c. Let ξ be a point in X_0 with coordinate $z = \left\langle \phi^{\odot}, \xi \right\rangle \in \mathbb{R}^{n_c}$. Now the coordinate mapping $\xi \mapsto z(\xi)$ is a C^k-smooth injection onto V, a neighbourhood of the origin of $\mathbb{R}^{n_c}$. We define the mapping $\mathcal{H} : V \to X$ by

$$\mathcal{H}(z) := \mathcal{C}_\delta(\xi(z)).$$

This mapping can now be expanded as

$$\mathcal{H}(z) = \sum_{1 \leqslant |\nu| \leqslant 3} \frac{1}{\nu!} z^\nu + \mathcal{O}\left(\|z\|^4\right). \tag{4.7}$$

We recall equation (3.8) and rewrite it here succinctly as

$$\frac{du(t)}{dt} = A^{\odot *} u(t) + R(u(t)) \quad \text{for all } t \in \mathbb{R}. \tag{4.8}$$

As before, let $y(t)$ be the projection of a small solution $u(t)$ onto X_0 and $z(t)$ its coordinate vector with respect to a basis of eigenfunctions (of A) Φ. Then, because of the invariance of the center manifold,

$$u(t) = \mathcal{H}(z(t)) \quad \text{for all } t \in \mathbb{R}.$$

Differentiating both sides of this relation with respect to time and using (4.8), we get

$$\begin{aligned} \frac{du(t)}{dt} &= D\mathcal{H}(z)\dot{z}, \\ A^{\odot *}\mathcal{H}(z) + R(\mathcal{H}(z)) &= D\mathcal{H}(z)\dot{z}. \end{aligned} \tag{4.9}$$

Equation (4.9) is called the **homological equation**. The idea is now to substitute (4.3), (4.6) and (4.7) into the homological equation and solve for the unknown coefficients g_ν and h_ν by equating like powers of z.

In order to solve the homological equation, it is necessary to solve linear operator equations of the form

$$\begin{aligned} &\left(\lambda - A^{\odot *}\right)\phi^{\odot *} = \psi^{\odot *} \quad \text{or} \\ &\left(\lambda - A^{\odot *}\right)(v_0, v) = (w_0, w), \end{aligned} \tag{4.10}$$

where $\lambda \in \mathbb{C}$ and $\psi^{\odot *} \in X^{\odot *}$ is given. Note that we can use the representations in Table 2.1 to arrive at the second form of the equation. This type of equation can be solved using a version of the Fredholm alternative.

Theorem 4.2 (Fredholm alternative; [10] Lemma 3.2; [14] Lemma 33)**.** Let λ be arbitrary. Then (4.10) has a solution $\phi^{\odot *} = (v_0, v) \in \mathcal{D}\left(A^{\odot *}\right)$ if and only if $\psi^{\odot *} = (w_0, w)$ annihilates $N(\lambda - A^*)$, i.e.

$$\left\langle \phi^\odot, \psi^{\odot *} \right\rangle = 0 \quad \text{for all } \phi^\odot \in N(\lambda - A^*).$$

If λ is not an eigenvalue, there are explicit formulas for the solution.

Theorem 4.3 ([10] Lemma 3.3)**.** Suppose λ is not an eigenvalue. Then the unique solution $(v_0, v) \in \mathcal{D}\left(A^{\odot *}\right)$ of (4.10) is given by

$$v(\vartheta) = e^{\lambda\vartheta} v_0 + \int_\vartheta^0 e^{\lambda(\vartheta - \sigma)} w(\sigma)\, d\sigma \quad (\vartheta \in [-h, 0]),$$

$$v_0 = \Delta(\lambda)^{-1}\left(w_0 + \int_0^h d\zeta(\tau)\int_0^\tau e^{-\lambda\sigma}w(\sigma-\tau)\,d\sigma\right),$$

where $\Delta(\lambda)$ is the characteristic matrix.

Corollary 4.4 ([10] Corollary 3.4)**.** The following two special cases are useful for our calculations.
Let $(w_0, w) = (w_0, 0)$. Then the solution (v_0, v) of (4.10) is given by

$$(v_0, v) = \begin{pmatrix} \Delta(\lambda)^{-1}w_0 \\ \vartheta \mapsto e^{\lambda\vartheta}\Delta(\lambda)^{-1}w_0 \end{pmatrix}.$$

Let $(w_0, w) = (w_0, \vartheta \mapsto e^{\lambda\vartheta}\Delta(\lambda)^{-1}\zeta)$ for some fixed vector ζ in $\mathbb{R}^n$. Then

$$(v_0, v) = \begin{pmatrix} \Delta(\lambda)^{-1}\left[\Delta'(\lambda) - I\right]\Delta(\lambda)^{-1}\zeta \\ \vartheta \mapsto \Delta(\lambda)^{-1}\left[\Delta'(\lambda) - I - \vartheta\Delta(\lambda)\right]w(\vartheta) \end{pmatrix}.$$

If λ does happen to be an eigenvalue, (4.10) does not have a unique solution, if one even exists. However, it's possible to select a convenient choice among all available solutions, as the next theorem shows.

Theorem 4.5 ([10] Lemma 3.5)**.** Let $L : \mathcal{D}(L) \subset E \to E$ be a closed, densely defined operator on a Banach space E. Suppose that zero is a simple eigenvalue of L and L^* with corresponding eigenvectors ψ and ψ^*. Let P be the spectral projection operator of E onto the zero-eigenspace. Assume that for given $y^* \in E^*$ there exists a particular solution x_0^* in $\mathcal{D}(L^*)$ of the equation

$$L^*x^* = y^*. \tag{4.11}$$

Then the augmented system

$$\begin{cases} L^*x^* + s\psi^* & = y^* \\ \langle x^*, \psi\rangle & = 0 \end{cases} \tag{4.12}$$

has a unique solution $x^* = (I - P^*)\,x_0^*$ and $s = 0$, and x^* is the unique solution of (4.11) that annihilates ψ.

If λ is a simple eigenvalue of A and (4.10) has a solution, then we can apply Theorem 4.5 to the operator $L = \lambda - A^\odot$ on $X^\odot$ with domain $\mathcal{D}\left(A^\odot\right)$ to obtain the unique solution of (4.10) that vanishes on the eigenspace corresponding to λ.

The system (4.12) visually looks like a matrix equation of operators:

$$\begin{pmatrix} L^* & \psi^* \\ \psi & 0 \end{pmatrix}\begin{pmatrix} x^* \\ s \end{pmatrix} = \begin{pmatrix} y^* \\ 0 \end{pmatrix}.$$

This type of system, containing auxiliary equations and unknowns, is called a **bordered system**.

Definition 4.6. The unique solution x^* mentioned in Theorem 4.5 is denoted by

$$x^* = (L^*)^{\mathrm{INV}} y^*.$$

We can given an explicit expression for $\left(\lambda I - A^{\odot *}\right)^{\mathrm{INV}}$ when λ is a simple eigenvalue, just as we did in Theorem 4.3 for the non-singular case.

Theorem 4.7 ([10], Proposition 3.6)**.** Suppose λ is a simple eigenvalue of A and assume that (4.10) is consistent for a given $(w_0, w) \in X^{\odot *}$ with bordered inverse $(v_0, v) = \left(\lambda I - A^{\odot *}\right)^{\mathrm{INV}} (w_0, w)$ in $X^{\odot *}$. Let $q, p \in \mathbb{R}^n$, $\phi \in X$, $\phi^{\odot} \in X^{\odot}$ be as in Lemma 3.5, normalized to $\left\langle \phi^{\odot}, \phi \right\rangle = 1$. Then

$$v(\vartheta) = e^{\lambda \vartheta} v_0 + \int_{\vartheta}^{0} e^{\lambda(\vartheta - \sigma)} w(\sigma)\, d\sigma \quad (\vartheta \in [-h, 0]),$$

with

$$v_0 = \xi + \gamma q, \quad \xi := \Delta(\lambda)^{\mathrm{INV}} \left(w_0 + \int_0^h d\zeta(\tau) \int_0^{\tau} e^{-\lambda \sigma} w(\sigma - \tau)\, d\sigma \right).$$

The constant γ is given by

$$\gamma = -p\Delta'(\lambda)\xi - p \int_0^h \int_{\tau}^h e^{-\lambda s}\, d\zeta(s) \int_{-\tau}^0 e^{-\lambda \sigma} w(\sigma)\, d\sigma\, d\tau.$$

A special case that can be used is the following.

Corollary 4.8 ([10], Corollary 3.7)**.** Suppose that in (4.10) we have $(w_0, w) = (\eta, 0) + \kappa(q, \phi)$, where $\eta \in \mathbb{R}^n$ is an arbitrary vector and κ is a scalar. Then

$$v_0 = \xi + \gamma q, \quad v(\vartheta) = e^{\lambda \vartheta}(\xi + \gamma q - \kappa \vartheta q), \quad (\vartheta \in [-h, 0])$$

with

$$\xi = \Delta(\lambda)^{\mathrm{INV}} \left(\eta + \kappa \Delta'(\lambda) q\right) \quad \text{and} \quad \gamma = -p\Delta'(\lambda)\xi + \frac{1}{2}\kappa p \Delta''(\lambda) q.$$

In this case, we employ the notation $v = B_{\lambda}^{\mathrm{INV}}(\zeta, \kappa)$.

This concludes our description of the general method to find the normal form coefficients.

4.3 Derivation of the Hopf critical normal form

For the different types bifurcations, (4.3) can be reduced to different specific normal forms. We will derive the Hopf critical normal form and its coefficients here in detail; the other normal forms will be simply listed in the next section. For their derivation, we refer the reader to [10] or [14].

A Hopf bifurcation is a codimension-one bifurcation where $\sigma(A)$ contains a simple purely imaginary pair $\lambda_{1,2} = \pm i\omega_0$ with $\omega_0 > 0$ and no other purely imaginary eigenvalues.

Let ϕ and $\phi^{\odot}$ be complex eigenvectors of A and A^* corresponding to $\lambda_1 = +i\omega_0$, satisfying $\langle \phi, \phi^{\odot} \rangle = 1$ and let p and q be as in Lemma 3.5. The generic normal form equation (4.3) can be reduced to the **Poincaré normal form**

$$\dot{z} = i\omega_0 z + c_1(0) z|z|^2 + \mathcal{O}\left(\|z\|^4\right), \tag{4.13}$$

where z is complex and the critical normal form coefficient $c_1(0)$ is unknown. Any point y in the real two-dimensional center subspace X_0 corresponding to $\lambda_{1,2}$ may be uniquely expressed with respect to the set $\left\{\phi, \overline{\phi}\right\}$ by means of the smooth complex coordinate mapping

$$y \mapsto (z, \bar{z}), \quad z := \langle \phi^{\odot}, y \rangle.$$

The homological equation (4.9) becomes

$$A^{\odot *}\mathcal{H}(z,\bar{z}) + R(\mathcal{H}(z,\bar{z})) = D_z\mathcal{H}(z,\bar{z})\dot{z} + D_{\bar{z}}\mathcal{H}(z,\bar{z})\dot{\bar{z}}$$

with center manifold expansion

$$\mathcal{H}(z,\bar{z}) = z\phi + \bar{z}\overline{\phi} + \sum_{2 \leqslant j+k \leqslant 3} \frac{1}{j!k!} h_{jk} z^j \bar{z}^k + \mathcal{O}\left(\|z\|^4\right). \tag{4.14}$$

Note that since the image of $\mathcal{H}$ lies in the real space X, it follows that its coefficients satisfy $h_{kj} = \overline{h}_{jk}$. The derivatives $\dot{z}$ and $\dot{\bar{z}}$ are given by (4.13) and its complex conjugate.

We will now expand the homological equation in some detail. After combining (4.13) and (4.14), the right hand side of the homological equation becomes

$$\begin{aligned}
\text{RHS} &= (\phi + h_{11}\bar{z} + h_{20}z)\dot{z} + \left(\overline{\phi} + h_{02}\bar{z} + h_{11}z\right)\dot{\bar{z}} \\
&= i\omega_0 z\phi + i\omega_0 h_{20} z^2 + c_1(0) z^2\bar{z}\phi + -i\omega_0 \bar{z}\overline{\phi} - i\omega_0 h_{02}\bar{z}^2 + \overline{c_1(0)}\bar{z}^2 z\overline{\phi} + \mathcal{O}\left(\|z\|^4\right).
\end{aligned}$$

Likewise, we can expand the left hand side, using (4.14) and (4.4).

$$\begin{aligned}
\text{LHS} = {} & A^{\odot *} z\phi + A^{\odot *}\bar{z}\overline{\phi} + \frac{1}{2}A^{\odot *}h_{02}\bar{z}^2 + A^{\odot *}h_{11}z\bar{z} + \frac{1}{2}A^{\odot *}h_{20}z^2 \\
& + \frac{1}{2}z^2 B(\phi,\phi) + z\bar{z}B(\phi,\overline{\phi}) + \frac{1}{2}\bar{z}^2 B(\overline{\phi},\overline{\phi}) + \frac{1}{2}z\bar{z}^2 B(\phi,h_{02}) + z^2\bar{z}B(\phi,h_{11}) \\
& + \frac{1}{2}z^3 B(\phi,h_{20}) + \frac{1}{2}\bar{z}^3 B(\overline{\phi},h_{02}) + z\bar{z}^2 B(\overline{\phi},h_{11}) + \frac{1}{2}\bar{z}z^2 B(\overline{\phi},h_{20}) \\
& + \frac{1}{6}z^3 C(\phi,\phi,\phi) + \frac{1}{2}z^2\bar{z}C(\overline{\phi},\phi,\phi) + \frac{1}{2}z\bar{z}^2 C(\overline{\phi},\overline{\phi},\phi) + \frac{1}{6}\bar{z}^3 C(\overline{\phi},\overline{\phi},\overline{\phi}) \\
& + \mathcal{O}\left(\|z\|^4\right).
\end{aligned}$$

Here, we have used the multilinearity and the symmetry of B and C. Comparing coefficients of the quadratic terms z^2 and $z\bar{z}$ leads to two non-singular linear equations for h_{20} and h_{11}, namely

$$\begin{aligned}\frac{1}{2}A^{\odot *}h_{20}+B(\phi,\phi)&=i\omega_0 h_{20},\\ A^{\odot *}h_{11}+B(\phi,\overline{\phi})&=0.\end{aligned}$$

Using our expression for B, we can transform our system into the form required by (4.10):

$$\begin{aligned}\left(2i\omega_0 I-A^{\odot *}\right)h_{20}&=D^2f(0)(\phi,\phi)r^{\odot *},\\ A^{\odot *}h_{11}&=-D^2f(0)(\phi,\phi)r^{\odot *}.\end{aligned}$$

Now, both 0 and $2i\omega_0$ are hypothesized to be non-eigenvalues. Furthermore, $r_j^{\odot *}=(e_j,0)$. Therefore, we can apply Corollary 4.4 to these systems, finding the explicit solutions

$$\begin{aligned}h_{20}&=e^{2i\omega_0\vartheta}\Delta(2i\omega_0)^{-1}D^2f(0)(\phi,\phi)\\ h_{11}&=\Delta(0)^{-1}D^2f(0)(\phi,\overline{\phi}),\end{aligned}$$

We can obtain $c_1(0)$ from the system corresponding to the cubic term $z^2\bar{z}$. It reads

$$\left(i\omega_0 I-A^{\odot *}\right)h_{21}=C(\phi,\phi,\overline{\phi})+B(\overline{\phi},h_{20})+2B(\phi,h_{11})-2c_1(0)\phi\equiv\psi^{\odot *}.$$

We now use the Fredholm Alternative (Theorem 4.2) and pair the above expression with our $\phi^{\odot}\in N(i\omega_0 I-A^*)$. We now know that $\left\langle\phi^{\odot},\psi^{\odot *}\right\rangle=0$. Since $\left\langle\phi^{\odot},\phi\right\rangle=1$ by assumption, this leads to

$$c_1(0)=\frac{1}{2}\left\langle\phi^{\odot},C(\phi,\phi,\overline{\phi})+B(\overline{\phi},h_{20})+2B(\phi,h_{11})\right\rangle. \tag{4.15}$$

Now, let's have a look at Table 2.1 again. Note that the right operand of the pairing is an element of $X^{\odot *}$. If we write it in the representation $(\alpha,\eta)\in\mathbb{R}^n\times L^\infty([-h,0],\mathbb{R}^n)$, we see that it only has a component in $\mathbb{R}^n$, because of the definition of B and C (see (4.5)). As we saw in Lemma 3.5, we can assume $\phi^{\odot}$ has the NBV-representation

$$\phi^{\odot}(\vartheta)=p\left(I+\int_0^{\vartheta}\int_{\sigma}^{h}e^{\lambda(\sigma-s)}d\zeta(s)\,d\sigma\right);$$

however, to evaluate the pairing in (4.15), we need to have a $\mathbb{R}^n\times L^1([0,h],\mathbb{R}^n)$ representation (c,g). This can be achieved by setting

$$c=\lim_{\vartheta\downarrow 0}\phi^{\odot *}(\vartheta)=p,\quad g=\dot{\phi}^{\odot *}.$$

The pairing is now given by

$$\langle(c,g),(\alpha,\eta)\rangle=c\alpha+\int_0^h g(\vartheta)\,\eta(-\vartheta)\,d\vartheta.$$

Substituting our expressions, we see that the integral will vanish because $\eta\equiv 0$, and (4.15) becomes

$$\begin{aligned}c_1(0)&=\frac{1}{2}\left\langle\left(p,\dot{\phi}^{\odot *}\right),C(\phi,\phi,\overline{\phi})+B(\overline{\phi},h_{20})+2B(\phi,h_{11})\right\rangle\\ &=\frac{1}{2}\left\langle(p,\dot{\phi}^{\odot *}),\left[D^3f(0)(\phi,\phi,\overline{\phi})+D^2f(0)(\overline{\phi},h_{20})+2D^2f(0)(\phi,h_{11})\right]r^{\odot *}\right\rangle\end{aligned}$$

$$= \frac{1}{2}p\Big[D^3f(0)(\phi,\phi,\overline{\phi}) + D^2f(0)(\overline{\phi},h_{20}) + 2D^2f(0)(\phi,h_{11})\Big].$$

Substituting our expressions for h_{20} and h_{11}, we finally get

$$\begin{aligned} c_1(0) = \frac{1}{2}p\cdot\Big[&D^2f(0)\Big(\overline{\phi},e^{2i\omega_0\vartheta}\Delta(2i\omega_0)^{-1}D^2f(0)(\phi,\phi)\Big)\\ &+2D^2f(0)\Big(\phi,\Delta(0)^{-1}D^2f(0)(\phi,\overline{\phi})\Big)+D^3f(0)(\phi,\phi,\overline{\phi})\Big], \end{aligned}$$

We now have the full derivation of the Hopf critical normal form. All other normal forms and their coefficients will be listed without derivation in the next section (see [10] for more details).

4.4 List of critical normal forms

4.4.1 Codimension 1

Hopf

As we have seen, the (critical) normal form is given by

$$\dot{z} = i\omega_0 z + c_1(0) z^2 \bar{z} + \cdots .$$

The critical normal form coefficient in this expression is $c_1(0)$, but traditionally we use the **First Lyapunov Coefficient** given by

$$l_1(0) = \frac{1}{2\omega_0}\left(c_1(0) + \overline{c_1(0)}\right) = \frac{1}{\omega_0}\text{Re } c_1(0).$$

As derived in the previous section, the coefficient $c_1(0)$ is given by

$$\begin{aligned} c_1(0) = \frac{1}{2} p \cdot \Big[& D^2 f(0)\left(\overline{\phi}, e^{2i\omega_0\vartheta}\Delta(2i\omega_0)^{-1} D^2 f(0)(\phi,\phi)\right) \\ & + 2D^2 f(0)\left(\phi, \Delta(0)^{-1} D^2 f(0)(\phi,\overline{\phi})\right) + D^3 f(0)(\phi,\phi,\overline{\phi})\Big], \end{aligned} \tag{4.16}$$

4.4.2 Codimension 2

Bautin (Generalized Hopf)

If the First Lyapunov Coefficient of a Hopf bifurcation point crosses zero, we say that a Generalized Hopf bifurcation takes place. The standard normal form is given by

$$\dot{z} = (\beta_1(\alpha) + i\omega_0) z + \beta_2(\alpha) z|z|^2 + l_2(0) z|z|^4 + \cdots .$$

Here, $l_2(0)$ is the Second Lyapunov Coefficient. For numerical computations, we use the Poincaré smooth normal form given by

$$\dot{z}(t) = i\omega_0 z + c_1(0) z^2 \bar{z} + c_2(0) z^3 \bar{z}^2 + \cdots .$$

Of course, we assume that $c_1(0) = 0$. The Second Lyapunov Coefficient can be recovered from

$$l_2(0) = \frac{1}{\omega_0}\text{Re } c_2(0).$$

In order to compute it, we again need vectors such that

$$A\phi = 0, \quad A^*\phi^{\odot} = 0, \quad \left\langle \phi^{\odot}, \phi \right\rangle = 1.$$

We reuse some coefficients from the standard Hopf bifurcation:

$$\begin{aligned} h_{20} &= e^{2i\omega_0\vartheta}\Delta(2i\omega_0)^{-1} D^2 f(0)(\phi,\phi), \\ h_{11} &= \Delta(0)^{-1} D^2 f(0)(\phi,\overline{\phi}). \end{aligned}$$

The extra coefficients we need are:

$$h_{30} = e^{3i\omega_0\vartheta}\Delta(3i\omega_0)^{-1}\left[3D^2 f(0)(\phi, h_{20}) + D^3 f(0)(\phi,\phi,\phi)\right],$$

$$
\begin{aligned}
h_{21} &= B^{\mathrm{INV}}_{i\omega_0}\Big[D^3f(0)(\phi,\phi,\overline{\phi})+D^2f(0)(\overline{\phi},h_{20})+2D^2f(0)(\phi,h_{11}),-2c_1(0)\Big],\\
h_{31} &= e^{2i\omega_0\vartheta}\Delta(2i\omega_0)^{-1}\Big[D^2f(0)(\overline{\phi},h_{30})+3D^2f(0)(h_{20},h_{11})+3D^2f(0)(\phi,h_{21})\\
&\quad +3D^3f(0)(\phi,\overline{\phi},h_{20})+3D^3f(0)(\phi,\phi,h_{11})+D^4f(0)(\phi,\phi,\phi,\overline{\phi})\Big]\\
&\quad -6c_1(0)\Delta(2i\omega_0)^{-1}\left[\Delta'(2i\omega_0)-I-\theta\Delta(2i\omega_0)\right]h_{20},\\
h_{22} &= \Delta(0)^{-1}\Big[2D^2f(0)(\overline{\phi},h_{21})+2D^2f(0)(h_{11},h_{11})+2D^2f(0)(\phi,\overline{h_{21}})\\
&\quad +D^2f(0)(h_{20},\overline{h_{20}})+D^3f(0)(\overline{\phi},\overline{\phi},h_{20})+D^3f(0)(\phi,\phi,\overline{h_{20}})\\
&\quad +4D^3f(0)(\phi,\overline{\phi},h_{11})+D^4f(0)(\phi,\phi,\overline{\phi},\overline{\phi})\Big].
\end{aligned}
$$

where we employed the notation for the bordered inverse introduced in Corollary 4.8. The critical coefficient is now given by

$$
\begin{aligned}
c_2 = \frac{1}{12}p\cdot\Big[&6D^2f(0)(h_{11},h_{21})+3D^2f(0)(\overline{h_{21}},h_{20})+3D^2f(0)(\overline{h_{20}},h_{30})\\
&+3D^2f(0)(\phi,h_{22})+2D^2f(0)(\overline{\phi},h_{31})+6D^3f(0)(\overline{\phi},h_{20},h_{11})\\
&+6D^3f(0)(\phi,h_{11},h_{11})+3D^3f(0)(\phi,h_{20},\overline{h_{20}})+6D^3f(0)(\phi,\overline{\phi},h_{21})\\
&+3D^3f(0)(\phi,\phi\overline{h_{21}})+D^3f(0)(\overline{\phi},\overline{\phi},h_{30})+6D^4f(0)(\phi,\phi,\overline{\phi},h_{11})\\
&+3D^4f(0)(\phi,\overline{\phi},\overline{\phi},h_{20})+D^4f(0)(\phi,\phi,\phi,\overline{h_{20}})+D^5f(0)(\phi,\phi,\phi,\overline{\phi},\overline{\phi}).
\end{aligned}
$$

Fold-Hopf

Again, we have a "standard" normal form and a smooth normal form. The standard one is called the Gavrilov normal form and is given by

$$
\begin{cases}
\dot{z}_0 &= \delta(\alpha)+b(\alpha)z_0^2+c(\alpha)|z_1|^2+\cdots,\\
\dot{z}_1 &= \sigma(\alpha)z_1+d(\alpha)z_0z_1+e(\alpha)z_0^2z_1+\cdots,
\end{cases}
\qquad z_0\in\mathbb{R},z_1\in\mathbb{C}.
$$

We use the Poincaré smooth normal form, given by

$$
\begin{cases}
\dot{z}_0 &= g_{200}z_0^2+g_{011}|z_1|^2+g_{300}z_0^3+g_{111}z_0|z_1|^2+\cdots,\\
\dot{z}_1 &= i\omega_0z_1+g_{110}z_0z_1+g_{210}z_0^2z_1+g_{021}z_1|z_1|^2+\cdots,
\end{cases}
\qquad z_0\in\mathbb{R},z_1\in\mathbb{C}.
$$

The coefficients g_{jkl} are real in the first and complex in the second equation. The relationships between the coefficients of the first and the second normal form are:

$$
\begin{aligned}
&b(0)=g_{200},\quad c(0)=g_{011},\quad d(0)=g_{110}-i\omega_0\frac{g_{300}}{g_{200}},\\
&e(0)=\mathrm{Re}\left[g_{210}+g_{110}\left(\frac{\mathrm{Re}\,g_{021}}{g_{011}}-\frac{3g_{300}}{2g_{200}}+\frac{g_{111}}{2g_{011}}\right)-\frac{g_{021}g_{200}}{g_{011}}\right].
\end{aligned}
$$

Two quantities that characterize the bifurcation are

$$
s\equiv g_{200}g_{011},\qquad \theta\equiv\frac{\mathrm{Re}\,g_{110}}{g_{200}}.
$$

In this case, we need two pairs of eigenvectors:

$$
A\phi_0=0,\quad A\phi_1=i\omega_0\phi_1,\quad A^*\phi_0^{\odot}=0,\quad A^*\phi_1^{\odot}=i\omega_0\phi_1^{\odot},\quad \left\langle\phi_i^{\odot},\phi_j\right\rangle=\delta_{ij}.
$$

These can be used to compute some center manifold coefficients:

$$
\begin{aligned}
h_{200} &= B_0^{\mathrm{INV}}\big[D^2f(0)(\phi_0,\phi_0), -p_0\cdot D^2f(0)(\phi_0,\phi_0)\big],\\
h_{020} &= e^{2i\omega_0\vartheta}\Delta(2i\omega_0)^{-1}D^2f(0)(\phi_1,\phi_1),\\
h_{110} &= B_{i\omega_0}^{\mathrm{INV}}\big[D^2f(0)(\phi_0,\phi_1), -p_1\cdot D^2f(0)(\phi_0,\phi_1)\big],\\
h_{011} &= B_0^{\mathrm{INV}}\Big[D^2f(0)(\phi_1,\overline{\phi}_1), -p_0\cdot D^2f(0)(\phi_1,\overline{\phi}_1)\Big].
\end{aligned}
$$

This leads to expressions for the critical normal form coefficients:

$$
\begin{aligned}
g_{200} &= \frac{1}{2}p_0\cdot D^2f(0)(\phi_0,\phi_0),\\
g_{110} &= p_1\cdot D^2f(0)(\phi_0,\phi_1),\\
g_{011} &= p_0\cdot D^2f(0)(\phi_1,\overline{\phi}_1),\\
g_{300} &= \frac{1}{6}p_0\cdot\big[3D^2f(0)(\phi_0,h_{200}) + D^3f(0)(\phi_0,\phi_0,\phi_0)\big],\\
g_{111} &= p_0\cdot\Big[D^2f(0)(\phi_0,h_{011}) + D^2f(0)(\overline{\phi}_1,h_{110}) + D^2f(0)(\phi_1,\overline{h_{110}}) + D^3f(0)(\phi_0,\phi_0,\overline{\phi}_1)\Big],
\end{aligned}
$$

$$
\begin{aligned}
g_{210} &= \frac{1}{2}p_1\cdot\big[D^2f(0)(\phi_1,h_{200}) + 2D^2f(0)(\phi_0,h_{110}) + D^3f(0)(\phi_0,\phi_0,\phi_1)\big],\\
g_{021} &= \frac{1}{2}p_1\cdot\Big[D^2f(0)(\overline{\phi}_1,h_{020}) + 2D^2f(0)(\phi_1,h_{011}) + D^3f(0)(\phi_1,\phi_1,\overline{\phi}_1)\Big].
\end{aligned}
$$

Double Hopf

The Poincaré smooth normal form is given by

$$
\begin{cases}
\dot{z}_1 &= i\omega_1 z_1 + g_{2100}z_1|z_1|^2 + g_{1011}z_1|z_2|^2 + g_{3200}z_1|z_1|^4\\
&\quad + g_{2111}z_1|z_1|^2|z_2|^2 + g_{1022}z_1|z_2|^4 + \cdots,\\
\dot{z}_2 &= i\omega_2 z_2 + g_{1110}z_2|z_1|^2 + g_{0021}z_2|z_2|^2 + g_{2210}z_2|z_1|^4\\
&\quad + g_{1121}z_2|z_1|^2|z_2|^2 + g_{0032}z_2|z_2|^4 + \cdots
\end{cases}
\qquad z_1\in\mathbb{C}, z_2\in\mathbb{C}.
$$

Here, the coefficients g_{jklm} are all complex. Two quantities that characterize the bifurcation are

$$
\theta(0) \equiv \frac{\mathrm{Re}\ g_{1011}}{\mathrm{Re}\ g_{0021}}, \qquad \delta(0) \equiv \frac{\mathrm{Re}\ g_{1110}}{\mathrm{Re}\ g_{2100}}.
$$

The center manifold coefficients are given by

$$
\begin{aligned}
h_{1100} &= \Delta(0)^{-1}D^2f(0)(\phi_1,\overline{\phi}_1),\\
h_{2000} &= e^{2i\omega_1\vartheta}\Delta(2i\omega_1)^{-1}D^2f(0)(\phi_1,\phi_1),\\
h_{1010} &= e^{i(\omega_1+\omega_2)\vartheta}\Delta(i(\omega_1+\omega_2))^{-1}D^2f(0)(\phi_1,\phi_2),\\
h_{1001} &= e^{i(\omega_1-\omega_2)\vartheta}\Delta(i(\omega_1-\omega_2))^{-1}D^2f(0)(\phi_1,\overline{\phi}_2),\\
h_{0020} &= e^{2i\omega_2\vartheta}\Delta(2i\omega_2)^{-1}D^2f(0)(\phi_2,\phi_2),\\
h_{0011} &= \Delta(0)^{-1}D^2f(0)(\phi_2,\overline{\phi}_2),\\
h_{3000} &= e^{3i\omega_1\vartheta}\Delta(3i\omega_1)^{-1}\big[3D^2f(0)(h_{2000},\phi_1) + D^3f(0)(\phi_1,\phi_1,\phi_1)\big],\\
h_{2010} &= e^{i(2\omega_1+\omega_2)\vartheta}\Delta(i(2\omega_1+\omega_2))^{-1}\Big[2D^2f(0)(h_{1010},\phi_1)
\end{aligned}
$$

$$
\begin{aligned}
&\quad + D^2f(0)(h_{2000}, \phi_2) + D^3f(0)(\phi_1, \phi_1, \phi_2)\Big],\\
h_{2001} &= e^{i(2\omega_1-\omega_2)\vartheta}\Delta(i(2\omega_1-\omega_2))^{-1}\Big[2D^2f(0)(h_{1001}, \phi_1)\\
&\quad + D^2f(0)(h_{2000}, \overline{\phi}_2) + D^3f(0)(\phi_1, \phi_1, \overline{\phi}_2)\Big],\\
h_{1020} &= e^{i(\omega_1+2\omega_2)\vartheta}\Delta(i(\omega_1+2\omega_2))^{-1}\Big[2D^2f(0)(h_{1010}, \phi_2)\\
&\quad + D^2f(0)(\overline{h_{0020}}, \phi_1) + D^3f(0)(\phi_1, \phi_2, \phi_2)\Big],\\
h_{1002} &= e^{i(\omega_1-2\omega_2)\vartheta}\Delta(i(\omega_1-2\omega_2))^{-1}\Big[2D^2f(0)(h_{1001}, \overline{\phi}_2)\\
&\quad + D^2f(0)(\overline{h_{0020}}, \phi_1) + D^3f(0)(\phi_1, \overline{\phi}_2, \overline{\phi}_2)\Big],\\
h_{0030} &= e^{3i\omega_2\vartheta}\Delta(3i\omega_2)^{-1}\left[3D^2f(0)(h_{0020}, \phi_2) + D^3f(0)(\phi_2, \phi_2, \phi_2)\right].
\end{aligned}
$$

This leads to the following critical normal form coefficients:

$$
\begin{aligned}
g_{2100} &= \frac{1}{2}p_1 \cdot \Big[2D^2f(0)(h_{1100}, \phi_1) + D^2f(0)(h_{2000}, \overline{\phi}_1) + D^3f(0)(\phi_1, \phi_1, \overline{\phi}_1)\Big],\\
g_{1011} &= p_1 \cdot \Big[D^2f(0)(h_{0011}, \phi_1) + D^2f(0)(h_{1001}, \phi_2) + D^2f(0)(h_{1010}, \overline{\phi}_2) + D^3f(0)(\phi_1, \phi_2, \overline{\phi}_2)\Big],\\
g_{1110} &= p_2 \cdot \Big[D^2f(0)(\overline{h_{1001}}, \phi_1) + D^2f(0)(h_{1010}, \overline{\phi}_1) + D^2f(0)(h_{1100}, \phi_2) + D^3f(0)(\phi_1, \overline{\phi}_1, \phi_2)\Big],\\
g_{0021} &= \frac{1}{2}p_2 \cdot \Big[2D^2f(0)(h_{0011}, \phi_2) + D^2f(0)(h_{0020}, \overline{\phi}_2) + D^3f(0)(\phi_2, \phi_2, \overline{\phi}_2)\Big].
\end{aligned}
$$

Chapter 5

Additions to DDE-BIFTOOL

DDE-BIFTOOL is a Matlab tool for bifurcation analysis of DDEs, designed by Koen Engelborghs. It is able to analyze DDEs with discrete, fixed delays and discrete delays depending on the state varables and the parameters. Both the package and its manual are freely available (see [4]). The main objective of this thesis is of course to extend DDE-BIFTOOL with ways to compute normal forms. In this chapter, we will describe how this was done and what other functionality has been added. Among its intended audience are current users of DDE-BIFTOOL who want to know what has changed in existing subroutines and how the new subroutines should be used. Chapter 6 contains concrete examples.

5.1 Bifurcation detection

5.1.1 Global setup

Our first aim was to be able to detect various bifurcations. To do this, it was necessary to slightly modify the `br_contn`-routine.

The call to the detection routine is as follows:

`[newbranch, success] = br_bifdet(branch)`

Here, `branch` is the current branch (the last point of which is the newly detected point). The output parameter `newbranch` is the original branch augmented with a newly found bifurcation point (if one was found), and `success` tells us whether a bifurcation was found and successfully added to the branch.

The detection routine needs a few configuration parameters. These have been collected in the new substructure `method.bifurcation`. An overview can be found in Table 5.1.

If the detection flag is set, the detection routine is called after a new point has been found during the continuation process. DDE-BIFTOOL in its original form already can compute the stability of a point; `br_bifdet` simply uses this existing functionality (`p_stabil`) to compute the eigenvalues at the newly found point. It only keeps the eigenvalues of which the real part is greater or equal than the parameter `method.bifurcation.minimal_real_part`; tweaking this minimum may be necessary because of memory overflow in the detection routines. After computing the roots, the detection algorithm then applies certain test functions to

Parameter	Default	Description
`detect`	1	Toggle detection on/off
`minimal_real_part`	−0.1	Minimal real part of the characteristic roots used during detection
`correction_tolerance`	10^{-7}	Custom value for `method.point.minimal_accuracy` during codimension 1 (Hopf) bifurcation correction
`radial_tolerance_factor`	0.25	The maximum distance to the branch the bifurcation point may have relative to the distance between the last two points
`secant_iterations`	30	The maximum number of iterations for the secant method used to correct codimension 2 points
`secant_tolerance`	10^{-9}	The minimal accuracy for the secant method

TABLE 5.1: *Fields of the structure* `method.bifurcation`*, which is a substructure of a* `branch` *structure.*

check whether a bifurcation is happening. If the sign of one of these test functions changes, a bifurcation is happening. (See below.)

If, according to one of the test functions, the roots indicate a bifurcation, we construct a new point halfway between the last and the second to last branch point. Now there are two options: for codim 1 bifurcations (i.e. Hopf), we simply use the existing functionality of `p_correc` to correct the midpoint to a true bifurcation point. For codim 2 bifurcations, we use a secant method to arrive at a point where the test function is close enough to zero. The maximum number of iterations and the minimal accuracy can be set using the parameters `method.bifurcation.secant_iterations` and `method.bifurcation.secant_tolerance`, respectively.

There is a possibility that the correction fails. The reason for this is probably a false positive from the specific test function. In this case, the detection algorithm issues a warning and continues to check for other bifurcations (if there are any left on its list). Do note that it may happen that `p_correc` is too strict; in this case, modifying its tolerance is possible via the parameter `method.bifurcation.correction_tolerance`.

If the correction succeeds, a second possibility for errors is that the newly found point does not lie on the branch. By default, the point is considered "away from the branch" if its distance to the line connecting the last and second to last points on the branch is more than 25% of the length of this line. If this is the case, the detection algorithm again issues a warning and stops. This radial tolerance factor can be modified by specifying the parameter `method.bifurcation.radial_tolerance_factor`. In order to compute the distances between points, we added an inner product function:

`ip = p_inprod(p1, p2)`
: Computes the inner product between the two points `p1` and `p2`. It was based on the `p_norm` function and constructed in such a way that `p_inprod(p1,p1) = p_norm(p1)`^{2}.

Field	Content
kind	'genh'
flag	''
parameter	$\mathbb{R}^{1\times p}$
x	$\mathbb{R}^{n\times 1}$
omega	$\mathbb{R}$
stability	empty or struct
nvec.p	$\mathbb{R}^{n\times 1}$
nvec.q	$\mathbb{R}^{n\times 1}$
nmfm	empty or struct

Field	Content
kind	'zeho'
flag	''
parameter	$\mathbb{R}^{1\times p}$
x	$\mathbb{R}^{n\times 1}$
omega	$\mathbb{R}$
stability	empty or struct
nvec.p	$\mathbb{R}^{n\times 1}$
nvec.q	$\mathbb{R}^{n\times 1}$
nmfm	empty or struct

Field	Content
kind	'hoho'
flag	''
parameter	$\mathbb{R}^{1\times p}$
x	$\mathbb{R}^{n\times 1}$
omega1	$\mathbb{R}$
omega2	$\mathbb{R}$
stability	empty or struct
nvec.p	$\mathbb{R}^{n\times 1}$
nvec.q	$\mathbb{R}^{n\times 1}$
nmfm	empty or struct

TABLE 5.2: *New point structure types for bifurcation points.*

If all is well, the bifurcation point is added to the branch. (Not without difficulty, see Section 5.4.) The point is placed between the last and second to last points of the branch.

Note: in the current implementation, if the correction as a bifurcation point fails, detection is resumed, i.e. the remaining test functions are evaluated until another bifurcation is detected or until the list of test functions is exhausted. On the other hand, if a corrected bifurcation point is not on the branch, detection is *not* resumed.

5.1.2 Bifurcation point types

If a Hopf bifurcation is detected on a steady state branch, the bifurcation candidate is converted into a `hopf` point and corrected. In the same spirit, if a bifurcation is detected on a `hopf` branch, the point is converted to one of the following new point types: `genh`, `zeho` and `hoho`, containing Generalized Hopf, Zero-Hopf and Double Hopf points respectively.

At present, `genh` points and `zeho` points only differ from `hopf` points by their `kind` label. Only `hoho` has a clear difference: instead of one field `omega`, it features two fields `omega1` and `omega2`. For completeness, Table 5.2 shows the full structure specifications (also including the new fields that have been added to *all* point structures).

Of course, new point types need new conversion functions:

`[genh, success] = p_togenh(hopf)`

Converts a point of `hopf` type to a point of `genh` type. Currently, this only copies the point and makes sure that the `flag` and `nmfm` fields are set. If `point.kind` is not equal

to `hopf`, `success` is set to `0`, otherwise `1`.

`[hoho, success] = p_tohoho(hopf)`
: Converts a point of `hopf` type to a point of `hoho` type. It copies the point and removes the `omega` field, but sets its value as `hoho.omega1`. It then looks for the second imaginary pair and sets its frequency as `hoho.omega2`. If `point.kind` is not equal to `hopf` or no other imaginary pair is found, `success` is set to `0`, otherwise `1`.

`[zeho, success] = p_tozeho(hopf)`
: Converts a point of `hopf` type to a point of `zeho` type. Currently, this only copies the point and makes sure that the `flag` and `nmfm` fields are set. If `point.kind` is not equal to `hopf`, `success` is set to `0`, otherwise `1`.

5.1.3 Test functions

Here we list the test functions used to detect bifurcations.

Hopf

Given a set of roots $\{\lambda_j\}_{j=1}^{n}$, we compute the product of sums

$$\Lambda = \prod_{i<j} (\lambda_i + \lambda_j).$$

A sign change in this quantity signifies a bifurcation.

We have encapsulated this function in the `nmfm_hopfdet` routine:

`sign = nmfm_hopfdet(roots)`
: This function computes the quantity Λ as a function of the eigenvalues in `roots` and normalizes the `sign` to $+1$ or -1.

It can happen that, due to memory overflow, the product becomes infinite. In this case, a warning is issued. The solution is to lower the minimal real part of the roots.

Generalized Hopf

The test function for the Generalized Hopf bifurcation is simply the First Lyapunov Coefficient of the Hopf bifurcation. See below for more information on this subject.

Zero-Hopf

For the Zero-Hopf bifurcation, we look at the sign of the smallest real eigenvalue. To select this eigenvalue, there is an auxiliary function:

`[smallest_real_part, fullroots] = nmfm_smlrp(point, method, remove_pair)`
: This function selects the smallest real eigenvalue of `point`, which must be of `hopf` type.

If no stability information is present, the eigenvalues are computed using `method`. If `remove_pair` is set to `1`, then the eigenvalue pair $\pm i\omega$ is removed from the roots before the selection is made. The field `point.omega` is used for this. If wanted, the newly computed roots are returned as `fullroots`.

Double Hopf

For the Double Hopf bifurcation, we look at the sign of the smallest real part of the complex eigenvalue pairs. To this end, there is an auxiliary function:

`[smallest_real_part, fullroots] = nmfm_smlrpip(point, method, remove_pair)`
This function selects the smallest real part of the complex eigenvalue pairs of `point`, which must be of `hopf` type. If no stability information is present, the eigenvalues are computed using `method`. If `remove_pair` is set to `1`, then the eigenvalue pair $\pm i\omega$ is removed from the roots before the selection is made. The field `point.omega` is used for this. If wanted, the newly computed roots are returned as `fullroots`.

5.2 Computation of normal form coefficients

We have implemented computation of the normal form coefficients of Hopf, Generalized Hopf, Double Hopf and Zero-Hopf. The implementation works fairly similar, so in this section, we focus on the First Lyapunov Coefficient (which is the normal form coefficient for Hopf bifurcations).

5.2.1 The abstract expression

When a Hopf bifurcation is detected, the First Lyapunov Coefficient is computed. To do this, we had to convert the expression

$$\begin{aligned} c_1(0) = \frac{1}{2}p \cdot \Big[& D^2 f(0)\left(\overline{\phi}, e^{2i\omega_0\vartheta}\Delta(2i\omega_0)^{-1}D^2 f(0)(\phi,\phi)\right) \\ & + 2D^2 f(0)\left(\phi, \Delta(0)^{-1}D^2 f(0)(\phi,\overline{\phi})\right) + D^3 f(0)(\phi,\phi,\overline{\phi})\Big], \end{aligned} \tag{5.1}$$

into a form that allows us to evaluate it numerically. If we have this number, the First Lyapunov Coefficient is given by

$$l_1(0) = \frac{1}{2\omega_0}\left(c_1(0) + \overline{c_1(0)}\right). \tag{5.2}$$

Equation (4.16) contains the following elements:

(1) The characteristic matrix $\Delta(\lambda)$;

(2) The critical eigenvalue $\lambda_0 = i\omega_0$ (characteristic of the Hopf bifurcation);

(3) Vectors p and q such that $\Delta(\lambda_0)q = 0 = p\Delta(\lambda_0)$ and $pq = 1$;

(4) The function f, used in $\dot{x}(t) = f(x_t, \alpha)$;

(5) Derivatives $D^j f$ of this function;

(6) Eigenfunctions ϕ and $\overline{\phi}$ of A such that $\left\langle \phi, \overline{\phi} \right\rangle = 1$.

Some of these quantities are still too abstract to use in a numerical implementation. One of the big problems is: our f here is still defined as a function $C([-h,0],\mathbb{R}^n) \times \mathbb{R}^m \to \mathbb{R}^n$. Below, we present a framework to solve this problem.

5.2.2 The numerical framework

In this section, we present a framework to convert theoretical, abstract expressions from sun-star calculus into workable DDE-BIFTOOL equivalents. It is based on [10], Section 4.2.3. As was mentioned earlier, we assume that we have discrete delays.

Consider the DDE

$$\dot{x}(t) = f(x_t),$$

where $f : C([-h,0],\mathbb{R}^n) \to \mathbb{R}^n$ is at least five times continuously differentiable. Let n be the dimension of the system. Suppose that we only have r discrete delays, i.e. we have

$$0 = \tau_0 < \tau_1 < \tau_2 < \cdots < \tau_r = h.$$

(This means that we formally have $r+1$ delays, including the zero delay.)

Now, given a function $\varphi \in C([-h,0],\mathbb{R}^n)$ (i.e. a candidate to apply f to), introduce $n\cdot(r+1)$ continuous functions $\varphi_j^k : [-h,0] \to \mathbb{R}$, with

$$\varphi_j^k = \varphi_j(-\tau_k)$$

(i.e. φ_j^k denotes the j-th coordinate function of φ applied to $-\tau_k$). We group them together in the $n \times (r+1)$ matrix

$$\Phi \equiv \left\{\varphi_j^k\right\}_{\substack{j=1\\k=0}}^{\substack{n\\r}} \equiv \begin{pmatrix} \varphi_1(-\tau_0) & \cdots & \varphi_1(-\tau_r) \\ \varphi_2(-\tau_0) & \cdots & \varphi_2(-\tau_r) \\ \vdots & & \vdots \\ \varphi_n(-\tau_0) & \cdots & \varphi_n(-\tau_r) \end{pmatrix}.$$

Now, recall that we only use DDE-BIFTOOL for systems with discrete delays. This means that there exists a function $G \in C^5(\mathbb{R}^{n(r+1)},\mathbb{R}^n)$, i.e. a function operating on numbers instead of functions, such that

$$f(\varphi) = G(\Phi).$$

This is actually the way DDE-BIFTOOL already treats its right hand side functions. For an illustration of this fact, see Example 5.1.

Example 5.1. Suppose we have the following system of DDEs:

$$\begin{aligned} \dot{x}_1(t) &= x_1(t)^2 - x_2(t-\tau_1) + 3x_1(t-\tau_2)^2 \\ \dot{x}_2(t) &= x_2(t)^2 + x_1(t-\tau_1) - 3x_2(t-\tau_2)^2 \end{aligned}$$

This means $n = 2$ and $r = 2$. Then our function G is given by the coordinate functions

$$G_1(\Phi) = (\varphi_1^0)^2 - \varphi_2^1 + 3\left(\varphi_1^2\right)^2$$
$$G_2(\Phi) = (\varphi_2^0)^2 + \varphi_1^1 - 3\left(\varphi_2^2\right)^2$$

and in fact, we enter the following in the `sys_rhs` file (but note that we start numbering the delays at 1):

$$\texttt{f}(1,1) = \texttt{xx}(1,1)^2 - \texttt{xx}(2,2) + 3 * \texttt{xx}(1,3)^2$$
$$\texttt{f}(2,1) = \texttt{xx}(2,1)^2 + \texttt{xx}(1,2) - 3 * \texttt{xx}(2,3)^2$$

5.2.3 The numerical expression

Using the framework outlined above, we can rewrite all abstract quantities involving f to real-valued quantities involving G. As it happens, only the critical eigenvalue $\lambda_0 = i\omega_0$ is available from the start without any conversion.

The first rewrite is simple: we replace $f(\varphi)$ by $G(\Phi)$. The derivatives of f require more care. For $\ell \in \mathbb{N}$, the ℓ-th order derivative $D^\ell f(0)$ of f at zero is a bounded ℓ-linear form from $C([-h,0],\mathbb{R}^n)$ to $\mathbb{R}^n$. We denote the derivative at zero of the i-th component of G with respect to its (j,k)-th variable (i.e. to φ_j^k) by $D_j^k G_i(0) \in \mathbb{R}$. The second derivative at zero with respect to $\varphi_{j_1}^{k_1}$ and $\varphi_{j_2}^{k_2}$ is denoted by $D_{j_1 j_2}^{k_1 k_2} G_i(0) \in \mathbb{R}$, and so on for all higher-order derivatives. So these derivatives take a certain number of functions $C([-h,0],\mathbb{R}^n)$ and produce a vector in $\mathbb{R}^n$.

For instance, when $\ell = 1$, we have, for $\varphi \in C([-h,0])$,

$$D^1 f_i(0)\varphi = \sum_{j_1=1}^{n}\sum_{k_1=0}^{r} D_{j_1}^{k_1} G_i(0)\varphi_{j_1}^{k_1} \equiv D_{j_1}^{k_1} G_i(0)\varphi_{j_1}^{k_1},$$

where we have temporarily introduced the convention that repeated indices imply summation. It will prove convenient to condense n of these coordinate expressions into one n-dimensional vector equation (i.e. dropping the i):

$$D^1 f(0)\varphi = D_{j_1}^{k_1} G(0)\varphi_{j_1}^{k_1}. \tag{5.3}$$

This form will be implemented in DDE-BIFTOOL.

The shorthand notation allows us to express all higher order derivatives rather efficiently. We list them here up to order five, as this is the highest number of derivatives required for normal form computations.

$$D^2 f(0)(\varphi,\psi) = D_{j_1 j_2}^{k_1 k_2} G(0)\varphi_{j_1}^{k_1}\psi_{j_2}^{k_2}, \tag{5.4}$$
$$D^3 f(0)(\varphi,\psi,\chi) = D_{j_1 j_2 j_3}^{k_1 k_2 k_3} G(0)\varphi_{j_1}^{k_1}\psi_{j_2}^{k_2}\chi_{j_3}^{k_3}, \tag{5.5}$$
$$D^4 f(0)(\varphi,\psi,\chi,\zeta) = D_{j_1 j_2 j_3 j_4}^{k_1 k_2 k_3 k_4} G(0)\varphi_{j_1}^{k_1}\psi_{j_2}^{k_2}\chi_{j_3}^{k_3}\zeta_{j_4}^{k_4}, \tag{5.6}$$
$$D^5 f(0)(\varphi,\psi,\chi,\zeta,\eta) = D_{j_1 j_2 j_3 j_4 j_5}^{k_1 k_2 k_3 k_4 k_5} G(0)\varphi_{j_1}^{k_1}\psi_{j_2}^{k_2}\chi_{j_3}^{k_3}\zeta_{j_4}^{k_4}\eta_{j_5}^{k_5}. \tag{5.7}$$

We now turn our attention to the characteristic matrix $\Delta(\lambda)$, which will be evaluated at different values of λ. For discrete delays, formula (3.7) becomes a finite sum of matrices:

$$\Delta(\lambda) = \lambda I - \sum_{k=0}^{r} A_k e^{-\lambda\tau_k},$$

where r is the number of delays, τ_k is the k-th delay and

$$A_k = \begin{pmatrix} D_1^k G(0) & \cdots & D_n^k G(0) \end{pmatrix} = \begin{pmatrix} D_1^k G_1(0) & \cdots & D_n^k G_1(0) \\ \vdots & & \vdots \\ D_1^k G_n(0) & \cdots & D_n^k G_n(0) \end{pmatrix}, \tag{5.8}$$

i.e. the derivative matrix with respect to all variables corresponding to the k-th delay.

Now we have the characteristic matrix, the vectors p and q can simply be constructed out of the nullspaces of $\Delta(\lambda_0)$ and $\Delta(\lambda_0)^T$. If q_0 and p_0 are arbitrary vectors satisfying $\Delta(\lambda_0)q_0 = 0 = p_0\Delta(\lambda_0)$, we can normalize them by putting

$$\beta = \frac{1}{\sqrt{p_0\Delta'(\lambda_0)q_0}}, \quad q = \beta q_0, \quad p = \beta p_0.$$

This will automatically lead to normalized eigenfunctions, because of Lemma 3.5. However, this means that we need to numerically have the derivative of the characteristic matrix, i.e.

$$\Delta'(\lambda) = I + \sum_{k=0}^{r} \tau_k A_k e^{-\lambda\tau_k}. \tag{5.9}$$

Other normal form coefficients will need the second derivative as well, i.e.

$$\Delta''(\lambda) = I - \sum_{k=0}^{r} \tau_k^2 A_k e^{-\lambda\tau_k}. \tag{5.10}$$

The eigenfunction ϕ is given by

$$\phi(\vartheta) = e^{\lambda_0\vartheta} q = e^{i\omega_0\vartheta} q, \tag{5.11}$$

and hence

$$\overline{\phi}(\vartheta) = e^{-i\omega_0\vartheta}\overline{q}.$$

Evaluated at $\vartheta = \tau_0, \ldots, \tau_r$, these can then be used as input for the various derivatives.

5.2.4 DDE-BIFTOOL implementation

In order to compute the multilinear forms, an extra system specification was developed. In the new version of DDE-BIFTOOL, a derivative file should implement the following function:

`function y = sys_mfderi(xx,par,varargin)`

This function should implement the equations (5.3), (5.4), (5.5), (5.6) and (5.7) all at once. The variable arguments here are the vectors $\phi, \psi, \chi, \zeta, \eta$ appearing in these equations. The number of variable arguments determines which derivative is returned. The result is an n-dimensional vector.

A Maple script is available to automatically generate the Matlab file if the right hand side is specified.

The original function `sys_deri` is still called by all "old" subroutines. This file, as well as the `sys_rhs` file, can also be generated by the Maple script.

In order to facilitate the normal form computations, several auxiliary subroutines have been added. They are prefixed by `nmfm`.

`Delta = nmfm_charmat(xx,par,lambda)`
: This function computes the characteristic matrix $\Delta($`lambda`$)$ as described in Equations (3.7) and (5.8).

`DDelta = nmfm_charmatderi(xx,par,lambda)`
: This function computes the derivative of the characteristic matrix $\Delta($`lambda`$)$ as described in Equations (5.9) and (5.8).

`D2Delta = nmfm_charmatderi2(xx,par,lambda)`
: This function computes the second derivative of the characteristic matrix $\Delta($`lambda`$)$ as described in Equations (5.10) and (5.8).

`PHI = nmfm_handletomatrix(fn, arg)`
: This function takes a n-dimensional vector-valued function handle `fn` and an r-dimensional vector `arg` and produces an $n \times r$-matrix

$$\big(\texttt{fn(arg(1))} \quad \cdots \quad \texttt{fn(arg(r))}\big).$$

It is used to convert a function such as ϕ, given in Equation (5.11), to a matrix on which the derivative can operate (as in Equation (5.3)). For this it is necessary to put the list of r delays in `arg`.

`BINV = nmfm_binv(xx, par, lambda, q, p, zeta, kappa)`
: This function is used in the computation of codimension 2 bifurcations. It computes the bordered inverse as computed in Corollary 4.8.

And last but not least, the subroutines which actually compute the normal form coefficients.

`point = nmfm_hopf(newpoint, oldpoint)`
: This function computes the First Lyapunov Coefficient for the Hopf bifurcation occurring at `newpoint` (which should be of type `hopf`) and stores it in `point.nmfm.L1`. It simply evaluates Equations (5.2) and (5.1) using the machinery explained above. If the optional argument is provided, `oldpoint` is used in the computation of the null vectors (see below).

`point = nmfm_genh(newpoint, oldpoint)`
: This function computes the Second Lyapunov Coefficient for the Generalized Hopf bifurcation occurring at `newpoint` (which should be of type `genh`) and stores it in `point.nmfm.L2`. If the optional argument is provided, `oldpoint` is used in the computation of the null vectors (see below).

`point = nmfm_zeho(newpoint, oldpoint)`
: This function computes the various normal form coefficients for the Zero-Hopf bifurcation occurring at `newpoint` (which should be of type `zeho`). They are stored in

Type	Fields
hopf	L1
genh	L2
zeho	g200, g110, g011, g300, g111, g210, g021, a, b, c, d, e, s, theta
hoho	g2100, g1011, g1110, g0021, theta, delta

TABLE 5.3: *Normal form coefficients as stored in the* `point.nmfm` *structure.*

`point.nmfm`. If the optional argument is provided, `oldpoint` is used in the computation of the null vectors (see below).

`point = nmfm_hoho(newpoint, oldpoint)`
This function computes the various normal form coefficients for the Double Hopf bifurcation occurring at `newpoint` (which should be of type `hoho`). They are stored in `point.nmfm`. If the optional argument is provided, `oldpoint` is used in the computation of the null vectors (see below).

Note: the function `br_bifdet` will automatically call these subroutines.

5.2.5 Storage of the normal form coefficients

As was already indicated above, the critical normal form coefficients are stored in the new point field structure `nmfm`. This fields contains all normal form coefficients as elements. For the specifics, see Table 5.3.

All point manipulation routines have been updated to ensure that the `nmfm` field is always present in all point types. Note, however, that not all point manipulation routines actually *compute* the normal form coefficients when creating a new point: they leave the field empty. We chose to do this in order to reduce overhead in contexts where the normal form coefficients are not specifically wanted. The only place where the normal form coefficients are set is in `br_contn` if bifurcation detection is enabled.

5.2.6 Bordering technique for null vectors

All normal form computations require a number of null vectors, satisfying $p\Delta(\lambda) = 0$ and $\Delta(\lambda)q = 0$. When continuing a Hopf branch, it is necessary to have these vectors at every step, because we have to compute L_1. In order to compute the null vectors robustly and efficiently, we have implemented a bordering technique. We assume that some approximate null vectors p_0 and q_0 are given. To get our new vector q, we solve the system

$$\begin{pmatrix} \Delta(\lambda) & p_0 \\ \overline{q_0}^T & 0 \end{pmatrix} \begin{pmatrix} q \\ s \end{pmatrix} = \begin{pmatrix} 0_n \\ 1 \end{pmatrix}$$

and for p we solve

$$\begin{pmatrix} p & s \end{pmatrix} \begin{pmatrix} \Delta(\lambda) & p_0 \\ \overline{q_0}^T & 0 \end{pmatrix} \begin{pmatrix} q \\ s \end{pmatrix} = \begin{pmatrix} 0_n & 1 \end{pmatrix}.$$

At the first point of the branch, we compute the null vectors of $\Delta(i\omega)$ directly by Matlab's `null` function and use them as input for the bordering technique. It might happen that the

`null` function fails; in that case, we compute all eigenvectors and eigenvalues with the built-in `eig` function (using QZ-decomposition) and select the eigenvector corresponding to the zero eigenvalue.

For all subsequent points in the branch, we use the old null vectors as input ($p \to p_0, q \to q_0$). To facilitate this, another new field has been added to the `hopf` structure: we now have access to `hopf.nvec.p` and `hopf.nvec.q`. If an old point is provided to the normal form routine, these vectors are used in the bordering technique. If no such point is provided, the `null` result is used as input. The `br_contn` routine has been updated to always use the bordering technique.

5.3 Correction of codim 2 bifurcations

If a codim 2 bifurcation is detected, we want to zoom in to the actual bifurcation point. This is straightforwardly done by a secant method. For completeness, we briefly describe the process:

(1) Start with one point at which a test function is negative and one point at which the test function is positive

(2) Construct the point halfway between the positive and the negative point

(3) Compute the test function at this point

(4) If the test function is negative, make this point the new negative point; otherwise, the new positive point

(5) Repeat until the absolute value of the test function is smaller than some predefined tolerance

This secant method has been incorporated into the routine `br_bifdet`. For Generalized Hopf, the test function is the First Lyapunov Coefficient, for Zero-Hopf, the test function is the smallest real eigenvalue, and for Double Hopf, the test function is the smallest real part of the complex eigenvalue pairs. By default, the tolerance mentioned above is the same as that for `hopf`, namely 10^{-9}.

5.4 Flagging bifurcation points

When one has a branch of steady states on which a Hopf bifurcation occurs, one typically wants to plot the branch as a line and give the Hopf point a special marker (e.g. a big dot). Ideally, one simply passes an entire branch to a plot routine, along with an instruction how to display the various bifurcations occurring along a branch.

Within the architecture of Matlab, this task provided a challenge. As DDE-BIFTOOL branches are stored as simple arrays of point-structures, it was not possible to have a branch of `stst` points containing a `hopf` point as well, because different types of structures cannot occur in the same array. Therefore, we chose to introduce an extra field to all point structures, called `flag`, to store information on what kind of bifurcation occurs at this point. For example, a `stst` can have `stst.flag = 'hopf'` or a `hopf` can have `hopf.flag = 'genh'` (for Generalized Hopf).

To make this work, two changes had to be made throughout all DDE-BIFTOOL source files. First, all subroutines creating new point structures have been modified to also set `point.flag = ''` (so that all points in a branch have precisely the same structure). Second, all subroutines adding points to branches had to be modified at the lines where the concatenation takes place.

Given a branch with bifurcations, one also wants to extract all points flagged as a particular bifurcation. This is possible using the new function `br_getflags`:

`FPI = br_getflags(branch)`
: This function extracts a list of flagged point indices out of `branch`. `FPI(1,:)` holds the indices of all points marked as `hopf`, `FPI(2,:)` those of `fold`, etc.

To streamline this process, we set up an indexing scheme to uniquely identify a type of bifurcation by a number. Two new functions are available to simplify this process.

`num = bif2num(bifstring)`
: Converts the bifurcation type contained in `bifstring` to its bifurcation index.

`bifstring = num2bif(num)`
: Converts the bifurcation index `num` to its bifurcation type.

So, for instance, we have `num2bif(1) = 'hopf'` and `bif2num('fold') = 2`. See Table 5.4 for the full specification.

Index	Flag	Point type
0	`stst`	Steady State
1	`hopf`	Hopf
2	`fold`	Fold
3	`psol`	Periodic Solution
4	`hcli`	Homoclinic
5	`genh`	Generalized Hopf
6	`hoho`	Double Hopf
7	`zeho`	Zero-Hopf

TABLE 5.4: *Bifurcation type numbering scheme.*

5.5 Orbit simulation

Matlab has several standard functions to simulate DDEs: `dde23` handles standard DDEs (with constant delays) and `ddesd` handles DDEs with state-dependent delays (SDD DDEs). The format in which the DDE should be passed to these functions is compatible with the DDE-BIFTOOL format. However, while DDE-BIFTOOL is capable of handling general state-dependent delays, `ddesd` can only handle delays that are functions of t and $x(t)$. This limitation carries over into the new features.

To facilitate simulating orbits, three new functions have been written. One of them is a wrapper for the Matlab routines, one of them is a helper function for the case of SDD equations, and one allows for a quick simulation with only a point structure as input.

`sol = p_integ(point, t0, tf, doplot, history, h, plotpoints)`
Integrates the system with `point` as constant history from time `t0` to `tf`. The last four arguments are optional. If `doplot` is set to `1`, the result will be plotted. A non-constant history function handle can be given in `history`. If `h` is nonempty and `doplot` is `1`, the history function will be plotted on the interval $[-h, 0]$. The number of plotpoints to be used for the plot is set in `plotpoints`. For custom computations, one can use the solution structure `sol` provided by Matlab (see the `dde23` documentation).

`sol = px_integ(x, par, t0, tf, doplot, history, h, plotpoints)`
Integrates the system at parameter value `par` with constant history given by `x` from time `t0` to `tf`. The last four arguments are optional; see `p_integ`.

`tauvec = px_integ_sddhelp(t,y,par,r)`
If an SDD equation is used, this function computes the vector of delays as function of the time and the state variables. `r` is the number of delays. *Only works if the delays are dependent on t and $x(t)$ only.*

Chapter 6

Example systems

6.1 The Ikeda equation

We first analyze a simple DDE, providing all system definition files and Matlab commands necessary to get the bifurcation results.

The DDE was inspired by the field of nonlinear optics and laser physics (see [7] for more information). It models the light intensity of a laser with a delayed optical feedback and it was formulated in 1979 by Kazuhiro Ikeda. The exact DDE reads

$$\varepsilon \dot{y} = -y + \lambda [1 - \sin(y(t-1))],$$

where $\varepsilon \equiv \tau / T$ is the ratio of the linear decay time and the delay T. Expanding the bifurcation parameter as $\lambda = \varepsilon^{-2}\Lambda$ leads to the so-called leading order problem

$$\dot{y} = -\frac{1}{2}\pi + \frac{1}{2}\Lambda y(t-1)^2 \quad \equiv f(y(t), y(t-1), \Lambda, \tau),$$

We see that the relevant quantities are the values $\vec{y} = (y(t), y(t-1)) = (y_1, y_2)$ and the parameters $(\Lambda, \tau) = (p_1, p_2)$.

Our aim is to reproduce Figure 2 in [7], to compute the First Lyapunov Coefficient of the Hopf bifurcation, and to verify its sign by simulation.

As input for `sys_deri`, we need a lot of derivatives of f. We will group them by order:

	1	2
y	$\frac{\partial f}{\partial y_1} = 0$	$\frac{\partial f}{\partial y_2} = \Lambda y_2$
p	$\frac{\partial f}{\partial p_1} = \frac{1}{2} y_2^2$	$\frac{\partial f}{\partial p_2} = 0.$

$y \backslash ^{p,y}$	1	2	1	2
1	$\frac{\partial^2 f}{\partial y_1 \partial \Lambda} = 0$	$\frac{\partial^2 f}{\partial y_1 \partial \tau} = 0$	$\frac{\partial^2 f}{\partial y_1^2} = 0$	$\frac{\partial^2 f}{\partial y_1 \partial y_2} = 0$
2	$\frac{\partial^2 f}{\partial y_2 \partial \Lambda} = y_2$	$\frac{\partial^2 f}{\partial y_2 \partial \tau} = 0$	$\frac{\partial^2 f}{\partial y_1 \partial y_2} = 0$	$\frac{\partial^2 f}{\partial y_2^2} = \Lambda$

Below, we list the system definition files. Note that the `sys_deri` and `sys_mfderi` files have been generated by the Maple script.

sys_init.m

```
function [name,dim]=sys_init()

name='ikeda';
dim=1;
path(path,'../../ddebiftool/');

return;
```

sys_tau.m

```
function tau=sys_tau()

% par:  Lambda tau

tau=[2];

return;
```

sys_rhs.m

```
function f=sys_rhs(y,par)

% y: y(t), y(t-1)
% par: Lambda, tau

f = -pi/2 + par(1)/2*y(2)^2;

return;
```

sys_deri.m

```
function J = sys_deri(xx,par,nx,np,v)

J = [];

if length(nx) == 1 && isempty(np) && isempty(v)
        switch nx
        case 0
                J = [0];
        case 1
                J = [par(1)*xx(1,2)];
        end
elseif isempty(nx) && length(np) == 1 && isempty(v)
        switch np
        case 1
                J = [1/2*xx(1,2)^2];
        case 2
                J = [0];
        end
elseif length(nx) == 1 && length(np) == 1 && isempty(v)
        switch nx
        case 0
                switch np
                case 1
                        J = [0];
                case 2
                        J = [0];
                end
        case 1
                switch np
                case 1
                        J = [xx(1,2)];
                case 2
                        J = [0];
                end
        end
elseif length(nx) == 2 && isempty(np) && ~isempty(v)
        nx1 = nx(1); nx2 = nx(2);
        switch nx1
        case 0
                switch nx2
```

```
                case 0
                        J = [0];
                case 1
                        J = [0];
                end
        case 1
                switch nx2
                case 0
                        J = [0];
                case 1
                        J = [par(1)*v(1)];
                end
        end
end

if isempty(J)
        display([nx np size(v)]);
        error('SYS_DERI: requested derivative could not be computed!');
end
```

sys_mfderi.m

```
function y = sys_mfderi(xx,par,varargin)

if nargin == 2
        error('SYS_MFDERI: no arguments.');
elseif nargin > 7
        error('SYS_MFDERI: too many arguments.');
end

y = 0;

numarg = nargin - 2;

switch numarg
        case 1
                u1 = varargin{1};
                y = [par(1)*xx(1,2)*u1(1,2)];
        case 2
                u1 = varargin{1}; u2 = varargin{2};
                y = [par(1)*u1(1,2)*u2(1,2)];
   otherwise
      y = 0;
end
```

We start by generating the upper branch. In order to do this, we construct a steady state point.

```
stst.kind='stst';
stst.parameter=[1 1];
stst.x=2;
stst.flag = '';
method=df_mthod('stst');
[stst,success]=p_correc(stst,[],[],method.point);
```

We now generate a branch with free parameter Λ (which has index 1) and set the limits for continuation. We set `stepsize = 0.05`.

```
branch2=df_brnch(1,'stst');
branch2.parameter.min_bound=[1 0.5];
branch2.parameter.max_bound=[1 2];
branch2.parameter.max_step=[1 stepsize];
```

We add our first point to the branch and add a second point that is slightly perturbed in Λ.

```
branch2.point=stst;
stst.parameter(1)=stst.parameter(1)+0.1;
[stst,success]=p_correc(stst,[],[],method.point);
branch2.point(2)=stst;
```

We continue the branch, turning on bifurcation detection, but turning off plotting for now.

```
branch2.method.continuation.plot = 0;
[branch2,s,f,r]=br_contn(branch2,100,1);
branch2=br_rvers(branch2);
[branch2,s,f,r]=br_contn(branch2,100,1);
```

This branch does not contain a bifurcation and DDE-BIFTOOL correctly discovers this fact: the output for the upper branch is

```
BR_CONTN warning: boundary hit.
BR_CONTN: no bifurcations detected.
P_CORREC warning: use of nonsquare Jacobian.
P_CORREC warning: use of nonsquare Jacobian.
P_CORREC warning: use of nonsquare Jacobian.
DETECT_BIF: There was no bifurcation found near par(1) = 0.9040.
BR_CONTN warning: boundary hit.
BR_CONTN: no bifurcations detected.
```

The routine `br_bifdet` notifies us of a false positive: there was a sign change, but the `p_correc` function was not able to find a Hopf point within the desired tolerance (here the default tolerance).

We repeat the above steps to generate the lower branch of steady states; we only need to change the line

```
stst.x=-2;
```

This produces the following output:

```
BR_CONTN warning: boundary hit.
BR_CONTN: no bifurcations detected.
P_CORREC warning: use of nonsquare Jacobian.
P_CORREC warning: use of nonsquare Jacobian.
DETECT_BIF: hopf point found at par(1) = 0.7854.
DETECT_BIF: l1 = -0.0591623057, omega = 1.5707963268, par(1) = 0.7854383082.
BR_CONTN warning: boundary hit.
BR_CONTN: 1 bifurcation(s) detected.
```

We see that one Hopf bifurcation is detected at $\Lambda \approx 0.7854$, with First Lyapunov Coefficient $l_1 \approx -0.0592$.

There is now one point flagged as `hopf`, which happens to be point 25. We extract this number automatically using `br_getflats` and we use it to get a periodic solution:

```
FPI = br_getflags(branch1);
hopfcand = branch1.point(FPI(bif2num('hopf'),1));
[hopf,success]=p_correc(hopf,1,[],method.point);
intervals = 18;
degree = 3;
[psol, stepcond] = p_topsol(hopf,1e-2,degree,intervals);
method=df_mthod('psol');
[psol, success] = p_correc(psol,1,stepcond,method.point);
```

We create and continue a branch of periodic solutions, using a degenerate periodic solution with amplitude zero as second point. We turn bifurcation detection off (as there is no detection for cycles):

```
branch3 = df_brnch(1,'psol');
branch3.parameter.min_bound=[1 0.5];
branch3.parameter.max_bound=[1 2];
branch3.parameter.max_step=[1 0.05];
deg_psol = p_topsol(hopf,0,degree,intervals);
deg_psol.mesh=[];
branch3.point = deg_psol;
psol.mesh = [];
branch3.point(2)=psol;
branch3.method.bifurcation.detect = 0;
[branch3,s,f,r] = br_contn(branch3,50);
```

We now plot the steady state branches, specifying that points flagged as `hopf` should be marked with a circle.

```
figure;
[xm,ym] = df_measr(0,branch1,'stst');
br_plot(branch1, xm, ym, '--', 'hopf','o');
[xm,ym] = df_measr(0,branch2,'stst');
br_plot(branch2, xm, ym, '--', 'hopf','o');
```

We also plot the minima and maxima of the periodic branch (in the same figure) and the period (in a new figure).

```
[xm,ym] = df_measr(0,branch3,'psol');
ym.col='max';
br_plot(branch3,xm,ym);
ym.col = 'min';
br_plot(branch3,xm,ym);
figure;
ym.field = 'period';
br_plot(branch3,xm,ym);
```

The resulting plots are shown in Figure 6.1. As you can see, they are identical to the ones shown in [7].

To verify the sign of the computed First Lyapunov Coefficient, we have to run actual simulations of the DDE, i.e. time integrations. As was explained in Section 5.5, this can now be done through DDE-BIFTOOL. It can be done in two ways: starting from a specific `point` structure, or starting from a given point in the state space. We are interested in how points that are *not* on our branches evolve, so we choose the second option.

For each simulation, we chose a certain parameter value Λ (which remained fixed during the simulation) and a certain initial value y_0. In Figure 6.1a, we have marked these initial values. They are treated as a constant history. (In general, the specific history chosen doesn't impact the long-term behavior of the solution, so we just use a constant history for simplicity.)

The code below results in Figure 6.2.

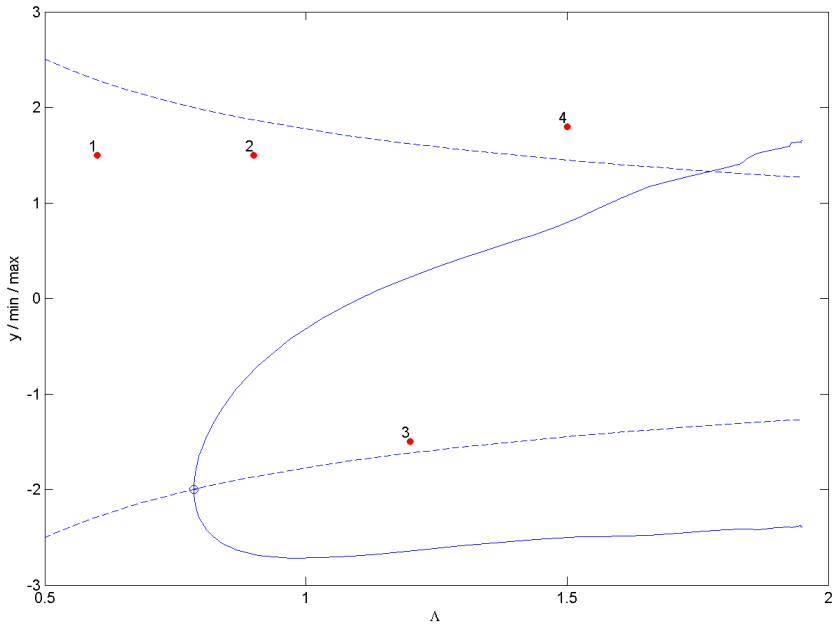
```
plist = [0.6, 0.9, 1.2, 1.5];
xlist = [1.5, 1.5, -1.5, 1.8];
tlist = [50, 50, 50, 5];
hlist = [10, 10, 10, 1];
mlist = [500, 500, 500, 500];
```

```
no = length(plist);
start = [];
for i = 1:no
   h = figure;
   set(gca,'LooseInset',get(gca,'TightInset'))
   mypar = plist(i);
   myx = xlist(i);
   px_integ(myx, [mypar, 1], 0, tlist(i), 1, start, hlist(i), mlist(i));
   xlabel('t');
   ylabel('y');
   title(sprintf('dde23 solution for \\Lambda = %g starting at y = %g', mypar, myx));
end
```

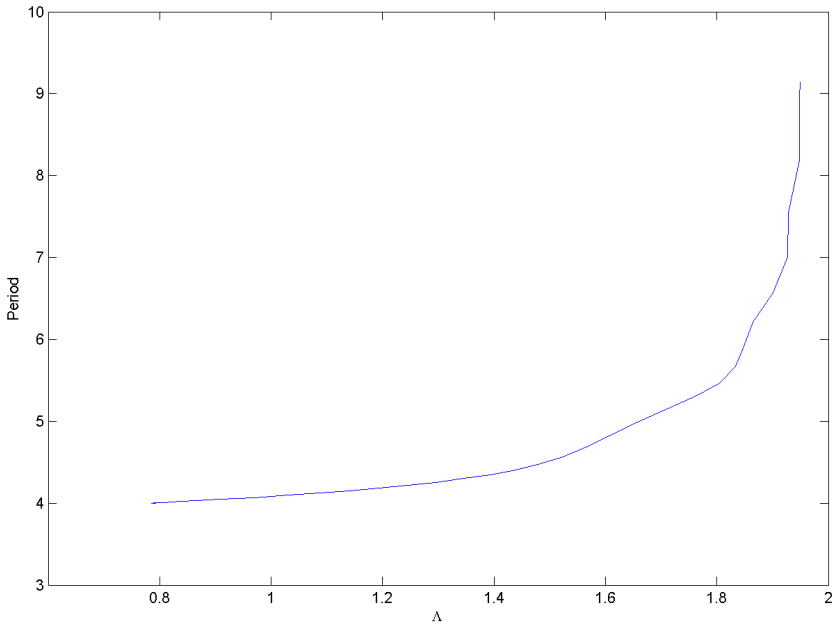
Some further remarks: for each plot, we have additionally selected an integration interval (the endpoint being contained in `tlist`), a history parameter h (in `hlist`) and a discretization number for the time axis (in `mlist`). Only the last plot needs different integration parameters, because the orbit quickly explodes.

From the plots, we may conclude that the bifurcation is supercritical. Before the bifurcation, at $\Lambda = 0.6$ (Figure 6.2a), the orbit is clearly attracted to the lower steady state. After the bifurcation (Figure 6.2b), we see that an orbit starting outside of the periodic orbit is attracted to the periodic orbit. An orbit starting near the equilibrium but inside the periodic orbit (Figure 6.2c) is also attracted by the periodic orbit. Lastly, we see that a solution starting on the other side of the second equilibrium is repelled (Figure 6.2d).

This means that our numerically computed First Lyapunov Coefficient has the correct sign, i.e. negative.

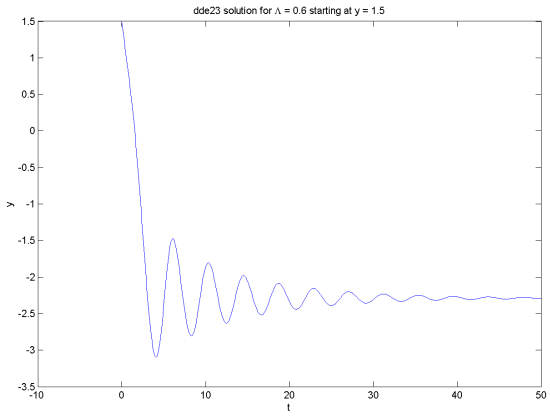


(A) *The steady state branches and the periodic branch. The steady state branches are plots of y versus Λ, the periodic branch shows the minima and maxima of the periodic solution versus Λ.*

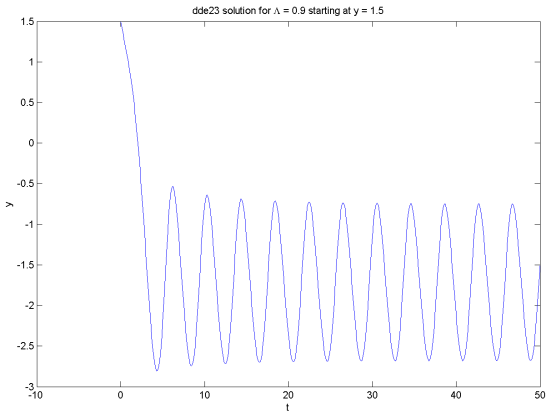


(B) *The periodic branch visualized in a period versus Λ plot.*

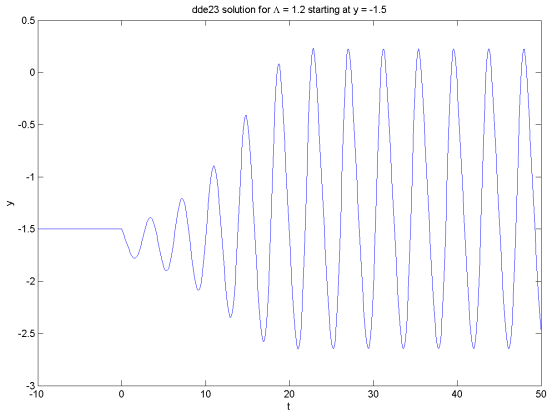
FIGURE 6.1: *Plots of the analysis of the Ikeda equation. Hopf bifurcations are labelled with a circle.*



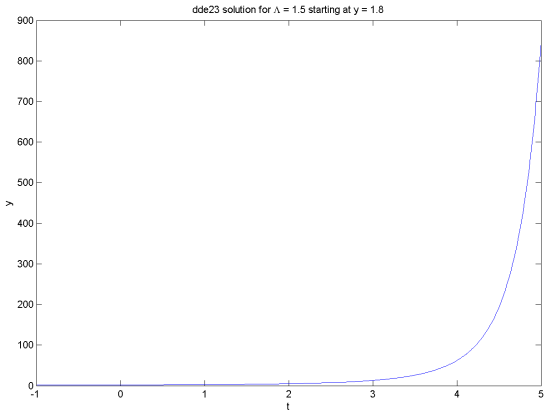
(A) *1:* $\Lambda = 0.6$, $y_0 = 1.5$



(B) *2:* $\Lambda = 0.9$, $y_0 = 1.5$



(C) *3:* $\Lambda = 1.2$, $y_0 = -1.5$



(D) *4:* $\Lambda = 1.5$, $y_0 = 1.8$

FIGURE 6.2: *Simulations of the Ikeda equation for different parameter values and initial values.*

6.2 A neural mass model

We look at a non-dimensionalized model of two interacting layers of neurons that was considered in [8] and [10]:

$$\begin{cases} \dot{x}_1(t) = -x_1(t) - ag(bx_1(t-\tau_1)) + cg(dx_2(t-\tau_2)), \\ \dot{x}_2(t) = -x_2(t) - ag(bx_2(t-\tau_1)) + cg(dx_1(t-\tau_2)), \end{cases} \tag{6.1}$$

where $g : \mathbb{R} \to \mathbb{R}$ is of the sigmoidal form

$$g(z) = [\tanh(z-1) + \tanh(1)]\cosh(1)^2.$$

The variables $x_1(t)$ and $x_2(t)$ represent the population-averaged neural activity at time t in layers one and two, respectively. The parameter $a > 0$ is a measure of the strength of inhibitory feedback, while $c > 0$ measures the strength of the excitatory effect of one layer on the other. The parameters $b > 0$ and $d > 0$ are saturation rates and the delays $\tau_{1,2}$ represent time lags in the inhibitory feedback loop and excitatory inter-layer connection. Note that the system is symmetric with respect to interchanging the labels 1 and 2, so equilibria are necessarily of the form (x_0, x_0).

In accordance with [8] and [10] we fix the numerical values

$$b = 2.0, \quad d = 1.2, \quad \tau_1 = 12.7, \quad \tau_2 = 20.2$$

and consider the feedback strengths a and c as free control parameters.

The system was intensively studied numerically in [10]. However, no use of DDE-BIFTOOL was made. We have converted the system into the DDE-BIFTOOL format and our aim is to reproduce Figure 4.4a from [10].

We begin our routine in the standard way. We set up a starting steady state point and define a steady state branch:

```
% construct a first, approximate steady state point:
stst.kind = 'stst';
stst.parameter = [0.2, 2, 15/29, 1.2, 12.7, 20.2];
stst.x = [0; 0];

method = df_mthod('stst');
[stst,success] = p_correc(stst,[],[],method.point);

branch1 = df_brnch(1,'stst');
branch1.parameter.min_bound = [1 0];
branch1.parameter.max_bound = [1 0.55];
branch1.parameter.max_step = [1 0.0005];

branch1.point = stst;

stst.parameter(1) = stst.parameter(1) + 0.0005;
[stst,success] = p_correc(stst,[],[],method.point);
branch1.point(2) = stst;

branch1.method.continuation.plot = 0;
branch1.method.bifurcation.minimal_real_part = -0.03;
[branch1,s,f,r] = br_contn(branch1,200);
branch1 = br_rvers(branch1);
[branch1,s,f,r] = br_contn(branch1,200);
```

Two things are worth of note here: first, we choose a small step size in a. The reason for this is that there is a region where the equilibrium becomes unstable. Secondly, we set the minimal real part for the detection algorithm to -0.03, because lower bounds cause overflow.

Execution of the above code finds three Hopf bifurcations. There's also a false positive.

```
BR_BIFDET: hopf point found at par(1) = 0.2988605971.
BR_BIFDET: the detected hopf point does not fall within the branch.

BR_BIFDET: hopf point found at par(1) = 0.2988605971.
BR_BIFDET: L1 = -0.0054576675, omega = 0.2710841446, par(1) = 0.2988605971.

BR_BIFDET: hopf point found at par(1) = 0.3453112603.
BR_BIFDET: L1 = -0.0133009607, omega = 0.1826428346, par(1) = 0.3453112603.

BR_BIFDET: hopf point found at par(1) = 0.3676281680.
BR_BIFDET: L1 = -0.0028635604, omega = 0.7234938546, par(1) = 0.3676281680.
```

Apparently, all Hopf bifurcations are subcritical. We now use the standard facility to plot this branch, setting `hopf` points to `'o'`:

```
figure;
[xm,ym] = df_measr(0,branch1,1);
br_plot(branch1,xm,ym,'-','hopf','o');
xlabel('a');
ylabel('x_1');
```

This yields Figure 6.3. We see that the algorithm produces tiny deviations from 0, suggesting that the equilibria are unstable there.

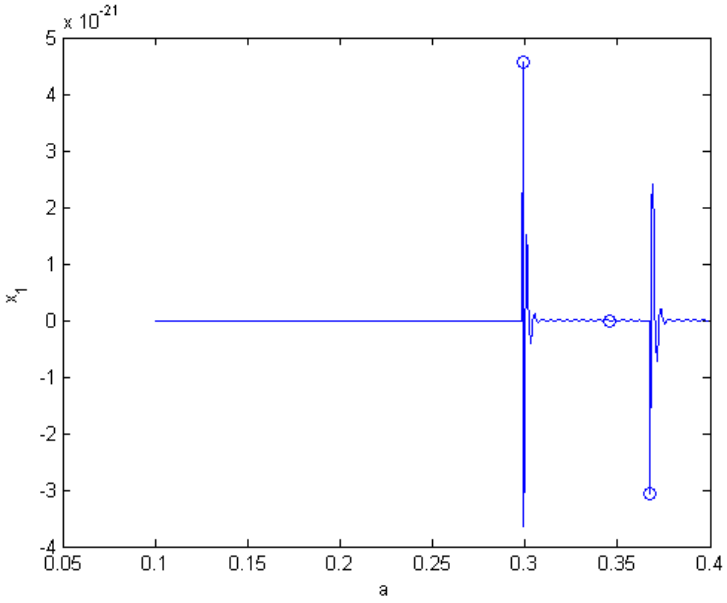


FIGURE 6.3: *A branch of steady states for system* (6.1) *in the free parameter a.*

As we want to reproduce Figure 4.4a from [10], we focus on the first two Hopf bifurcations. As before, we select them using the flagged point indices. We start a branch of Hopf points in the parameters a and c from the middle point:

```
FPI = br_getflags(branch1);

stst_hopf = branch1.point(FPI(bif2num('hopf'),2));
hopf = p_tohopf(stst_hopf);

method=df_mthod('hopf');
[hopf,success] = p_correc(hopf,1,[],method.point);

method.stability.minimal_real_part=-3;
```

```
hopf.stability=p_stabil(hopf,method.stability);
first_hopf = normal_form(hopf);

branch2 = df_brnch([1 3],'hopf');
branch2.parameter.min_bound=[5 0; 6 0; 3 0; 1 0];
branch2.parameter.max_bound=[1 0.55; 3 1];
branch2.parameter.max_step=[1 0.002; 3 0.005];

branch2.point=first_hopf;

hopf.parameter(3) = hopf.parameter(3)+0.005;
[hopf,success]=p_correc(hopf,1,[],method.point);
hopf = normal_form(hopf, first_hopf);
branch2.point(2)=hopf;

branch2.method.continuation.plot = 0;
branch2.method.bifurcation.minimal_real_part = -0.03;
branch2.method.stability.minimal_time_step = 0.005; % default 0.01
[branch2,s,f,r]=br_contn(branch2,300);

branch2 = br_rvers(branch2);
[branch2,s,f,r]=br_contn(branch2,300);
```

A few comments. First, we need to ensure that the `L1` field of the hopf points we create is actually set (because not all point manipulation routines do this automatically, to reduce overhead). Secondly, in the computation of `L1` of the second point of the branch, we use the null space information of first point, in order to get a smooth transition. Lastly, it proves necessary to change the minimal time step of the continuation.

We do the same for the second Hopf branch. The only difference is that we use `FPI(bif2num('hopf'),3)`.

Execution of the code above yields a lot of detections and a few false positives. The program outputs the parameter values of the bifurcation points, as well as the (most important) normal form coefficients. For the first Hopf branch:

```
BR_BIFDET: Unable to correct as hoho point near par(1) = 0.2793.
BR_BIFDET: Unable to correct as zeho point near par(1) = 0.2713.

BR_BIFDET: genh point found at par(1) = 0.2566151592, par(3) = 0.6210710570.
BR_BIFDET: L2 = 0.0018108418, omega = 0.1722173038, par(1) = 0.2566151592, par(3) = 0.6210710570.

BR_BIFDET: hoho point found at par(1) = 0.0903758971, par(3) = 0.7732080925.
BR_BIFDET: omega1 = 0.1560400861, omega2 = 0.2899790044, par(1) = 0.0903758971, par(3) = 0.7732080925.
theta(0) = 1.5262842865, delta(0) = 1.6991967926.

BR_BIFDET: Unable to correct as genh point near par(1) = 0.0028.

BR_BIFDET: zeho point found at par(1) = 0.0038000136, par(3) = 0.8396666836.
BR_BIFDET: omega = 0.1485575098, par(1) = 0.0038000136, par(3) = 0.8396666836.
s = 0.0000855577, theta = 2.0243160718.
```

For the second Hopf branch:

```
BR_BIFDET: genh point found at par(1) = 0.2518646976, par(3) = 0.5812003799.
BR_BIFDET: L2 = 0.0011032789, omega = 0.2759092846, par(1) = 0.2518646976, par(3) = 0.5812003799.

BR_BIFDET: Unable to correct as zeho point near par(1) = 0.2503.

BR_BIFDET: hoho point found at par(1) = 0.0903759166, par(3) = 0.7732080731.
BR_BIFDET: omega1 = 0.2899790029, omega2 = 0.1560400878, par(1) = 0.0903759166, par(3) = 0.7732080731.
theta(0) = 1.6991968331, delta(0) = 1.5262843306.

BR_BIFDET: Unable to correct as genh point near par(1) = 0.0109.

BR_BIFDET: zeho point found at par(1) = 0.0132891719, par(3) = 0.8554819555.
BR_BIFDET: omega = 0.2958045676, par(1) = 0.0132891719, par(3) = 0.8554819555.
s = 0.0000803557, theta = 2.0966106567.
```

As you can see, in the case of the Zero-Hopf and Hopf-Hopf bifurcations, only some key coefficients are displayed. We can get the entire set of coefficients by accessing the `nmfm` structure. For instance, those appearing in the (mostly) upper branch can be obtained in the following way:

```
FPI = br_getflags(branch3);
zeho = branch3.point(FPI(bif2num('zeho'),1));
hoho = branch3.point(FPI(bif2num('hoho'),1));
zeho.nmfm
hoho.nmfm
```

This yields:

```
 g200: -0.0064
 g110: -0.0134 + 0.0001i
 g011: -0.0125 - 0.0000i
 g300: -5.2284e-04
 g111: -0.0033 - 0.0000i
 g210: -0.0015 - 0.0000i
 g021: 0.0068 + 0.0082i
    b: -0.0064
    c: -0.0125 - 0.0000i
    d: -0.0134 - 0.0240i
    e: 0.0021
    s: 8.0356e-05 + 2.1254e-22i
theta: 2.0966

g2100: 0.0121 + 0.0026i
g1011: 0.0185 + 0.0058i
g1110: 0.0184 - 0.0079i
g0021: 0.0109 - 0.0031i
theta: 1.6992
delta: 1.5263
```

We now plot the branches (along with `branch1` for reference) with the codim 2 bifurcations marked.

```
figure;
[xm,ym] = df_measr(0,branch2,1);
br_plot(branch2,xm,ym,'-', 'genh', 'o', 'zeho', 'o', 'hoho', '*');
[xm,ym] = df_measr(0,branch3,1);
br_plot(branch3,xm,ym,'-', 'genh', 'o', 'zeho', 'o', 'hoho', '*');
br_plot(branch1,xm,ym,'--');
xlabel('a');
ylabel('c');
```

This yields Figure 6.4. This picture matches Figure 4.4a from [10] (save from the fact that in the latter, a rescaling of the parameters is used).

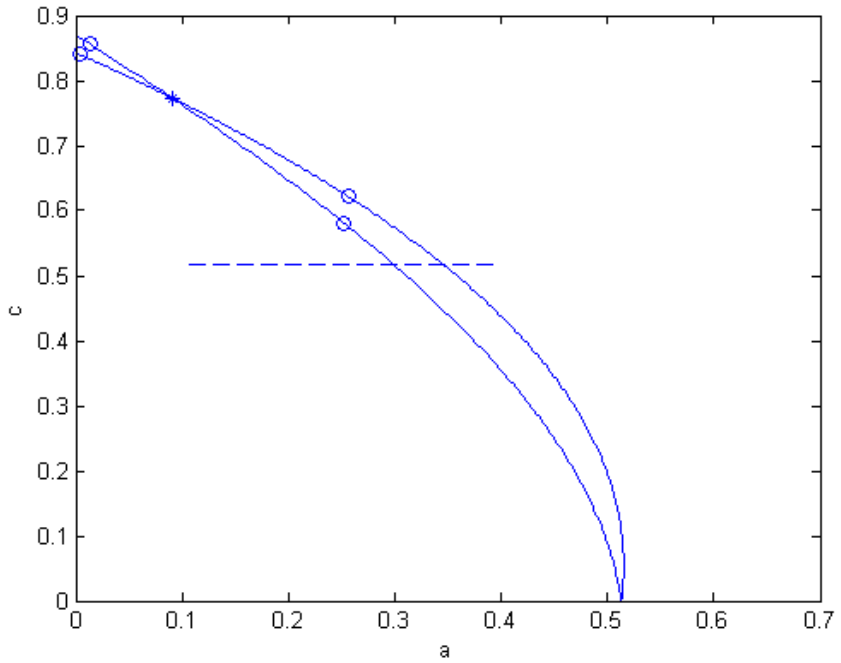


FIGURE 6.4: *The two Hopf branches in* (a, c) *for system* (6.1) *emanating from the two found Hopf bifurcations, with indicated codim 2 bifurcations. For reference, the equilibrium branch is drawn as well (– –).*

6.3 Two coupled neurons

Using our new functionality, we can further analyze one of the demo systems that comes packaged with DDE-BIFTOOL. It reads

$$\begin{cases} \dot{x}_1(t) = -\kappa x_1(t) = \beta \tanh(x_1(t-\tau_s)) + a_{12} \tanh(x_2(t-\tau_2)), \\ \dot{x}_2(t) = -\kappa x_2(t) = \beta \tanh(x_2(t-\tau_s)) + a_{21} \tanh(x_1(t-\tau_1)). \end{cases} \tag{6.2}$$

In the manual of DDE-BIFTOOL v2, this system is quite extensively analyzed. We repeat part of the analysis.

We set $\kappa = 0.5$, $\beta = -1$, $a_{12} = 1$, $a_{21} = 2.34$, $\tau_1 = \tau_2 = 0.2$ and $\tau_s = 1.5$. The origin is an equilibrium. We correct it (just to be sure) and add it to a new branch with free parameter a_{21}. We then perform the standard routine: we perturb the point a bit in the free parameter, correct it, and use the result as a second branch point. We then set some boundaries and continue the branch.

```
stst.kind='stst';
stst.parameter=[1/2 -1 1 2.34 0.2 0.2 1.5];
stst.x=[0 0]';

method=df_mthod('stst',1);
method.stability.minimal_real_part=-2;

[stst,success]=p_correc(stst,[],[],method.point);

branch1=df_brnch(4,'stst');
branch1.point=stst;

stst.parameter(4)=stst.parameter(4)+0.1;
[stst,success]=p_correc(stst,[],[],method.point);
branch1.point(2)=stst;

branch1.parameter.min_bound(4,:)=[4 0];
branch1.parameter.max_bound(1,:)=[4 5];
branch1.parameter.max_step(1,:)=[4 0.2];

branch1.method.continuation.plot=0;

[branch1,s,f,r]=br_contn(branch1,100);
branch1=br_rvers(branch1);
[branch1,s,f,r]=br_contn(branch1,100);
```

The continuation detects a Hopf bifurcation at $a_{21} \approx 0.8071$ with First Lyapunov Coefficient $L_1 \approx -0.0601$.

We store a list of flagged points indices in `FPI`. This allows us to quickly extract the Hopf point, eliminating the need to do this visually.

```
FPI = br_getflags(branch1);
hopfno = FPI(bif2num('hopf'),1);
hopf=p_tohopf(branch1.point(hopfno));
```

We now correct the Hopf point and use it to start a branch in the free parameters a_{21} and τ_s. We continue it without detection.

```
branch2=df_brnch([4 7],'hopf');
branch2.parameter.min_bound(4,:)=[4 0];
```

```
branch2.parameter.max_bound(1:2,:)=[[4 2]' [7 10]']';
branch2.parameter.max_step(1:2,:)=[[4 0.1]' [7 0.1]']';

method=df_mthod('hopf',1);
method.stability.minimal_real_part=-1;

[hopf,success]=p_correc(hopf,4,[],method.point);
first_hopf=hopf;

hopf.parameter(7)=hopf.parameter(7)+0.1;
[hopf,success]=p_correc(hopf,4,[],method.point);

branch2.point=first_hopf;
branch2.point(2)=hopf;

branch2.method.continuation.plot = 0;

[branch2,s,f,r]=br_contn(branch2,300);
branch2=br_rvers(branch2);
[branch2,s,f,r]=br_contn(branch2,50);
```

This code detects a Double Hopf bifurcation at $a_{21} \approx 0.2073$, $\tau_s \approx 8.6341$:

```
BR_BIFDET: omega1 = 0.3287154056, omega2 = 0.9157115847, par(4) = 0.2072962663, par(7) = 8.6340742491.
theta(0) = 292.5003948528, delta(0) = 0.0136751952.
[branch3,s,f,r]=br_contn(branch3,100);
```

We can now use this point to switch to the second Hopf branch. (Note that we do the initial correction in τ_s.)

```
branch3=df_brnch([4 7],'hopf');
branch3.parameter=branch2.parameter;
branch3.parameter.max_step(1:2,:)=[[4 0.1]' [7 0.03]']';

FPI = br_getflags(branch2);
pointno = FPI(bif2num('hoho'),1);
hopf = p_tohopf(branch2.point(pointno));
[hopf,success]=p_correc(hopf,7,[],method.point);
branch3.point=hopf;

hopf.parameter(4)=hopf.parameter(4)-0.05;
[hopf,success]=p_correc(hopf,7,[],method.point);
branch3.point(2)=hopf;

branch1.method.continuation.plot=0;
branch3.method.continuation.plot_progress=0;
branch3.method.bifurcation.detect = 0;
branch3.method.stability.minimal_time_step = 0.005; % default 0.01

[branch3,s,f,r]=br_contn(branch3,100);
branch3=br_rvers(branch3);
[branch3,s,f,r]=br_contn(branch3,100);
```

Plotting the resulting branches yields the bifurcation diagram in Figure 6.5.

During continuation of a Hopf branch, DDE-BIFTOOL computes the First Lyapunov Coefficient at each point. Plotting them yields the two pictures in Figure 6.6.

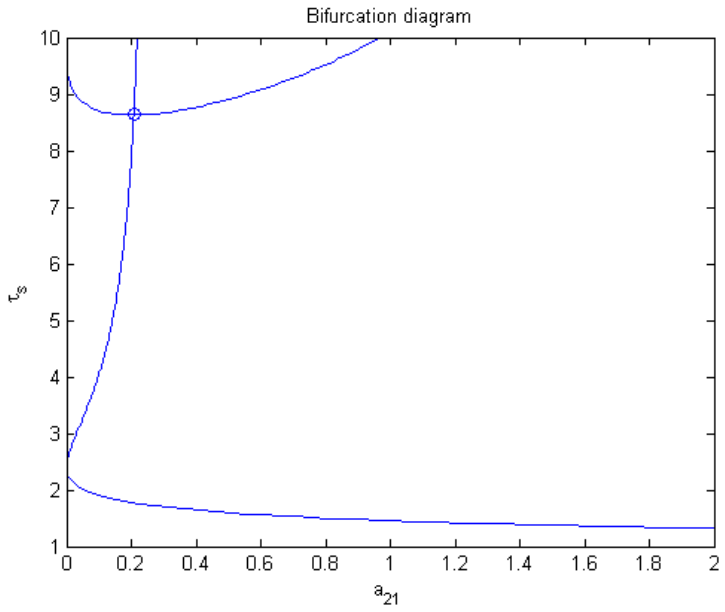
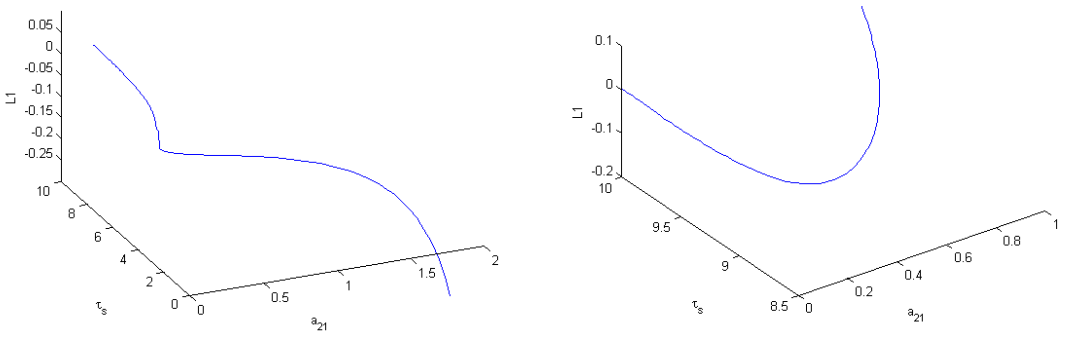


FIGURE 6.5: *The two Hopf branches of system* (6.2) *in the* (a_{21}, τ_s)*-plane.*



(A) *Lower branch* (B) *Upper branch*

FIGURE 6.6: *First Lyapunov Coefficient along the two Hopf branches.*

6.4 A neural field model

As a final example, we look at a neural field model with transmission delays which was studied in [14]. We aim to find a Hopf bifurcation, compute its First Lyapunov coefficient, and check whether it matches the simulations. The novelty of this example lies in the higher number of variables and the large number of delays.

The model treats a neural network as a continuum in which individual spikes are replaced by a spiking rate and space is continuous. However, in order to study it numerically, the equation needs to be discretized again. The non-discretized equation reads

$$\frac{\partial V}{\partial t}(t,\mathbf{r}) = -\alpha V(t,\mathbf{r}) + \int_\Omega J(\mathbf{r},\mathbf{r}')\, S(V(t-\tau(\mathbf{r},\mathbf{r}'),\mathbf{r}')\, d\mathbf{r}',$$

where $\Omega \subset \mathbb{R}^n$ is bounded, simply connected and open, $V(t,\mathbf{r})$ is the pre-synaptic membrane potential at time t of neurons at position $\mathbf{r} \in \Omega$, and $\alpha > 0$ is an exponential decay rate. The propagation delay $\tau(\mathbf{r},\mathbf{r}') \in C(\overline{\Omega}\times\overline{\Omega})$ measures the time it takes for a signal sent by a neuron at point $\mathbf{r}'$ to reach a neuron at point $\mathbf{r}$. In a similar fashion, $J(\mathbf{r},\mathbf{r}')$ represents the strength of this connection. Lastly, the function $S \in C^\infty(\mathbb{R})$ is the firing rate function.

We choose the spatial domain $\Omega = [-1,1]$ which can now be discretized into m subintervals of equal length $h = \frac{2}{m}$. This leads to a system of equations with $m+1$ discrete delays and $m+1$ variables:

$$\frac{\partial V_i}{\partial t}(t) = -\alpha V_i(t) + h\sum_{j=1}^{m+1} a_j \hat{J}(|i-j|h)\, S(V_j(t-\tau_0-|i-j|h))\ (i=1,\dots,m+1), \quad (6.3)$$

with

$$a_j = \begin{cases} \frac{1}{2} & \text{if } j \in \{1, m+1\}, \\ 1 & \text{otherwise} \end{cases}$$

and

$$\hat{J}(x) = A_1 e^{-\xi_1 x} + A_2 e^{-\xi_2 x},$$

where $A_{1,2}$ and $\xi_{1,2}$ are positive parameters. The firing rate function S is given by an odd sigmoid with steepness $r > 0$:

$$S(V) = \frac{1}{1+e^{-rV}} - \frac{1}{2}.$$

In order to be able to feed this system into DDE-BIFTOOL, we slightly rewrite the equation into a vector form. This yields

$$\frac{\partial V}{\partial t}(t) = -\alpha V(t) + h\sum_{j=1}^{m+1} \hat{J}(jh)(E_u^j + E_d^j)E_0\, S(V(t-\tau_j)),$$

where E_0, E_u and E_d are $(m+1)\times(m+1)$ matrices given by

$$E_0 = \begin{pmatrix} \frac{1}{2} & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & \frac{1}{2} \end{pmatrix}, \quad E_u = \begin{pmatrix} 0 & 1 & & & \\ & \ddots & \ddots & & \\ & & & \ddots & 1 \\ & & & & 0 \end{pmatrix}, \quad E_d = (E_u)^T.$$

In this context, the firing rate function produces a vector:

$$S(V) = \begin{pmatrix} \dfrac{1}{1+e^{-rV_1}} - \dfrac{1}{2} \\ \vdots \\ \dfrac{1}{1+e^{-rV_n}} - \dfrac{1}{2} \end{pmatrix}.$$

The delays are given by

$$\tau_j = a + (j-1)h \quad (j = 1, \ldots, m+1),$$

where a is a positive parameter.[1]

To summarize, given a discretization number m, we have $m+1$ variables, $m+1$ delays and 7 non-delay parameters: $\alpha, r, A_1, A_2, \xi_1, \xi_2, a$.

We fix $\alpha = 1, A_1 = 12, A_2 = -10, \xi_1 = 3, \xi_2 = 1$ and $a = 0.31$, and use r as a continuation parameter with initial value $r = 2.6$. In the standard way, we set up a `stst` branch and continue the trivial equilibrium in r. Note that we also have to set the delays and that we treat `m` as a global variable, so that other files (such as `sys_tau`) can automatically adapt.

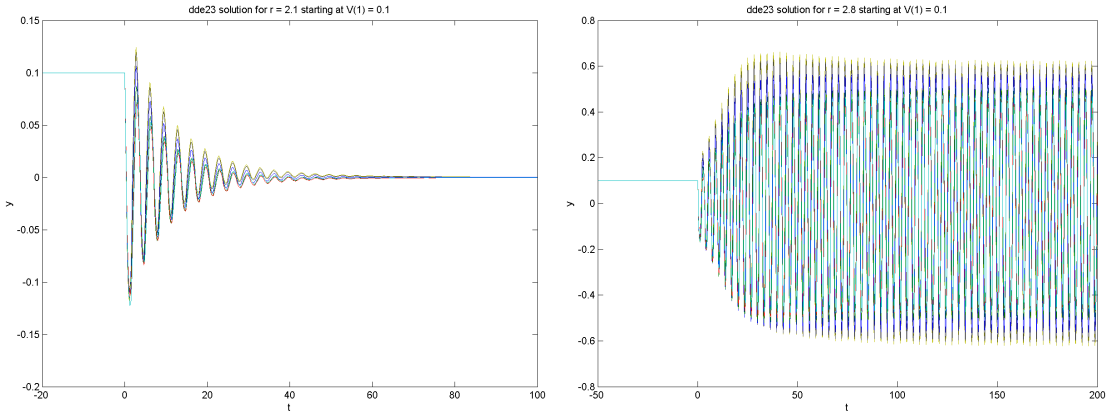
```
global m;
m = 10;
h  = 2/m;
lags = zeros(1,m+1);
for i = 1:m+1
   lags(i) = 0.31 + (i-1)*h;
end

stst.kind='stst';
stst.parameter=[1 2.6 12 -10 3 1 0.31 lags];
stst.x=[0 0]';

method = df_mthod('stst',1);
method.stability.minimal_real_part = -1;
[stst,success] = p_correc(stst,[],[],method.point);

stst_branch = df_brnch(2,'stst');

stst_branch.parameter.min_bound=[2 2];
stst_branch.parameter.max_bound=[2 3];
stst_branch.parameter.max_step=[2 0.01];

stst_branch.point = stst;

stst.parameter(2)=stst.parameter(2) - 0.01;
[stst,success] = p_correc(stst,2,[],method.point);

stst_branch.point(2)=stst;

stst_branch.method.continuation.plot = 0;
stst_branch.method.stability.minimal_real_part = -3;
stst_branch.method.bifurcation.minimal_real_part = -1;

stst_branch = br_contn(stst_branch,100);
stst_branch=br_rvers(stst_branch);
stst_branch = br_contn(stst_branch,100);
```

A Hopf bifurcation is found at $r \approx 2.35$, and it's First Lyapunov Coefficient is negative:

[1] In (6.3), this parameter is called τ_0. However, in DDE-BIFTOOL, the delays are numbered starting from 1. We will keep denoting the value of the first delay by a to avoid confusion of the type "`tau(1)` $= \tau_0$".

```
BR_BIFDET: hopf point found at par(2) = 2.3473099182.
BR_BIFDET: L1 = -0.0196309567, omega = 1.8492260135, par(2) = 2.3473099182.
```

We run some simulations using `dde23` to get the results in Figure 6.7. We see that a periodic solution emerges from the Hopf bifurcation to the right, just as the sign of the First Lyapunov Coefficient predicts!



(A) $r = 2.1$ (B) $r = 2.8$

FIGURE 6.7: *Simulations left and right of the Hopf bifurcation, starting from the constant history* $x = \frac{1}{10}(1,\ldots,1)^T$. *The different colored lines represent the* 11 *co-ordinates of the state vector.*

There are two logical next steps now: increasing m and/or continuing a Hopf branch in the parameter a. However, both come with some problems.

For small m (< 10), the necessary derivative files can be generated by the standard Maple script. However, for larger m, the computations become very time-consuming. On first thought, an alternative could be to use the default files that approximate derivatives by a finite elements method. However, for large m, even this approach makes the computations within DDE-BIFTOOL very slow, because of the repeated calls to `sys_rhs`. Furthermore, an approximate multilinear form file is not yet available. Therefore, we refrain from analyzing the system for higher m.

We would also like to be able to continue the Hopf point in the parameter a. However, the $m+1$ delays all depend on the single parameter a. Therefore, it would be necessary to treat the system as a state-dependent delay (SSD) equation. In principle, this is possible, but we have no guarantee that the expressions for the normal form coefficients will still be correct, as the theory for SDD equations is far more complex than for the standard DDEs. Nevertheless, we have tried the procedure, and the resulting picture can be found in Figure 6.8. During continuation, three Zero Hopf points are found.

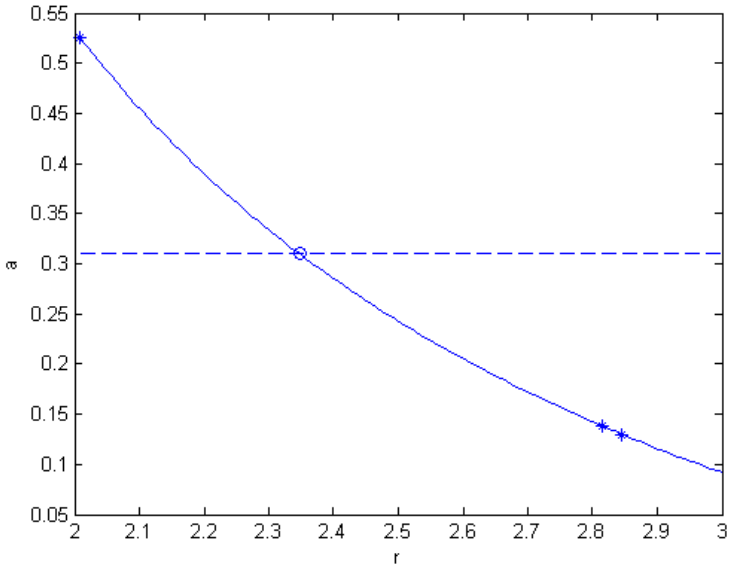


FIGURE 6.8: *The Hopf branch of the neural field system treated as an SDD equation in the* (r, a)*-plane. The Hopf bifurcation of the steady state branch is denoted by a circle; the stars are Zero Hopf detections on the Hopf branch.*

Chapter 7

Conclusion

We have seen that the use of sun-star calculus allows us to obtain expressions for the critical normal form coefficients of bifurcations in DDEs. In the case of discrete delays, we have shown that the numerical framework used by DDE-BIFTOOL is compatible with these expressions.

Given a system of DDEs featuring discrete delays, it is now possible to use DDE-BIFTOOL to detect the Hopf, Generalized Hopf, Zero-Hopf and Double Hopf bifurcations, to compute their normal form coefficients and to plot the bifurcation points easily in a branch. In addition, a Maple script is available to generate most necessary Matlab files from scratch, starting with the DDE only. Hopefully, these features can help researchers who use DDEs in applications.

There is still a lot of room for improvement, of course. Though not strictly necessary, it would be nice to have fold and cusp detection and normal form computation as well. The lack of these in the case of bifurcations that have to do with cycles and homoclinic orbits is more pressing. Perhaps it will one day be possible to numerically analyze DDEs with continuous delays as well. Lastly, there is still no interactive GUI for DDE-BIFTOOL. However, feeling like I have done my share, I happily leave these challenges for someone else to face.

Appendix A

List of DDE-BIFTOOL changes

In this appendix, we list the changes made to the original DDE-BIFTOOL v2.01. We used Matlab's internal change comparison tool. For the new files, we refer to the section in the main thesis where its functionality is described.

A.1 New files

bif2num.m

See Section 5.4.

br_bifdet.m

See Section 5.1.

br_flag.m

See Section 5.4.

br_getflags.m

See Section 5.4.

nmfm_binv.m

See Section 5.2.4.

nmfm_border.m

See Section 5.2.6.

nmfm_charmat.m

See Section 5.2.4.

nmfm_charmatderi.m

See Section 5.2.4.

nmfm_charmatderi2.m

See Section 5.2.4.

nmfm_genh.m

See Section 5.2.4.

nmfm_handletomatrix.m

See Section 5.2.4.

nmfm_hoho.m

See Section 5.2.4.

nmfm_hopf.m

See Section 5.2.4.

nmfm_hopfdet.m

See Section 5.1.3.

nmfm_smlrp.m

See Section 5.1.3.

nmfm_smlrpip.m

See Section 5.1.3.

nmfm_zeho.m

See Section 5.2.4.

num2bif.m

See Section 5.4.

p_inprod.m

See Section 5.1.1.

p_integ.m

See Section 5.5.

p_togenh.m

See Section 5.1.2.

p_tohoho.m

See Section 5.1.2.

p_tozeho.m

See Section 5.1.2.

px_integ.m

See Section 5.5.

px_integ_sddhelp.m

See Section 5.5.

A.2 Changed files

biftool.m

Updated contributors.

br_contn.m

Lines	Change
All	Renamed variable `l` to `ll` (to avoid confusion with numeral `1`)
1	Extra output (list of flagged points)
35-58	Setup bifurcation detection
270-300	Change to prevent concatenation errors
302-312	Compute L_1 in case of a Hopf branch
314-355	Call of `br_bifdet`, registration of flagged points, plot bifurcation points
363-371	Report detections

br_plot.m

Lines	Change
1	Added variable input
22	Now accepts branches of length 1 as well
27-52	Calls `br_plot` iteratively to produce different markers for bifurcation points

df_method.m

Lines	Change
26-33	Added default values for `method.bifurcation`

p_axpy.m

Lines	Change
20-22	Initialize `flag` and `nmfm` properties
124-127	Initialize `nvec` and `nmfm.L1` properties for Hopf

p_correc.m

Lines	Change
53-61	Add `flag` and `nmfm` fields if not present
613-618	Compute and store normal form information in case of Hopf

p_tofold.m

Lines	Change
13-16	Set `flag` and `nmfm` fields

p_tohcli.m

Lines	Change
23-26	Set `flag` and `nmfm` fields

p_tofold.m

Lines	Change
13-16	Set `flag` and `nmfm` fields

p_tohopf.m

Lines	Change
1	Added `success` output
18-21	Set flag field and initialize `success`
57-60	If no pair of roots is found, return `success = 0` instead of error
65-82	Support for codimension 2 bifurcations
92-100	Ensure `nmfm` and `nvec` are present

p_topsol.m

Lines	Change
25-28	Set `flag` and `nmfm` fields

p_tostst.m

Lines	Change
26-29	Set `flag` and `nmfm` fields
54	Support for codimension 2 bifurcations

stst_stabil_nwt_corr.m

Lines	Change
20	Initialize `l1`
63-68	Issue warning if `l1` is empty

Bibliography

[1] D. Breda, TRACE-DDE, http://sole.dimi.uniud.it/ dimitri.breda/research/software/.

[2] D. Breda, S. Maset, and R. Vermiglio, TRACE-DDE*: a tool for robust analysis and characteristic equations for delay differential equations*, Topics in Time Delay Systems: Analysis, Algorithms, and Control (J.J. Loiseau, W. Michiels, S.-I. Niculescu, and R. Sipahi, eds.), Lecture Notes in Control and Information Sciences, vol. 388, Springer, New York, 2009, pp. 145–155.

[3] O. Diekmann, S. van Gils, S. M. Verduyn Lunel, and H.-O. Walther, *Delay equations, functional-, complex-, and nonlinear analysis*, Springer-Verlag, 1995.

[4] K. Engelborghs, T. Luzyanina, and G. Samaey, DDE-BIFTOOL *v. 2.00: A matlab package for bifurcation analysis of delay differential equations*, Tech. Report TW 330, Department of Computer Science, Katholieke Universiteit Leuven, 2001.

[5] B. Ermentrout, *X-windows phaseplane plus auto*, http://www.math.pitt.edu/ bard/xpp/xpp.html.

[6] ———, *Simulating, analyzing, and animating dynamical systems: A guide to* XPPAUT *for researchers and students*, 1 ed., Software, Environments and Tools, vol. 14, Society for Industrial and Applied Mathematics, 2002.

[7] T. Erneux, L. Larger, K. Green, and D. Roose, *Modelling nonlinear optics phenomena using delay differential equations*, Proceedings of the International Conference on Differential Equations (Hasselt, Belgium), 2004.

[8] S. A. van Gils, H. G. E. Meijer, and S. Visser, *Stability analysis of steady states in a neural mass model*, Tech. report, Department of Mathematics, University of Twente, 2010.

[9] W. Govaerts and J. D. Pryce, *Mixed block elimination for linear systems with wider borders*, J. Numer. Anal. **13** (1993), 161–180.

[10] S. G. Janssens, *On a normalization technique for codimension two bifurcations of equilibria of delay differential equations*, Master's thesis, Department of Mathematics, Utrecht University, 2010.

[11] Yu. A. Kuznetsov, *Elements of applied bifurcation theory*, Springer-Verlag, 2004.

[12] KU Leuven Scientific Computing Research Group, *Software for delay differential equations*, http://twr.cs.kuleuven.be/research/software/delay/software.shtml, June 2014.

[13] R. Szalai, *Knut: a continuation and bifurcation software for delay-differential equations (version 8)*, Department of Engineering Mathematics, University of Bristol, December 2013.

[14] S.A. van Gils, S.G. Janssens, Yu.A. Kuznetsov, and S. Visser, *On local bifurcations in neural field models with transmission delays*, Journal of mathematical biology **66** (2013), no. 4-5, 837–887.

Note: there is a vast literature on the subject on DDEs, ranging from purely theoretical works to the treatment of specific systems. This bibliography only includes those references that were actually cited in the text. For a more extensive list, see e.g. [3].