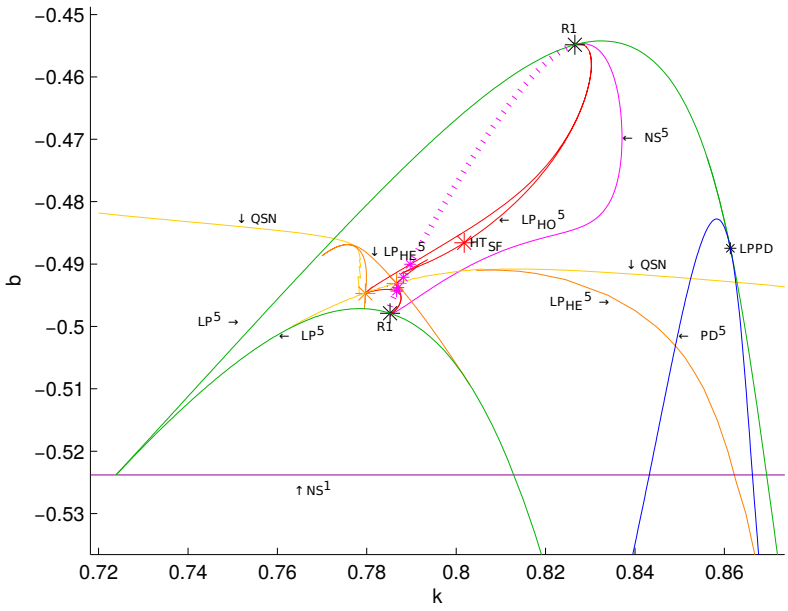
Advances in numerical bifurcation software: MatCont

Niels Neirynck



Promotor: Prof. M. Van Daele, Ghent University, Belgium
Promotor: Prof. H.G.E. Meijer, University of Twente, The Netherlands
Supervisor: Prof. W. Govaerts, Ghent University, Belgium

Thesis submitted to obtain the degree of
Doctor of Science: Computer Science

2018-2019

UNIVERSITEIT GENT

Department of Applied Mathematics, Computer Science and Statistics
Faculty of Science, Ghent University

Contents

Chapter 1

Introduction

The mathematical background of MatCont is bifurcation theory which is a field of hard analysis, see in particular [60]. Bifurcation theory treats dynamical systems from a high-level point of view. In the case of continuous dynamical systems this means that it considers nonlinear differential equations without any special form and without restrictions except for differentiability up to a sufficiently high order (in the present state of MatCont never higher than five.) The number of equations is not fixed in advance and neither is the number of variables or the number of parameters, some of which can be active and others not. The essential aim of bifurcation theory is to understand and classify the qualitative changes of the solutions to the differential equations under variation of the parameters. From the applications point of view this knowledge cannot be applied to practical situations without numerical software, except in some simple, usually artificially constructed situations.

A key ingredient of such numerical software packages is that of numerical continuation, whereby curves of objects of a given type (for example, equilibria, periodic orbits, Hopf bifurcation points, homoclinic orbits ...) are computed under variation of one or more system parameters, cf. [4, 41, 80].

The history of numerical software packages for dynamical systems, both continuous (ODEs) and discrete (maps) goes back to the 1980s. A survey of this history is given in Chapter 2 ("Interactive Continuation Tools", by W. Govaerts and Yu. A. Kuznetsov) in [58]. The first non-interactive packages appeared in the beginning of the 1980's and were written in Fortran. The most widely used packages of this generation are AUTO [26, 27] and LINBLF [53]. AUTO is still widely used because of its high speed in numerical computations; there are several environments which allow a more user-friendly approach to AUTO, the best known of which is XPPAUT [33]. LOCA[82] is another non-interactive package that is oriented towards relatively simple bifurcation problems in large-scale systems.

The first software environments for bifurcation analysis were DsTool [6] and CONTENT[61]. A recent newcomer is COCO ("Continuation Core") [16, 17] which is a MATLAB package with emphasis on numerical continuation, boundary value problems, theoretical rigor, algorithm development, and software engineering. One of its novel features is the vectorized form of the defining system for periodic orbits. For other packages

we refer to the survey in [58]. They have their own merits but at present MatCont has more functionalities related to bifurcation theory than any other package.

'MatCont' stands for 'MATLAB Continuation'. Its counterpart for discrete time systems generated by iterated maps is called 'MatContM'. Both packages can be used either from the command line or by using a GUI. The command line use is referred to as Cl_MatCont or Cl_MatContM, respectively. The GUI versions are more user-friendly and are probably used more often. The command line versions are more flexible and powerful but require more work and more insight in the underlying mathematics and numerical methods.

For advanced users the distinction between MatCont and Cl_MatCont tends to get blurred. Indeed, the ode-files (map-files in the case of maps) are best generated by using either the GUI or a standalone version of a part of it, cf. §3.6. The homotopy methods for orbits homoclinic to saddle or to saddle-node and to heteroclinic orbits require the use of the GUI. Also, there is a command line interface that allows to interact directly with the GUI from the MATLAB command line.

Both MatCont and MatContM are MATLAB [66] successor packages to CONTENT but were developed from scratch with many new functionalities. The project is lead jointly by W. Govaerts (Ghent University, Belgium) and Yu.A. Kuznetsov (Utrecht, The Netherlands) and more recently also by H.G.E. Meijer (University of Twente) who has been a long-time co-developer. The development of non-interactive MatCont started with the master theses of A. Riet [79] and W. Mestrom [69] at Utrecht University. The first GUI was built soon after by A. Dhooge (Ghent) and announced in [24]; O. De Feo provided general software support. Since that time this 2003 paper has been the main reference to MatCont in spite of the continuous development that followed. V. Govorukhin (Rostov, Russia) provided the high-order integration routines `ode78.m` and `ode87.m`. E. Doedel (Concordia University, Montreal, Canada) contributed to the continuation methods for bifurcations of periodic orbits [28, 29, 59]. B. Sautois (Ghent) introduced the computation of the phase response curve of a periodic orbit and its derivative [43]. This functionality is in particular useful in the study of synchronization of weakly coupled oscillators, e.g. in neural networks. V. De Witte (Ghent) [22] contributed to the initialization and continuation of homoclinic and heteroclinic orbits and introduced in [23] the computation of normal form coefficients of codimension 2 bifurcations of periodic orbits.

The first version of Cl_MatContM was written by R. Khoshsiar Ghaziani (Ghent and Shahrekord (Iran), 2008), cf. [55]; N. Neirynck (Ghent, 2012) added the GUI, cf. [71, 44]. MatContM provides the functionality to compute the normal form coefficients of bifurcations by automatic differentiation. The user can opt for this either for reasons of speed or because the MATLAB symbolic toolbox is not available. This functionality is largely due to J.D. Pryce (Cranfield University, UK)[75]. If symbolic differentiation is available, then numerical tests suggest that it is faster for low iteration numbers but not for high iteration numbers. Automatic differentiation was not introduced in MatCont since tests indicated that it was quite slow in that situation where no iterations are involved.

L. Vanhulle (Ghent) [92] contributed to the computation of homoclinic and heteroclinic connections, cf. §4.5. Among other people who were involved at some point we mention

M. Friedman (University of Alabama at Huntsville), E. Nikolaev (Jefferson University, Philadelphia) and P. Pareit (Ghent).

The software related to the MatCont project, including the manuals and tutorials, is freely available from www.sourceforge.net. The user should search for 'matcont' and then follow the 'readmefirst' and 'readme' pdf's.

We note that there exists a MatCont-inspired package Cl_MatContL that is dedicated to large equilibrium systems [9] but is not distributed with the regular MatCont and MatContM packages. It has no GUI, no normal form computations and only a small list of functionalities. For the code and references to the documentation we refer to [65]. Another related MATLAB package is DDE-biftool [31, 18] which deals with delay-differential equations.

A part of my contributions to the development of MatCont and MatContM was published in [3], [44], [50], [72], [73], but the present thesis contains a lot of unpublished work as well.

This thesis is structured as follows. In Chapter 2 (Preliminaries) we discuss general aspects of MatCont and mention some of the many application fields where MatCont is being used. We then briefly discuss the mathematical background of bifurcation theory for ODEs and for maps with survey tables of bifurcations and branch switchings. In §2.8 and §2.9 we give an overview of our own contributions to the development of the MatCont and MatContM software. Not all of this is further described in the present thesis; we focus on the aspects which are most useful to future users and developers.

In Chapter 3 (Numerical continuation: the algorithmic basis) we discuss the (numerical) algorithms which form the computational core of MatCont and MatContM. Numerical details are not given here since they can be found elsewhere; we focus on the aspects that have to be understood by users and developers.

In Chapter 4 (MatContM for maps) we discuss some of our own contributions to the MatContM software for maps and applications thereof. This involves Lyapunov exponents for maps, the growing of stable and unstable manifolds, the initialization of connecting orbits and the detection of codimension 1 and codimension 2 bifurcations in homoclinic connections. Section 4.5 on the intersection of a stable and an unstable manifold is new and unpublished.

Chapter 5 (Front end features of the new MatCont GUI) deals with the new MatCont GUI. It is user-oriented and describes the MatCont database and the **Data Browser** to access it. A survey of of the panels and their functionalities is also given.

Chapter 6 (Internal working of the new MatCont GUI) is developer-oriented. It describes the inner working of the MatCont software.

Chapter 7 (Future work) mentions some topics for further investigations with varying degrees of complexity.

Chapter 8 (Summary) contains summaries of the thesis in English and Dutch.

This thesis further contains the Appendices A-I. The first four A-D are tutorials to make the user familiar with the basic functionalities of the new version of **MatCont**. The Appendices E-G provide listings of the algorithms discussed in Chapter 4. The appendices

H-I provide the complete lists of settings and computations that are discussed in Chapter 6.

Chapter 2

Preliminaries

2.1 General aspects of MatCont

MatCont is a MATLAB software package for the computational (i.e. numerical) study of continuous dynamical systems, i.e. systems of ordinary differential equations with parameters, with the use of continuation and bifurcation methods. Numerical continuation is in principle a simple algorithm, at least in this context. It can be seen as the numerical pendant of homotopy theory or as the implicit function theorem in practice. On the other hand, bifurcation theory is a mathematically difficult subject and the translation to numerical methods is complex. It requires the reduction of a high-dimensional dynamical system to a problem in a low-dimensional (nonlinear) invariant center manifold, the numerical study on this manifold by the use of normal form theory and a transformation back to the high-dimensional space. All transformations are local and based on Taylor expansions.

The simplest nontrivial application of these methods is that of a Hopf bifurcation. Suppose one starts with an equilibrium state in a dynamical system which might be the set of stoichiometric equations in a continuously stirred tank reactor. This state is stable if it persists under small perturbations. However, it may loose stability if certain parameters are changed and then exhibit small amplitude periodic behaviour or else other scenarios are possible. In the case of small amplitude oscillations this is called a Hopf bifurcation. In such a case, MatCont can predict the loss of stability and compute the periodic orbits which appear. It can also predict whether or not the periodic orbits themselves will be stable or unstable and it can trace them for further changes in the parameters.

A Hopf bifurcation is an example of a codimension 1 bifurcation. In one-parameter problems the codim 1 bifurcation points typically divide the parameter line in regions (line segments) in which the behaviour of the dynamical system is qualitatively the same; more precisely, for all parameter values in a region the behaviours in the state space are topologically equivalent.

In two-parameter problems the parameter space is also typically divided into regions in which the behaviours are equivalent. These regions are separated by curves of codimension 1 bifurcation points which meet in codimension 2 bifurcation points.

In many application fields the qualitative dependence of dynamical systems on the values of parameters is crucial, see §2.2. This idea lies at the heart of both mathematical bifurcation theory and its implementation in numerical methods and software.

To deal with the complexity of dynamical systems MatCont supports several types of computations, in particular:

- Numerical continuation for 12 different curve types which include bifurcation curves, periodic orbits, homoclinic and heteroclinic orbits.
- Numerical integration with different integrators and the computation of Poincaré maps.
- Detection and location of bifurcation points of 23 bifurcation types if only equilibria and periodic orbits are counted (and many more if homoclinics and heteroclinics are included).
- A complicated network of initializer routines that links the previous types of computations.
- Computation of normal form coefficients for 21 bifurcation types which depend on partial derivatives up to fifth order.
- Computation of phase response curves of periodic orbits.
- Homotopy methods to initialize orbits homoclinic to saddle, to saddle-node and heteroclinic orbits.

Many of these routines are not available in any other software.

The computations can be done in the command line version Cl_MatCont of MatCont, for which there is an extensive user manual [45] and for an experienced user this is the most powerful and flexible way. However, for practical use by application-oriented users it is necessary to have a user-friendly GUI-version in which the basics can be learned from tutorials (the closest to this for Cl_MatCont is a collection of testruns that is provided in MatCont and discussed in the manual [45].)

2.2 Where MatCont is being used

Bifurcation theory has applications in many fields, in fact wherever phenomena are modeled by differential equations. Usually these applications need computational methods and software such as MatCont. Several books were written on the use of dynamical systems and bifurcation methods in specific application fields, e.g. [8, 30, 34, 87]. Among the thousands of papers we mention [37, 70, 88, 89, 76].

MatCont has been used as a tool for teaching courses in dynamical systems or mathematical modelling, but it is unclear how often and at what institutions or universities. It is easier to get an impression of its use for research purposes by considering its citations

in the core collection of the Web of Science. To be specific we note that on October 6, 2016 in the Web of Science core collection 462 papers cited the first paper [24] on MatCont. By September 22, 2018 this number increased to 617. The follow-up paper [25] was then cited 49 times. Nearly all citing papers deal with applications of bifurcation theory and they cover most fields of quantitative science. To give examples, we cite a number of publications that refer to MatCont:

- Steady states in coupled oscillators [5]
- Rayleigh-Bénard convection [15]
- Bacteria-phage interaction in a chemostat [94]
- Organic matter decomposition in a chemostat [49]
- Saccharomyces Serevisiae fermentation processes [83]
- Design of cell cycle oscillators [68]
- Use of transcriptomic data (Systems biology) [74]
- Control of rotating blade vibrations [51]
- Vehicle systems dynamics [20]
- Underactuated mechanical systems [47]
- Resonances of an accelerating beam [81]
- Electronic circuits [63]
- Population dynamics of Xenopus tadpole [10]
- Predator-prey models [77]
- Bottom fishing [14]
- Dynamics of landscapes [91]
- Neural models [42, 43]
- Pattern storage in neural networks [35]
- Small neural circuits [12]
- Jansen-Rit neural mass models [2]
- Metabolic Engineering (Bioinformatics) [84]
- Insulin secretion and hepatitis [99]

- Innate Immunity Responses of Sepsis [97]
- Chemical reaction engineering [78]
- Biochemistry [85, 86]
- Climate warming [62]
- Magnetic Resonance Force Microscopy [46]
- Harvesting piezoelectric vibration energy [98]
- Geophysics (the Lorenz-96 model) [52]
- Onset and dynamics of bicycle shimmy [90]
- Aeronautical engineering [93]

Other papers explicitly refer to MatCont but do not cite it, e.g.

- Infectious diseases [48]

The main other general purpose packages for continuation and bifurcation software PyDSTool, AUTO-07P, and COCO are also available on www.sourceforge.net. On August 6, 2018 the number of weekly downloads was recorded as 13 for PyDSTool, 43 for AUTO-07P, 6 for COCO and 390 for MatCont.

2.3 Bifurcation objects in ODEs

In Tables 2.1 and 2.2 we provide lists of the codimension 0, 1 and 2 objects that can be found in generic continuous dynamical systems. To each of them we attach a label based on standard terminology [60].

The relationships between these objects are complicated.

The detection relationships are presented in Figures 2.1 and 2.2. An arrow from O to EP or LC means that when we compute an orbit, it is generically possible that the orbit will converge to a (stable) equilibrium or to a (stable) limit cycle. An arrow from an object A different from O to an object B means that the continuation of a one-parameter family of objects of type A can generically lead to the detection of an object of type B, either because the B- object is a special case of the A- object or because it is a limit case when the parameter tends to a special value. An example of the first situation is a H point on a EP curve; an example of the second situation is a HHS point as the limit situation of an LC branch when the period tends to infinity. We do not distinguish between the two situations for two reasons. First, the difference depends somewhat on the careful definition of a family of objects. Second and related, in the implementations it may depend on the defining system that is used in the computation of the branch (e.g. a H point on a family of LC objects).

Table 2.1: Equilibrium- and cycle-related objects and their labels

Type of object	Label
Point	P
Orbit	O
Equilibrium	EP
Limit cycle	LC
Limit Point (fold) bifurcation	LP
Hopf bifurcation	H
Limit Point bifurcation of cycles	LPC
Neimark-Sacker (torus) bifurcation	NS
Period Doubling (flip) bifurcation	PD
Branch Point	BP
Cusp bifurcation	CP
Bogdanov-Takens bifurcation	BT
Zero-Hopf bifurcation	ZH
Double Hopf bifurcation	HH
Generalized Hopf (Bautin) bifurcation	GH
Branch Point of Cycles	BPC
Cusp bifurcation of Cycles	CPC
Generalized Period Doubling	GPD
Chenciner (generalized Neimark-Sacker) bifurcation	CH
1:1 Resonance	R1
1:2 Resonance	R2
1:3 Resonance	R3
1:4 Resonance	R4
Fold–Neimark-Sacker bifurcation	LPNS
Flip–Neimark-Sacker bifurcation	PDNS
Fold-flip	LPPD
Double Neimark-Sacker	NSNS

Table 2.2: Objects related to homoclinics to equilibria and their labels

Type of object	Label
Limit cycle	LC
Homoclinic to Hyperbolic Saddle	HHS
Homoclinic to Saddle-Node	HSN
Neutral saddle	NSS
Neutral saddle-focus	NSF
Neutral Bi-Focus	NFF
Shilnikov-Hopf	SH
Double Real Stable leading eigenvalue	DRS
Double Real Unstable leading eigenvalue	DRU
Neutrally-Divergent saddle-focus (Stable)	NDS
Neutrally-Divergent saddle-focus (Unstable)	NDU
Three Leading eigenvalues (Stable)	TLS
Three Leading eigenvalues (Unstable)	TLU
Orbit-Flip with respect to the Stable manifold	OFS
Orbit-Flip with respect to the Unstable manifold	OFU
Inclination-Flip with respect to the Stable manifold	IFS
Inclination-Flip with respect to the Unstable manifold	IFU
Non-Central Homoclinic to saddle-node	NCH

However, there are two exceptions of the same type. Namely, the arrows from EP to BP and from LC to BPC jump over two codimension levels. In fact, these situations are non-generic but they are so often found in systems with equivariance or invariant subspaces that most software packages support their detection.

Branching relationships are bottom-up. In general, if there is an arrow from an object A different from O to an object B, then for each object of type B there is a unique one-parameter family of objects of type A that branches off the B-object if a number of (1+codim A) free variables is chosen. However, there are four types of exceptions:

1. The arrows from EP to BP and from LC to BPC: in these cases there are generically two codimension 0 curves rooted at the codimension 2 points.

2. The arrows from H to HH and from NS to NSNS: in these cases there are generically two codimension 1 curves rooted at the codimension 2 points.

3. The arrow from NS to ZH. In this case, the existence of the NS curve rooted in the ZH point is subject to an inequality constraint.

4. The arrow from NS to HH. In this case there are generically two NS curves rooted in a HH point.

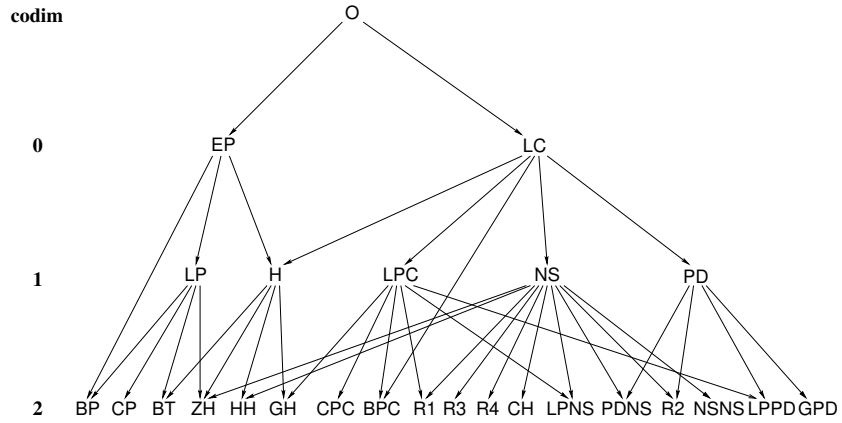


Figure 2.1: Relationships between equilibrium and limit cycle bifurcations.

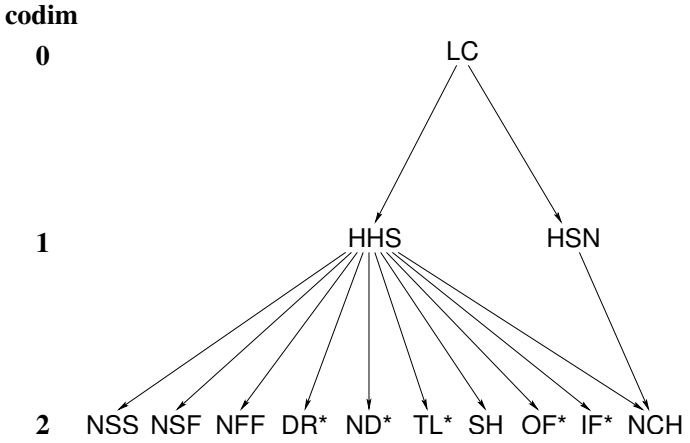


Figure 2.2: Relationships between homoclinic bifurcations; here * stands for S or U.

We note that it is generically also possible to start a curve of HHS orbits from a BT point (not indicated on Figures 2.1 and 2.2).

2.4 Bifurcation objects for cycles of maps

In Table 2.3 we provide a list of the codimension 0,1 and 2 objects that can be found in generic maps. To each of them we attach a label based on standard terminology [60].

The detection relationships between them are presented in Figure 2.3. The precise meaning of the arrows is simpler than in the case of ODEs: if we exclude FP then an arrow from an object A to an object B indicates that the B- object can generically be found as a regular point on a branch of A- objects. The only exception is the arrow from FP to BP which again is non-generic but found in many examples that exhibit a form of equivariance or have invariant subspaces.

Table 2.3: Dynamical objects for maps and their labels

Type of object	Label
Point	P
Fixed Point	FP
Limit Point of cycle bifurcation	LP
Period Doubling Point of cycles	PD
Neimark-Sacker bifurcation	NS
Branch Point	BP
Cusp bifurcation	CP
Generalized Period Doubling	GPD
Chenciner (generalized Neimark-Sacker) bifurcation	CH
1:1 Resonance	R1
1:2 Resonance	R2
1:3 Resonance	R3
1:4 Resonance	R4
Fold–Neimark-Sacker bifurcation	LPNS
Flip–Neimark-Sacker bifurcation	PDNS
Fold-flip	LPPD
Double Neimark-Sacker	NSNS

On the other hand, the branching diagram for maps is far more complicated than for ODEs; this is largely due to the fact that the iteration number is an additional issue to be taken into account. For reasons of clarity we therefore present two branching diagrams, namely Figure 2.4 and Figure 2.5. Here possible switches at codim-1 and codim-2 bifurcation points are indicated graphically. Several switches to branches of lower codimension curves with double, triple or quadruple iteration number are now possible, some of them depending on constraints.

2.5 Why programming in MATLAB?

MATLAB is an interpreted language and its speed of execution cannot compete with a compiled language such as FORTRAN, PYTHON or *C++*. But in many applications the speed of computing is less important than the human speed in programming or using the software. In fact more recent packages such as COCO [17] have also chosen MATLAB as a programming language.

A disadvantage is that MATLAB is known to often change its software, which can be quite inconvenient to programmers and users. This has also created problems for MATCONT, see §2.8. However, in most cases the new versions only affect specific toolboxes and do not change the core of MATLAB.

On the other hand, MATLAB is built upon extensive and well-tested numerical and

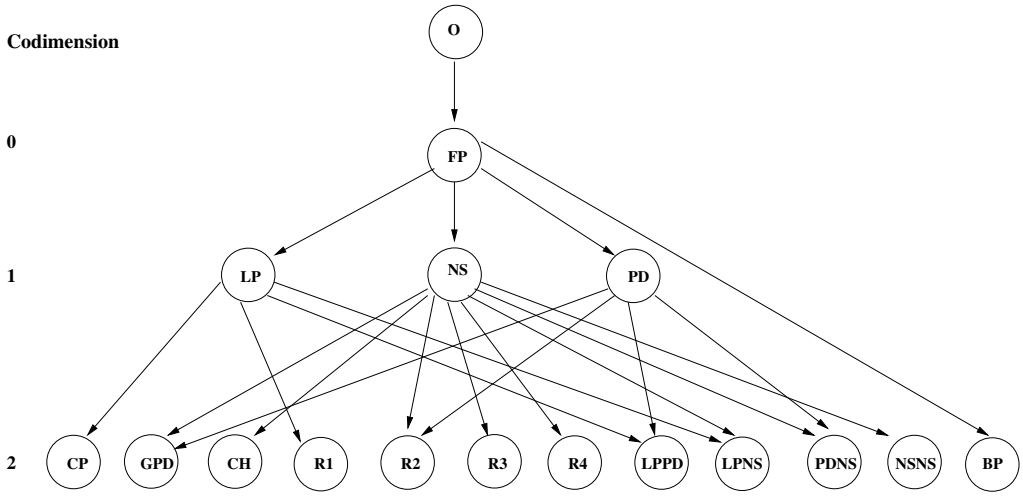


Figure 2.3: The detection diagram for maps.

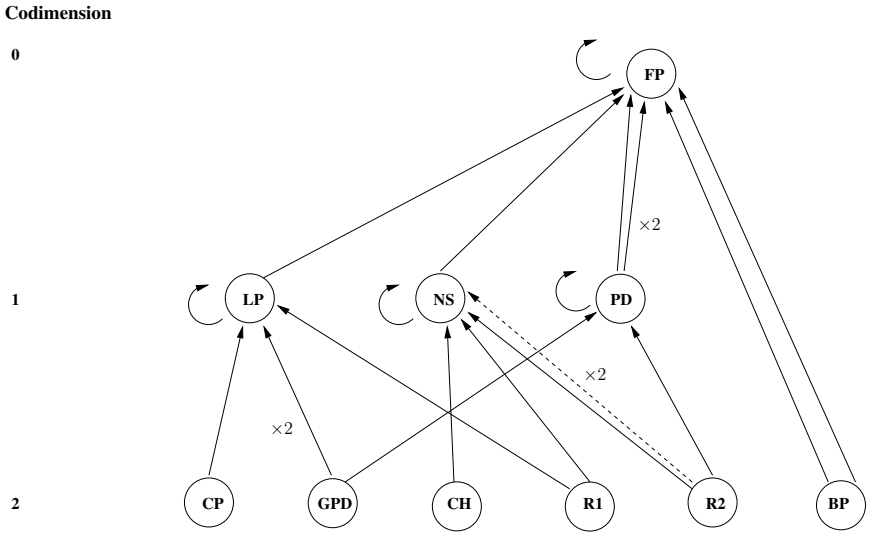


Figure 2.4: Branching diagram 1 for maps: dashed lines indicate switching subject to constraints and $\times 2$ indicates curve of double period.

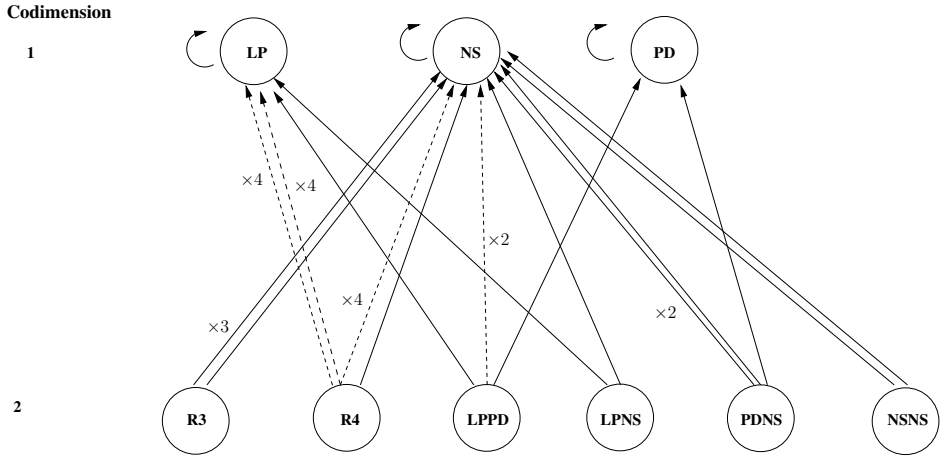


Figure 2.5: Branching diagram 2 for maps: dashed lines indicate switching subject to constraints and $\times 2(3, 4)$ indicates curve of double (triple,quadruple) period.

graphical libraries which allow for a quick programming. MATLAB is the language of choice in the engineering community and for many other scientists because the essential features can quickly be learned by following a few easily readable tutorials. Users can also easily learn the essentials of MatCont by going through a few tutorials, and apply this knowledge to study their own problems, export the results and produce figures for publication. The scope of application fields is therefore amazing, see §2.2.

2.6 Object-oriented programming and the GUI of MatCont

The GUI of MatCont has to deal with the complexity of the software and provide functionalities for the input of systems, computational options, control of computations with simultaneous output in 2D, 3D and numerical plots, analysing bifurcation points, archiving the obtained results and making them accessible for future use and for export to the general MATLAB environment.

Development of Graphical User Interfaces (GUI) and Object Oriented Programming (OOP) went hand in hand in the 1980's and became standard features in programming languages such as C++ and Java in the 1990's. In MATLAB the focus was rather on the ease of use of numerical programming and development tools for implementing numerical algorithms.

The first version of a MATLAB GUI for MatCont was developed in 2002-2003 by Annick Dhooge with the tools that were available at that time [24]. Though the algorithms were programmed from scratch, the aim was to have the outward look and feel of the predecessor package CONTENT [61] which was written in C++ and actually used OOP.

In the meantime the MATLAB programming language gradually acquired more func-

tionalities. The first step in this direction was the introduction of classes in which operations could be defined between objects of a specified class. An often-used feature was the overloading of operators. This was, in fact, used in 2005-2007 in the discrete-time version MatContM of MatCont to introduce automatic differentiation (AD) in the computation of normal form coefficients. The basic idea of AD is that the usual arithmetic and functional mathematical operations are overloaded with operations on Taylor polynomials [55].

Full object-oriented programming was introduced in MATLAB in two stages, namely in the releases 2008a and 2014b. In the 2008a release new language features were introduced for OOP, similar to the ones in C++ and Java such as handle classes which allow to use references instead of global structures. The Guide *Object-Oriented Programming* [67] was released with version 2012a. In the 2014b release object-oriented methods were introduced in the MATLAB graphics, which by the way caused some problems in the then current GUI MatCont version.

The present GUI for MatCont is completely rewritten and fully based on object-oriented principles with common practices from software design patterns. In particular it uses the MVC (Model-View-Controller) design principle towards which the MATLAB language is oriented since the 2008a release. According to this principle the "User" manipulates the "Controller". The "Model" receives input from the controller whenever input is needed and it also processes the input and checks its validity. The "Model" also updates the "View" through events and the "User" "sees" the "View". From the development point of view this eliminates the problem of handling the global structure `gds` which dominates the previous version of the GUI. However, the "Model" is quite complicated in the new version of the GUI and has several submodels. The information that was previously in the `gds` is now distributed over the submodels. One of the advantages that is visible to the user is in the consistent handling of the graphical output, which at times was awkward in the previous GUI, in particular after the changes in the MATLAB GUI library (R2014b update). The most obvious advantage is that a broader range of input is allowed (for example, a mathematical expression instead of just a number) while invalid input will be rejected. More details can be found in Chapter 6.

2.7 Comparison of MatCont and MatContM

The new GUI of MatCont is partly based on the GUI of MatContM which was written as part of the master thesis of N. Neirynck [71] but is much more comprehensive and presents many new features. The comprehensiveness is due to the fact that the underlying numerical code is also more complex (more curve types, more point types, more bifurcations, solvers for differential equations instead of maps, homotopy methods for initializing homoclinic and heteroclinic orbits, etc.) Among the new features we cite the following:

1. The MatContM GUI is based on the central idea of generating curves by numerical continuation. The new MatCont GUI is based on the more general idea of gener-

ating an output from a configuration. The configuration consists of a list of settings which correspond to the fields that appear in the Starter, Continuer and Integrator windows. Each object has a default value and 'knows' which type of information can be stored there. The configuration manager calls for the settings at the appropriate time and checks that input. In this way, it will not be possible to let the code crash by e.g. typing a question mark in a field that is meant for a parameter value. On the other hand, it will increase the flexibility by allowing to input MATLAB expressions. E.g. one can type $exp(2)$ in such a field; it will not be necessary to compute e^2 first and then enter its value.

The output is most typically a curve but could be adapted to be, for example, a Poincaré section or a set of Lyapunov exponents.

2. The MatCont GUI has a *central branch manager*. In this case a *branch* refers quite generally to a starting procedure for computations. This part of the code allows the developer to easily introduce a new type of computations, e.g. a new curve type or a computation of Poincaré sections. The central branch manager takes care of adapting the **Starter/Integrator/Continuer** windows as well as the output windows (2D, 3D, numeric).

3. The MatCont GUI has an `Output` (x,v,s,h,f) interpreter. This means that for every curve it keeps track of the meaning of the entries `x`,v,s,h,f. This serves two purposes. First, it allows a less error-prone handling of the output windows (2D, 3D, numeric). Second, it allows the user to understand the meaning of the output structure (Which entries correspond to parameters? Where is the Period in the columns of the $x-$ array when a branch of periodic orbits is computed? Which entries of the $h-$ vector correspond to userfunctions and which correspond to testfunctions for bifurcations?) In the pre-2018 version of the MatCont GUI and in the MatContM GUI this information is only available via the documentation or needs bookkeeping by the user (in the case of userfunctions and testfunctions).

4. Interaction between the command-line and GUI versions by modularity of the GUI. This implies that e.g. the visualisation parts of the MatCont GUI can be used to visualise results computed in Cl_MatCont.

2.8 Overview of contributions by N. Neirynck - I : General

Here we summarize our contributions prior to the development of the new MatCont environment.

2.8.1 Improvements and adaptations

- Merging of MatCont and Cl_MatCont. In MatCont5.4 (September 2014) and earlier versions of MatCont the command-line and GUI versions were separate. This was inconvenient from the point of algorithmic development since all algorithmic changes had to be input twice. N. Neirynck merged the two packages which required several important changes, since the continuer `cont.m` now runs in two different ways, depending on how the MATLAB session is started. The GUI version now runs on top of the command-line algorithms and allows more control with respect to pausing, resuming, extending, and output in graphical and numerical windows. Algorithmic improvements have to be implemented only once.

  Not all algorithms can be implemented easily in the GUI. So it is still possible to introduce, study and use algorithms in the command-line version before they are implemented in the GUI. An example is the routine `LimitCycle/initOrbLC.m` which allows to start the continuation of limit cycles from an orbit obtained by time integration. This functionality is also present in the GUI but does not use `initOrbLC.m`.

- In the 2014b release of MATLAB the GUI was restructured by using different Object Oriented methods which caused havoc in the MatCont output windows. Because the platform for the GUI was restructured, graphical input and output windows changed. The syntax of the commands that call the graphic libraries was changed. Windows could no longer be resized, labels disappeared and bifurcations could no longer be detected. Warnings about obsolete MATLAB constructions were interpreted as errors in the computation of testfunctions. The MatCont and MatContM codes were adapted by us to the changes in the MATLAB syntax. Since most warnings are suppressed in MatCont we wrote a script that checks all warnings that occur when running a MatCont or MatContM session and filters out the warnings about obsoleteness. In this way the developer can nearly automatically keep track of changes in the MATLAB syntax. Without this work the GUI packages would now be practically dead.

- Improved code for generating the system m-files. These files are sometimes called odefile or mapfile to indicate whether odes or maps are being studied. They are an essential part of the GUI versions of MatCont and MatContM. The symbolic derivatives in these m-files are generated by using the MATLAB symbolic toolbox. For a long time MATLAB offered the choice between the symbolic toolbox of Maple and that of MuPAD with Maple as the default. The Maple code puts some restrictions on the variable and parameter names. For example, 'C' could not be used and had to be replaced by a substitute name, for example 'CC' so that users sometimes had to use unnatural and awkward names. More recently, MATLAB decided to offer only MuPAD which made things worse since e.g. Latin names for Greek letters are also forbidden. We solved this problem by introducing an intermediate layer of names when using the symbolic toolbox. Internally, all parameter names are preceded by 'par_' and all coordinate names by 'coor_'. In this way, essentially all

restrictions on variable and parameter names are lifted. Also, in the present version spaces are allowed between the names of state variables, of parameters, and in the equations. However, restrictions still apply to the names of the auxiliary variables in the system definition files. These restrictions should be taken into account when using the system m-file generator (if not, the generator will give an MuPAD error message). Details and examples on the construction of system m-files are given in §3.6.

- Improved version of building system m-files in the case of userfunctions. In the previous versions of both MatCont and MatContM the symbolic derivatives of the system m-file were recomputed after each adding or removing of a userfunction. The new version is more efficient: it manipulates the structure of the set of userfunctions separately and reads the other symbolic derivatives of the system m-file from a saved MATLAB mat-file.

- Standalone version for building system m-files. The command line packages Cl_MatCont and Cl_MatContM also need system m-files. We wrote a standalone software that allows to build such files. The use of this code, essentially in the file `GUI/systems_standalone.m,` is documented in the MatCont manual [45] and the MatContM manual [50].

- General and algorithmic support of MatCont and MatContM. Here we present only some examples.

  - Running the curve object example (Appendix A in [45]) in older versions of MatCont worked nicely when it was executed after a fresh start of MATLAB. However, it failed invariably when other continuation runs were done before. This problem was caused by the lack of an initializer for the curve object and was solved by first declaring the global structure `cds` to be empty.
  - A long-standing bug in the code that generates system m-files was detected and solved. It involved the symbolic derivatives of fifth order in an ode or map with userfunctions.
  - In the older versions of MatCont the largest in magnitude eigenvalues of the saddle fixed points of homoclinic orbits were computed by calling `eigs` from MATLAB. These eigenvalues were produced in decreasing order of magnitude. The computations crashed when a changed version of `eigs` produced the eigenvalues in a more random way. As a remedy, `eigs` was replaced by `eig` and the eigenvalues were sorted afterwards.

2.8.2 New contributions

- An algorithm for switching to two different NS curves in a Double Neimark-Sacker (NSNS) point in MatContM. We implemented this algorithm and it is remarkably simple and efficient, and quite different from the idea that is traditionally used for

switching to a second branch of equilibria when a branch point of equilibria is detected on an equilibrium curve. The continuation variables in the continuation of a NS curve consist not only of the state variable x and the free parameter p but also of the scalar variable $k = \cos(\alpha)$ where the Neimark-Sacker eigenvalues of the Jacobian are $e^{\pm i\alpha}$. Hence the NSNS point corresponds in fact to two different points in (x, p, k) space with the same x and p but different k values. Therefore the two Neimark-Sacker branches can simply be started from these two points. In MatContM this corresponds to the initializers `init_NSNS_NS_same` and `init_NSNS_NS_other` where 'same' correspond to the curve on which the NSNS point was detected. We note that it is not necessary to compute tangent vectors and that it is even possible to change the choice of the free parameters, which is not the case in a branch point of equilibria.

- In Cl_MatContM manifold and connection computations were implemented, improved and documented in the manual [50]. They were also introduced in MatContM and discussed in the tutorials. More details are in §4.4.

- Lyapunov exponents in MatContM. MatContM now contains two routines to compute the Lyapunov exponents of a map. The class file `LyaExp.m` contains the routine from [7] to compute all Lyapunov exponents of a map. The class file `LyaExp2Dlargest.m` from [95] is a more restricted but efficient algorithm to compute the largest Lyapunov exponent in the case of planar systems. It was extensively used in [3]. Both class files are located in the directory `Lyapunov` and the output is written to the MATLAB workspace.

  The directory `GUI` contains two further files related to the computation of Lyapunov exponents. One of them is the file `FieldsModel.m` which acts as the *Model* part in a MVC setting. It contains the list of input variables that can be called by either of the Lyapunov algorithms. It also contain the conditions that check their validity. The *Controller* in the same MVC setting is the Starter window through which the user sets the values of the input variables. The other file is `Branch_LyaExp.m` which acts as the driver routine for the computations; it also acts as the *Viewer* in the MVC. The name of the file is derived from the fact that the computation of Lyapunov exponents is implemented as the computation of a new branch in a bifurcation point.

  In §4.3 practical details of the implementation are given. As an example application we study the monopoly model introduced in [76]. In this model we additionally detect stable behaviour in two small parameter intervals (length less than 2×10^{-3}).

- Unpublished but implemented work with L. Vanhulle: in her master thesis ([92], Ghent, September 2017) she developed an improved algorithm to compute in MATLAB the intersection points of two manifolds (typically a stable one and an unstable one). It was incorporated in MatContM in the file `Projectie2.m`. Details are given in §4.5.

2.8.3 Publications

- Paper [72]. This conference paper describes the then available functionalities of MatContM. It was written as a precursor paper to [3] and contains some example computations in the case of the monopoly model map of [76].

- Paper [3]. This example of an application of MatContM is joint work with B. Al-Hdaibat and W. Govaerts. It is a combined analytical and numerical study of a planar map that was introduced in [76] as a monopoly model in economics with cubic price and quadratic marginal cost functions. MatContM is used to compute branches of solutions of period 5, 10, 13 and 17 and to determine the stability regions of these solutions. General formulas for solutions of period 4 are derived analytically. It is shown that the solutions of period 4 are never linearly asymptotically stable. A nonlinear stability criterion is combined with basin of attraction analysis and simulation to determine the stability region of the 4-cycles. This corrects the erroneous linear stability analysis in previous studies of the model. The chaotic and periodic behavior of the monopoly model is further analyzed by computing the largest Lyapunov exponents, using the `LyaExp2Dlargest.m` implementation of the algorithm in [95]. This confirms the above-mentioned results. For more details and additional results see §4.3.1.

- TOMS paper with W. Govaerts, H.G.E. Meijer and Yu. A. Kuznetsov [73]. The introductory part of this paper is recalled in §4.4. The paper discusses the numerical study of bifurcations of homoclinic orbits of maps and applies the developed methods to obtain a rather complete bifurcation diagram of the resonance horn in a 1:5 Neimark-Sacker bifurcation point, revealing features that were unknown before, cf. the picture on the cover. This paper is presented in §4.6 and §4.7.

2.9 Overview of contributions by N. Neirynck - II : New MatCont

A major part of my work consists in developing a new MatCont-environment. We now summarize the main aspects:

- A clear separation of computational and control routines to increase flexibility, readability and maintainability.

- The workflow is consistently organized along the lines initializer — computation — solution. The notion of continuation is replaced by the more general notion of computation.

- A better handling of the generated data. These data are represented and managed by the diagram organizer, the data browser and the spreadsheet viewer.

- The internal working of the software is documented in Chapter 6. This chapter provides a general overview. More details are obtained through the internal documentation in the code which can be accessed online.

We now provide more details:

2.9.1 Software development aspects of the new MatCont

- Internally the structure of MatCont is completely reorganized. As an example, there are now only two plot functions (one for continuation, one for time integration) instead of one for every curve type.
- A central tool in the new GUI is the **Data Browser** which allows to navigate fast and easily through all stored results and use them to start new computations.
- The software contains automatic tests to check if a new MATLAB version produces the same results as the previous version.
- Error handling of plots is improved so that plot errors caused by e.g. command line interference, or by GUI interference when computations are suspended, do not crash the computations.
- Each input field has input restrictions and these are checked to minimize input errors. So for example it will not be possible to input a float or a question mark if a positive integer number is required. Errors are reported in the MATLAB command line. On the other hand, numerical fields can be filled with MATLAB expressions, provided they can be evaluated in the command line. So one can insert $2 * Pi$ instead of its decimal expansion 6.283184...

2.9.2 Functional aspects of the new MatCont

- Computation of Poincaré maps in MatCont. In the pre-2018 versions of MatCont Poincaré maps of ODEs were computed by using a specific curve type, called Discrete Orbit (DO). For this type of orbits the ODE solvers were manipulated to use an explicitly implemented approximation strategy to locate the Poincaré intersection points. In the new GUI there is no longer a specific curve type. The Poincaré sections are computed as a side result of the simulation by using the 'Events' option of the ODE solver. One shortcoming of the MATLAB solvers is that they do not interactively report on these events. In order to allow for interactive plotting of events, the plot routine will also monitor for sign changes in the values of the event function. Whenever a sign change is observed, the solver is called again on a smaller part of the curve to extract the event. Due to the implementation of this workaround, the list of detected events after computation in rare cases might contain more events than were displayed on an interactive plot. More information is given in §5.6 on the practical use of configuring events.

- A Command Line Interface (cli) allows a direct interaction between the command line and the GUI of MatCont. See §6.6.
- The GUI has a diagram organizer to define new diagrams, delete diagrams, move files between diagrams etcetera.
- Graphical 2D and 3D plots are reorganized: only one layout window is presented and nearly everything can be plotted.
- The Select Cycle functionality is now presented as a regular initializer from orbit to limit cycle.

2.9.3 Handling the `Options` in the new MatCont

The `Options` tab in the main **MatCont GUI** panel opens a list box window that allows the user to manage certain options in **MatCont**. Four of these options are global, namely:

- **Suspend Computation:** decide whether computations are suspended after each point, at special points or never.
- **Archive Filter:** number of unnamed curves of the same type that is preserved.
- **Output Interval:** number of continuation points that is computed before output is written to the **Plot2D, Plot3D, Numeric** and **Output** windows, and to the MATLAB command window.
- **Plot Properties:** general instructions for making plots.

These global options do not affect the computation of a curve and are not saved with the curve.

Three other options are computational and curve-related:

- **Jacobian Increment:** increment that is used in finite difference approximations for derivative functions when no symbolic derivatives are available
- **Moore-Penrose:** whether Moore-Penrose (default) or tangent continuation is to be used.
- **TSearchOrder:** cycling of unit tangent vectors in increasing or decreasing order of index when starting a continuation if no tangent vector is provided.

The choices for these options are saved with the curve so that the curve can be recomputed if required. The default values of the computational options are usually good and we recommend not to change them

The **Suspend Computation, Archive Filter** and **Output Interval** options were available in the **Options** panel of the older versions of MatCont. The **Moore-Penrose** option was present in the old MatCont but not visible in the GUI. The **Jacobian Increment** option was visible in all **Starter** windows separately. **Plot Properties** and **TSearchOrder** options are new.

2.9.4 Practical use aspects of the new MatCont

- In continuation plots it is possible to click on a found singular point to obtain information on the curve where it was found, the type of point and the normal form coefficients. By double clicking one selects the points as an initial point for another continuation.
- A spreadsheet view of a computed curve can be obtained by pressing the **View Curve** button in the **Curve** window.
- The Scroll-key can be used to scroll through MatCont windows. This functionality had to be implemented as MATLAB does not provide this functionality as part of their standard library.
- Several special keys can be used to control continuation computations, namely: **Escape** to stop, **Space bar** to resume, **Enter** to pause and **Control** to continue (when pressed) or to pause (when released). The use of the **Control** key is new.

Chapter 3

Numerical continuation: the algorithmic basis

3.1 Numerical continuation

In general, numerical continuation methods are used to compute solution manifolds of nonlinear systems of the form:

$$F(X) = 0, \tag{3.1}$$

where $X \in \mathbb{R}^{n+k}$ and $F : \mathbb{R}^{n+k} \to \mathbb{R}^n$ is a sufficiently smooth function. The solutions of this equation consist of regular pieces, which are joined at singular solutions. The regular pieces are curves when $k = 1$, surfaces when $k = 2$ and k-manifolds in general. Mathematically, this is a consequence of the Implicit Function Theorem.
We will use numerical continuation methods for analyzing the solutions of (3.1) when restricted to the case $k = 1$. In fact, we construct solution curves Γ in

$$\{X : F(X) = 0\}, \tag{3.2}$$

by generating sequences of points $X_i, i = 1, 2, ...$ along the solution curve Γ satisfying a chosen tolerance criterion. The general idea of a continuation method is that of a predictor-corrector scheme. Starting with an initial point on the continuation path, the goal is to trace the remainder of the path in steps. At each step, the algorithm first predicts the next point on the path along the tangent direction, and subsequently corrects the predicted point towards the solution curve. MatContM uses a variant of Moore-Penrose continuation which builds upon pseudo-arclength continuation; this amounts to using a variant of Newton's method for the corrector step, see Figure 3.1. For details of the continuation method used in (Cl_)MatCont and (Cl_)MatContM we refer to [45].

3.2 Testfunctions for bifurcations

Let $X = X(s)$ be a smooth, local parameterization of a solution curve of (3.1) where $k = 1$. Suppose that $s = s_0$ corresponds to a bifurcation point. A smooth scalar function

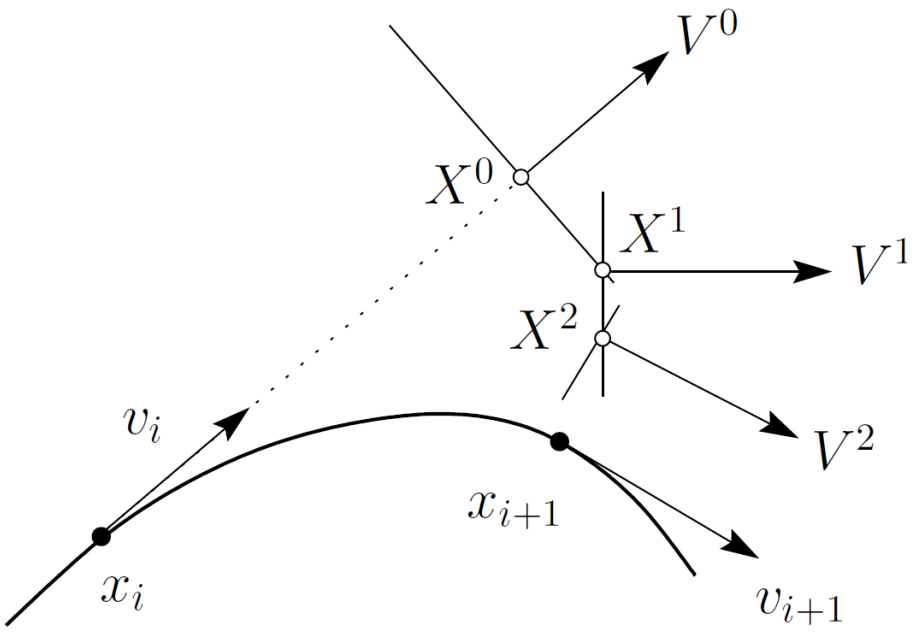


Figure 3.1: Moore-Penrose continuation with predicted/corrected points X^i and updated tangent vectors V^i

$\psi : \mathbb{R}^{n+1} \to \mathbb{R}^1$ defined along the curve is called a testfunction, a tool to detect singularities on a solution branch, for the corresponding bifurcation if $g(s_0) = 0$ and $g(s)$ changes sign at $s = s_0$, where $g(s) = \psi(X(s))$. The testfunction ψ is said to have a regular zero at s_0 if $\frac{dg}{ds}(s_0) \neq 0$. A bifurcation point is detected between two successive points X_0 and X_1 on the curve if $\psi(X_0)\psi(X_1) < 0$. To solve the augmented system

$$\begin{cases} F(X) = 0 \\ \psi(X) = 0, \end{cases} \tag{3.3}$$

MatCont uses a one-dimensional secant method to locate $\psi(X) = 0$ along the curve. Notice that this involves Newton corrections at each intermediate point.

3.3 Singularity matrix

Every singularity that can be expected along a curve needs at least one testfunction. Such a testfunction is usually not unique. For example, consider the situation where $X = [u, \lambda]$ with $u \in \mathbb{R}^n$ a state vector and $\lambda \in \mathbb{R}$ a parameter. Then (3.1) defines a curve of equilibria. The determinant of the Jacobian $F_u \in \mathbb{R}^{n\times n}$ is a testfunction for this bifurcation. The parameter-component $T_\lambda \in \mathbb{R}$ of the tangent vector $T = [T_u, T_\lambda]$ along the curve is another one. The user can choose the most convenient testfunction. Let us choose $\psi_1(X) = \det(F_u)$.

Unfortunately, if $\psi_1(X) = 0$ in an equilibrium point X this does not imply that X is a limit point of the equilibrium curve. Indeed, it can also be a branch point where two equilibrium branches intersect. To detect branch points we can use another testfunction ψ_2 where

$$\psi_2(X) = \det \begin{bmatrix} F_u & F_\lambda \\ T_u^T & T_\lambda \end{bmatrix}.$$

If $\psi_2(X) = 0$, then necessarily $\psi_1(X) = 0$ too, but not vice versa. So to make sure that we deal with a genuine limitpoint we also need $\psi_2(X) \neq 0$.

This type of situation is not rare. For example, along a curve of limitpoints both the Zero-Hopf bifurcation (ZH) and the Bogdanov-Takens (BT) bifurcation can be detected by the same testfunction which is essentially the same as the testfunction for a Hopf bifurcation along a curve of equilibria. So it is necessary to distinguish between the two cases by giving another testfunction which vanishes at one of ZH or BT and not at the other.

Let us suppose that on a particular curve n_s bifurcations are possible. Suppose also that we need n_t testfunctions defined along that curve where $n_t \geq n_s$.

To detect and identify all singularities we use a singularity matrix, i.e. a compact way to describe the relation between the singularities and the testfunctions. The singularity matrix S is an $n_s \times n_t$ matrix, such that:

$$S_{ij} = \begin{cases} 0 & \text{means : for singularity i testfunction j must vanish} \\ 1 & \text{means : for singularity i testfunction j must not vanish} \\ otherwise & \text{means : for singularity i ignore test function j} \end{cases} \tag{3.4}$$

As an example we consider again an equilibrium curve. In this case, a third bifurcation is also possible, namely that of a Hopf bifurcation. For this bifurcation a third testfunction ψ_3 is known and there is no need for a testfunction that does not vanish. With this ordering of the bifurcations a singularity matrix is given by:

$$S = \begin{pmatrix} 0 & 1 & 8 \\ 8 & 0 & 8 \\ 8 & 8 & 0 \end{pmatrix} \tag{3.5}$$

We note that in MatCont the bifurcations are ordered differently so that the singularity matrix looks somewhat different. Also, the testfunction for Hopf in fact detects Jacobian matrices with a pair of eigenvalues with sum zero, i.e. not only Hopf points but also neutral saddles with two real eigenvalues with sum zero.

3.4 Userfunctions

The user has the possibility to define specific functions which must be scalar and can depend only on the state variables and parameters. Userfunctions can only be active during continuation runs. Also, using state variables is only meaningful in the case of

equilibrium, Hopf, limit point and branch point curves. The user can request that the zeros of userfunctions are detected and computed during continuation runs as if they were singular points. This requires that the options `Userfunctions` and `UserfunctionsInfo` are set properly (see §3.5.3) and that the *system m-file* defines the userfunctions (§3.6).

3.5 Software

3.5.1 Continuer

The syntax of the continuer is:

$$[\mathtt{x}, \mathtt{v}, \mathtt{s}, \mathtt{h}, \mathtt{f}] = \mathtt{cont}(@\mathtt{curve}, \mathtt{x0}, \mathtt{v0}, \mathtt{options}) \tag{3.6}$$

Here `curve` is a MATLAB m-file that contains the curve description, cf. §3.5.2.
`x0` and `v0` are the initial point and the tangent vector at the initial point where the continuation starts, respectively.
`options` is a structure as described in §3.5.3.
The function returns:
`x` and `v`, i.e. the points and their tangent vectors along the curve. Each column in `x` and `v` corresponds to a point on the curve.
`s` is an array of structures; each entry is a structure that contains information about a detected singular point. This structure has the following fields:

s.index	index of the singularity point in `x`
s.label	label of the singularity
s.data	any kind of extra information
s.msg	a string containing a message for this particular singularity

`h` is used for output of the algorithm, currently this is a matrix with for each point a column with the following components (in that order) :

- Stepsize:
  Stepsize used to calculate this point (zero for initial point and singular points)
- Half the number of correction iterations, rounded up to the next integer
  For singular points this is the number of locator iterations
- Userfunction values :
  The values of all active userfunctions
- Testfunction values :
  The values of all active testfunctions

We note that the meaning of the values in the h-output not only depends on the curve type, but also on the choice of the active userfunctions and testfunctions during the continuation run.

In general, `f` can be anything depending on which curve file is used. However, `f` always contains eigenvalues if they are computed during the continuation. Eigenvalues are computed when *options* is set by

```
options=contset(options,'Eigenvalues',1);
```

`f` always contains multipliers if they are computed during the continuation. Multipliers are computed when *options* is set by

```
options=contset(options,'Multipliers',1);
```

See §3.5.3 for more details.

It is also possible to extend the most recently computed curve with the same options (also the same number of points) as it was first computed. The syntax to extend this curve is:

```
[x, v, s, h, f] = cont( x, v, s, h, f, cds)
```

`x`, `v`, `s`, `h` and `f` are the results of the previous call to the continuer and *cds* is the global variable that contains the curve description of the most recently computed curve. The function returns the same output as before extended with the new results.

3.5.2 Curve files

The continuer uses *curve definition files,* i.e. special m-files in which the type of the solution branch is defined. In MatCont twelve types are implemented, namely in the files `branchpoint.m`, `branchpointcycle.m`, `equilibrium.m`, `heteroclinic.m`, `homoclinic.m`, `homoclinicsaddlenode.m`, `hopf.m`, `limitcycle.m`, `limitpoint.m`, `limitpointcycle.m`, `neimarksacker.m` and `perioddoubling.m`. Each curve type has its own directory in MatCont.

In MatContM eight curve types are implemented, namely in the files `fixedpointmap.m`, `heteroclinic.m`, `heteroclinicT.m`, `homoclinic.m`, `homoclinicT.m`, `limitpointmap.m`, `perioddoublingmap.m` and `neimarksackermap.m`. Each curve type has its own directory in MatContM.

A `curve definition file` contains several sections such as *curve_func*, *jacobian*, *hessians*, *adapt*, etc. In some cases the problem definition uses auxiliary entities like bordering vectors and it may be needed to adapt them during the continuation. In *adapt* these entities are adapted. If *cds.options.Adapt* has a value n, then after n computed points a call
`[reeval,x,v]=feval(cds.curve_adapt,x,v)`
is executed. It is required that n be a nonnegative integer; if $n = 0$ then no adaptations are done. For some curve types, e.g. equilibrium and fixed point, *adapt* is an empty routine anyway. In the case of limit cycles the mesh is adapted at each call to `feval(cds.curve_adapt,x,v)`.

3.5.3 Options

In the continuation we use the *options* structure which is initially created with *contset*:
`options = contset`
will initialize the structure. The continuer stores the handle to the options in the variable *cds.options*. Options can then be set using
`options = contset(options, optionname, optionvalue);`
where *optionname* is an option from the following list.

InitStepsize the initial stepsize (default: 0.01)

MinStepsize the minimum stepsize to compute the next point on the curve (default: 10^{-5}). It is assumed that the minimum stepsize is not larger than the initial stepsize.

MaxStepsize the maximum stepsize (default: 0.1). It is assumed that the maximum stepsize is not smaller than the initial stepsize.

MaxCorrIters maximum number of correction iterations (default: 10)

MaxNewtonIters maximum number of Newton-Raphson iterations before switching to Newton-Chords in the corrector iterations (Jacobian is no longer updated) (default: 3)

MaxTestIters maximum number of iterations to locate a zero of a testfunction (default: 10)

Increment the increment to compute the derivatives numerically (default: 10^{-5})

FunTolerance tolerance of function values: $||F(x)|| \leq FunTolerance$ is the first convergence criterion of the Newton iteration (default: 10^{-6})

VarTolerance tolerance of coordinates: $||\delta x|| \leq VarTolerance$ is the second convergence criterion of the Newton iteration (default: 10^{-6})

TestTolerance tolerance of testfunctions (default: 10^{-5})

Singularities boolean indicating the presence of singularities (default: 0)

MaxNumPoints maximum number of points on the curve (default: 300)

Backward boolean indicating the direction of the continuation (direction of the initial tangent vector v_0) (default: 0)

CheckClosed number of points indicating when to start to check if the curve is closed (0 = do not check) (default: 50)

Adapt number of points indicating when to adapt the problem while computing the curve (0 = do not adapt) (default: 3)

IgnoreSingularity vector containing indices of singularities which are to be ignored (default: empty)

Multipliers boolean indicating the computation of the multipliers (default: 0)

TSearchOrder numerical value that indicates if unit vectors are cycled in increasing order of index (default: 1, increasing) or decreasing (set to a value different from 1), see §3.5.12.

Userfunctions boolean indicating the presence of userfunctions (default: 0)

UserfunctionsInfo is an array with structures containing information about the userfunctions. This structure has the following fields:

.label	label of the userfunction (must consist of four characters, including possibly trailing spaces)
.name	name of this particular userfunction
.state	boolean indicating whether the userfunction has to be evaluated or not

For the options MaxCorrIters, MaxNewtonIters, MaxTestIters, Increment, FunTolerance, VarTolerance, TestTolerance and Adapt the default values are in most cases good.

Options also contains some fields which are not set by the user but frozen or filled by calls to the curvefile, namely:

MoorePenrose boolean indicating the use of the Moore-Penrose continuation as the Newton-like corrector procedure (default: 1; if 0 then pseudo-arclength is used)

SymDerivative the highest order symbolic derivative which is present (default: 0)

SymDerivativeP the highest order symbolic derivative with respect to the parameter(s) which is present (default: 0)

AutDerivative boolean indicating the use of automatic differentiation in the computation of normal form coefficients, not present in MatCont (default: 1)

AutDerivativeIte an integer number that indicates the use of automatic differentiation when the iteration number of the map equals or exceeds this number, not present in MatCont (default: 24)

Testfunctions boolean indicating the presence of testfunctions and singularity matrix (default: 0)

WorkSpace boolean indicating to initialize and clean up user variable space (default: 0 and no other value is used in MatCont or MatContM)

Locators boolean vector indicating for which testfunctions a specific locator code exists to locate its zeroes. Otherwise the default locator is used (default: empty)

ActiveParams vector containing indices of the active parameter(s) (default: empty)

Some more details follow here on some of the options.

3.5.4 Derivatives of the defining system of the curve

In the defining system of the object that is to be continued, the derivatives can be provided that are needed for the continuation algorithm or other computations. The continuer stores the handles to the derivatives in the variables `cds.curve_jacobian` and `cds.curve_hessians`.

If `cds.symjac`= 1, then a call to `feval(cds.curve_jacobian, x)` must return the $(n-1) \times n$ Jacobian matrix evaluated at point x.

If `cds.symhess`= 1, then a call to `feval(cds.curve_hessians, x)` must return a 3-dimensional $((n-1) \times n \times n)$ matrix H such that $H(i,j,k) = \frac{\partial^2 F_i(x)}{\partial x_j \partial x_k}$.

In the present implementation `cds.symhess`= 0 in all cases. The curve definition file does not provide second order derivatives, since they are not needed in the used algorithms.

3.5.5 Singularities and testfunctions

To detect singularities on the curve one must set the option *Singularities* on. Singularities are detected using the singularity matrix, as described in section 3.3. The continuer stores the handles to the singularities, the testfunctions and the processing of the singularities respectively in the variables `cds.curve_singmat`,`cds.curve_testf` and `cds.curve_process`.

A call to `[S,L] = feval(cds.curve_singmat)` gets the singularity matrix S and a vector of strings which are abbreviations (labels) of the singularities.

A call to `feval(cds.curve_testf, ids, x, v)` then must return the evaluation of all testfunctions, whose indices are in the integer vector `ids`, at `x` (`v` is the tangent vector at `x`). As a second return argument it should return an array of all testfunction id's which could not be evaluated. If this array is not empty the newly found point on the curve is not accepted and the stepsize is decreased.

When a singularity is found, a call to
`[failed,s] = feval(cds.curve_process,i,x,v,s)` will be made to process singularity `i` at `x`. This is the point where computations can be done, like computing normal forms, eigenvalues, etc. of the singularity. These results can then be saved in the structure `s.data` which can be reused for further analysis. Note that the first and last point of the curve are also treated as singular.

3.5.6 Locators

It may be useful to have a specific locator code for locating certain singularities. To use a specific locator the user must set the option *Locators*. This is a vector in which the index of an element corresponds to the index of a singularity. Setting the entry to 1 means the presence of a user-defined locator. The continuer has stored the handles to the locators in

the variable `cds.curve_locator` and will then make a call to
`[x,v]=feval(cds.curve_locate,i,x1,v1,x2,v2)`
to locate singularity i which was detected between `x1` and `x2` with their corresponding tangent vectors `v1` and `v2`. It must return the located point and the tangent vector at that point. If the locator was unable to find a point it should return `x = []`.

3.5.7 Userfunctions

To detect zeros of userfunctions on the curve one must set the option *Userfunctions* on. The continuer has stored the handles to the userfunctions `cds.curve_userf`. First a call to `UserInfo = contget(cds.options, 'UserfunctionsInfo', [])` is made to get information on the userfunctions. A call to `feval(cds.curve_userf, UserInfo, ids, x, v)` then must return the evaluation of all userfunctions ids, whose information is in the structure `UserInfo`, at `x` (`v` is the tangent vector at `x`). As a second return argument it should return an array of all userfunction id's which could not be evaluated. If this array is not empty the stepsize will be decreased.

A special point on a bifurcation curve that is specified by a userfunction has a structure as follows:

`s.index`	index of the detected singular point defined by the userfunction.
`s.label`	a string that is in `UserInfo.label`, label of the singularity.
`s.msg`	a string that is set in `UserInfo.name`.
`s.data`	an empty tangent vector and values of the userfunctions and testfunctions in the singular point.

3.5.8 Defaultprocessor

In many cases it is useful to do some general computations for every calculated point on the curve. The results of these computations can then be used by for example the testfunctions. The continuer has stored the handle to the defaultprocessor in the variable `cds.curve_defaultprocessor`.

The defaultprocessor is called as
`[failed,f,s] = feval(cds.curve_defaultprocressor,x,v,s)`.
`x` and `v` are the point on the curve and its tangent vector. The argument `s` is only supplied if the point is a singular point, in that case the defaultprocessor may also add some data to the `s.data` field. If for some reason the default processor fails it should set `failed` to 1. This will result in a reduction of the stepsize and a retry which should solve the problem. Any information that is to be preserved, should be put in `f`. `f` must be a column vector and must be of equal size for every call to the default processor.

3.5.9 Special processors

After a singular point has been detected and located a singular point data structure will be created and initialized. If there are some special data (like eigenvalues) which may be

of interest for a particular singular point then a call to
`[failed,s] = feval(cds.curve_process,i,x,v,s)`
should store this data in the `s.data` field. Here `i` indicates which singularity was detected and `x` and `v` are the point and tangent vector where this singularity was detected.

3.5.10 Workspace

During the computation of a curve it is sometimes necessary to introduce variables and perform additional computations that are common to all points of the curve. The continuer has stored the handle to the initialization and clean-up of the workspace in the variables `cds.curve_init` and `cds.curve_done`. Initialization and clean-up can be relegated to a call of the type

```
feval(cds.curve_init,x,v).
```

This option has to be provided only if the variable `WorkSpace` in `cds.options` is switched on. In this case a call

```
feval(cds.curve_done,x,v)
```

must clear the workspace. Variables in the workspace must be set global. In the GUI of MatCont and MatContM `cds.options.Workspace` is never switched on.

3.5.11 Adaptation

It is possible to adapt the problem while generating the curve. If *Adapt* has a value, say 5, then after 5 computed points a call to `[reeval,x,v]=feval(cds.curve_adapt,x,v)` will be made where the user can program to change the system.

For some applications it is useful to change or modify the used testfunctions while computing the curve (like in bordering techniques). In order to preserve the correct signs of the testfunctions it is sometimes necessary to reevaluate the testfunctions after adaptation. To do this `reeval` should be one otherwise zero. The return variables `x` and `v` should be the updated `x` and `v` which may have changed because of the changes made to the system.

3.5.12 Tangent search order

To start a continuation, an initial point x_0 and a tangent vector v_0 are needed in general. Often, only x_0 is available. In this case, MatCont and MatContM successively try all unit vectors as candidate tangent vectors. By default, this is done in increasing order of index (cds.options.TSearchOrder = 1). If cds.options.TsearchOrder is set to a value different from 1 then the cycling is done in decreasing order of index.

In cases where the number of continuation variables is large (e.g. when computing homoclinic connections) the choice of cds.options.TSearchOrder can substantially change the speed of the computation.

3.6 The system m-file of an ode or map

A solution curve must be initialized before doing a continuation. Each curve file has its own initializers which use a *system m-file* where the ode or map is defined. In the first case the system m-file is also called the *odefile*, in the latter case the *mapfile*. A system m-file contains at least the following sections (in that order):
init, *fun_eval*, *jacobian*, *jacobianp*, *hessians*, *hessiansp*, *der3*, *der4* , *der5*.
A system m-file may also contain one or more sections that describe userfunctions.

We note that if state variables or parameters are added or deleted then this constitutes another dynamical system. So either all computed data should be deleted or ignored, or the name of the system should be changed. For simplicity and robustness the last option is strongly recommended. A system m-file can be defined by simply using the MATLAB editor or any other text editor.

In order to illustrate the recommended method and the elements of a *mapfile*, we consider the example M_{TN}, the map of a truncated normal form, i.e. the two-dimensional map, introduced in [60], §9.9, unfolding the normal form of an *R2* point to which it reduces for $\beta_1 = \beta_2 = 0$.

$$M_{TN} : \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix} \longmapsto \begin{pmatrix} -1 & 1 \\ \beta_1 & -1+\beta_2 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix} + \begin{pmatrix} 0 \\ C\xi_1^3 + D\xi_1^2\xi_2 \end{pmatrix} \tag{3.7}$$

A new *mapfile* can be created by calling `SysGUI.new`.

This opens a **System** window, which contains several fields and buttons. To identify the system, type for example

```
Tnfmap
```

in the **Name** field (it must be one word).

Input names of the **Coordinates**: `ksi1`, `ksi2`, and the **Parameters**: `beta1`, `beta2`, `CC`, `DD`. In the case of ODEs there is also an input field by which *time* can be given a name. The default is t and usually there is no reason to change that.

If shown, select symbolic generation of the 1st order derivatives by pressing the corresponding radio-button [1].

Finally, in the large input field, type the RHS of the truncated normal form map as

```
ksi1'=-ksi1+ksi2
ksl2'=beta1*ksi1-ksi2+beta2*ksi2+CC*ksi1^3+DD*ksi1^2*ksi2
```

[1] If the MATLAB Symbolic Toolbox is present, there will be buttons indicated 'symbolically'. The first-order derivatives are used in some of the integration algorithms, the first- and second-order derivatives are used in the continuation, while the third-order derivatives are employed in the normal form computations. The derivatives of fourth and fifth order are only used in the normal form computations of some codimension 2 bifurcations.

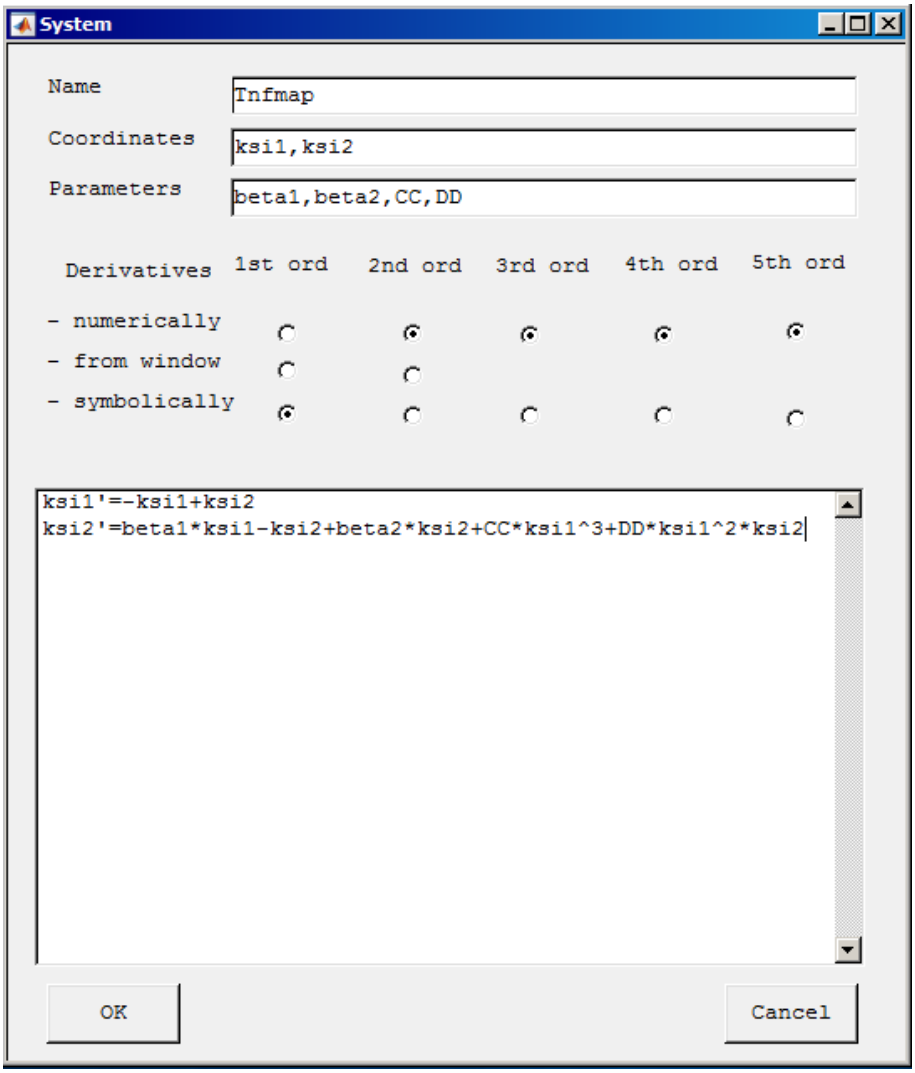


Figure 3.2: Specifying a new model, the truncated normal form map (3.7).

Avoid typical mistakes:

- Make sure the multiplication is written explicitly with $*$.
- Specify the right hand sides in the same order as the coordinates.

It is best not to add comma's or semicolons after the equations. Now the **System** window should look like in Figure 3.2, and the user can press the **OK** button. Two new files will be created in the `Systems` directory of MatContM, namely the mapfile `Tnfmap.m` and a `mat`-file `Tnfmap.mat`.

The mapfile can be edited later on by calling `SysGUI.edit(@name)` where `name` is the name of an existing *mapfile*. Userfunctions can be added by calling `SysGUI.userfunctions(@name)`. See Figure 3.3.

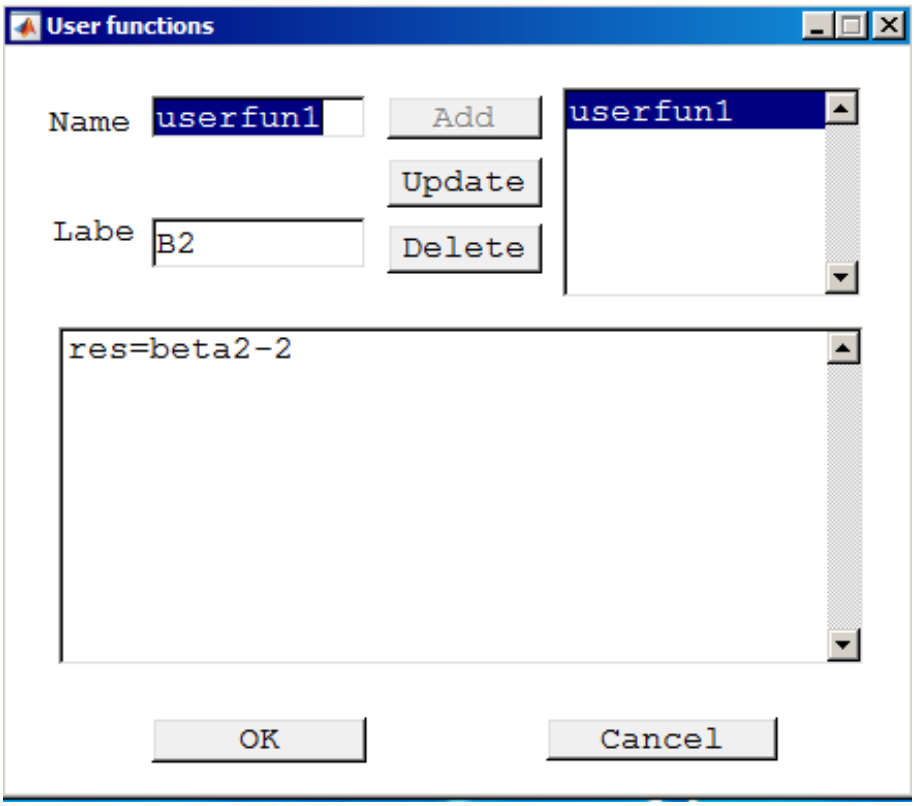


Figure 3.3: Adding a userfunction.

First we give the mapfile of M_{TN} using symbolic derivatives up to order 5:

```
function out = Tnfmap
out{1} = [];
out{2} = @fun_eval;
out{3} = @jacobian;
out{4} = @jacobianp;
out{5} = @hessians;
out{6} = @hessiansp;
out{7} = @der3;
out{8} = @der4;
out{9} = @der5;

% --------------------------------------------------------------------
function dydt = fun_eval(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
```

```
dydt=[-kmrgd(1)+kmrgd(2);
par_beta1*kmrgd(1)-kmrgd(2)+par_beta2*kmrgd(2)+par_CC*kmrgd(1)^3
                          +par_DD*kmrgd(1)^2*kmrgd(2);];

% ------------------------------------------------------------------------
function jac = jacobian(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
jac=[ -1 , 1 ; par_beta1 + 3*kmrgd(1)^2*par_CC + 2*kmrgd(1)*kmrgd(2)*par_DD ,
               par_beta2 + kmrgd(1)^2*par_DD - 1 ];
% ------------------------------------------------------------------------
function jacp = jacobianp(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
jacp=[ 0 , 0 , 0 , 0 ; kmrgd(1) , kmrgd(2) , kmrgd(1)^3 ,
                       kmrgd(1)^2*kmrgd(2) ];
% ------------------------------------------------------------------------
function hess = hessians(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
hess1=[ 0 , 0 ; 6*kmrgd(1)*par_CC + 2*kmrgd(2)*par_DD , 2*kmrgd(1)*par_DD ];
hess2=[ 0 , 0 ; 2*kmrgd(1)*par_DD , 0 ];
hess(:,:,1) =hess1;
hess(:,:,2) =hess2;
% ------------------------------------------------------------------------
function hessp = hessiansp(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
hessp1=[ 0 , 0 ; 1 , 0 ];
hessp2=[ 0 , 0 ; 0 , 1 ];
hessp3=[ 0 , 0 ; 3*kmrgd(1)^2 , 0 ];
hessp4=[ 0 , 0 ; 2*kmrgd(1)*kmrgd(2) , kmrgd(1)^2 ];
hessp(:,:,1) =hessp1;
hessp(:,:,2) =hessp2;
hessp(:,:,3) =hessp3;
hessp(:,:,4) =hessp4;
%------------------------------------------------------------------------
function tens3  = der3(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
tens31=[ 0 , 0 ; 6*par_CC , 2*par_DD ];
tens32=[ 0 , 0 ; 2*par_DD , 0 ];
tens33=[ 0 , 0 ; 2*par_DD , 0 ];
tens34=[ 0 , 0 ; 0 , 0 ];
tens3(:,:,1,1) =tens31;
tens3(:,:,1,2) =tens32;
tens3(:,:,2,1) =tens33;
tens3(:,:,2,2) =tens34;
%------------------------------------------------------------------------
function tens4  = der4(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
tens41=[ 0 , 0 ; 0 , 0 ];
tens42=[ 0 , 0 ; 0 , 0 ];
...
```

```
tens47=[ 0 , 0 ; 0 , 0 ];
tens48=[ 0 , 0 ; 0 , 0 ];
tens4(:,:,1,1,1) =tens41;
tens4(:,:,1,1,2) =tens42;
...
tens4(:,:,2,2,1) =tens47;
tens4(:,:,2,2,2) =tens48;
%-----------------------------------------------------------------------
function tens5  = der5(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
tens51=[ 0 , 0 ; 0 , 0 ];
tens52=[ 0 , 0 ; 0 , 0 ];
...
tens515=[ 0 , 0 ; 0 , 0 ];
tens516=[ 0 , 0 ; 0 , 0 ];
tens5(:,:,1,1,1,1) =tens51;
tens5(:,:,1,1,1,2) =tens52;
...
tens5(:,:,2,2,2,1) =tens515;
tens5(:,:,2,2,2,2) =tens516;
```

We observe that:

- In the case of maps `out` {1} is by default empty. This output field is provided for cases in which the user wants to do some initializations. In the case of odes it has a default content which for the `lpneuron.m` example with 13 state variables is:

  ```
  function [tspan,y0,options] = init
  handles = feval(lpneuron);
  y0=[0,0,0,0,0,0,0,0,0,0,0,0,0];
  options = odeset('Jacobian',handles(3),'JacobianP',handles(4),
                   'Hessians',handles(5),'HessiansP',handles(6));
  tspan = [0 10];
  ```

- The state variables are collected in a vector `kmrgd`.

- Internally the parameters are renamed to avoid clashes with the name restrictions of the symbolic toolbox. E.g. `CC` is replaced by `par_CC`.

After adding a userfunction userfun1=beta2-2 we obtain:

```
function out = Tnfmap
out{1} = @init;
out{2} = @fun_eval;
out{3} = @jacobian;
```

```
out{4} = @jacobianp;
out{5} = @hessians;
out{6} = @hessiansp;
out{7} = @der3;
out{8} = @der4;
out{9} = @der5;
out{10}= @userfun1;

% ---------------------------------------------------------------------
function dydt = fun_eval(t,kmrgd,beta1,beta2,CC,DD)
dydt=[-kmrgd(1)+kmrgd(2);
beta1*kmrgd(1)-kmrgd(2)+beta2*kmrgd(2)+CC*kmrgd(1)^3+DD*kmrgd(1)^2*kmrgd(2);];

% ---------------------------------------------------------------------
function jac = jacobian(t,kmrgd,beta1,beta2,CC,DD)
jac=[ -1 , 1 ; beta1 + 3*CC*kmrgd(1)^2 + 2*DD*kmrgd(1)*kmrgd(2) ,
               beta2 + DD*kmrgd(1)^2 - 1 ];
% ---------------------------------------------------------------------
function jacp = jacobianp(t,kmrgd,beta1,beta2,CC,DD)
jacp=[ 0 , 0 , 0 , 0 ; kmrgd(1) , kmrgd(2) , kmrgd(1)^3 , kmrgd(1)^2*kmrgd(2) ];
% ---------------------------------------------------------------------
function hess = hessians(t,kmrgd,beta1,beta2,CC,DD)
hess1=[ 0 , 0 ; 6*CC*kmrgd(1) + 2*DD*kmrgd(2) , 2*DD*kmrgd(1) ];
hess2=[ 0 , 0 ; 2*DD*kmrgd(1) , 0 ];
hess(:,:,1) =hess1;
hess(:,:,2) =hess2;
% ---------------------------------------------------------------------
function hessp = hessiansp(t,kmrgd,beta1,beta2,CC,DD)
hessp1=[ 0 , 0 ; 1 , 0 ];
hessp2=[ 0 , 0 ; 0 , 1 ];
hessp3=[ 0 , 0 ; 3*kmrgd(1)^2 , 0 ];
hessp4=[ 0 , 0 ; 2*kmrgd(1)*kmrgd(2) , kmrgd(1)^2 ];
hessp(:,:,1) =hessp1;
hessp(:,:,2) =hessp2;
hessp(:,:,3) =hessp3;
hessp(:,:,4) =hessp4;
%---------------------------------------------------------------------
function tens3  = der3(t,kmrgd,beta1,beta2,CC,DD)
tens31=[ 0 , 0 ; 6*CC , 2*DD ];
tens32=[ 0 , 0 ; 2*DD , 0 ];
tens33=[ 0 , 0 ; 2*DD , 0 ];
tens34=[ 0 , 0 ; 0 , 0 ];
tens3(:,:,1,1) =tens31;
```

```
tens3(:,:,1,2) =tens32;
tens3(:,:,2,1) =tens33;
tens3(:,:,2,2) =tens34;
%-----------------------------------------------------------------------
function tens4  = der4(t,kmrgd,beta1,beta2,CC,DD)
tens41=[ 0 , 0 ; 0 , 0 ];
tens42=[ 0 , 0 ; 0 , 0 ];
...
tens47=[ 0 , 0 ; 0 , 0 ];
tens48=[ 0 , 0 ; 0 , 0 ];
tens4(:,:,1,1,1) =tens41;
tens4(:,:,1,1,2) =tens42;
...
tens4(:,:,2,2,1) =tens47;
tens4(:,:,2,2,2) =tens48;
%-----------------------------------------------------------------------
function tens5  = der5(t,kmrgd,beta1,beta2,CC,DD)
tens51=[ 0 , 0 ; 0 , 0 ];
tens52=[ 0 , 0 ; 0 , 0 ];
...
tens515=[ 0 , 0 ; 0 , 0 ];
tens516=[ 0 , 0 ; 0 , 0 ];
tens5(:,:,1,1,1,1) =tens51;
tens5(:,:,1,1,1,2) =tens52;
...
tens5(:,:,2,2,2,1) =tens515;
tens5(:,:,2,2,2,2) =tens516;
%-----------------------------------------------------------------------
function userfun1=userfun1(t,kmrgd,beta1,beta2,CC,DD)
userfun1=beta2-2;
```

If no symbolic derivatives are available then MatCont uses finite difference approximations instead. However, this will be less accurate and the computed normal form coefficients are often unreliable, in particular when higher-order derivatives are involved. MatContM can also use automatic differentiation (AD). A mapfile of M_{TN} without symbolic derivatives is given by:

```
function out = Tnfmap
out{1} = [];
out{2} = @fun_eval;
out{3} = [];
out{4} = [];
out{5} = [];
```

```
out{6} = [];
out{7} = [];
out{8} = [];
out{9} = [];
out{10}= @userfun1;

% --------------------------------------------------------------------
function dydt = fun_eval(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
dydt=[-kmrgd(1)+kmrgd(2);
par_beta1*kmrgd(1)-kmrgd(2)+par_beta2*kmrgd(2)
                  +par_CC*kmrgd(1)^3+par_DD*kmrgd(1)^2*kmrgd(2);];

% --------------------------------------------------------------------
function jac = jacobian(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
% --------------------------------------------------------------------
function jacp = jacobianp(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
% --------------------------------------------------------------------
function hess = hessians(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
% --------------------------------------------------------------------
function hessp = hessiansp(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
%--------------------------------------------------------------------
function tens3  = der3(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
%--------------------------------------------------------------------
function tens4  = der4(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
%--------------------------------------------------------------------
function tens5  = der5(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
function userfun1=userfun1(t,kmrgd,par_beta1,par_beta2,par_CC,par_DD)
userfun1=beta2-2
```

Chapter 4

MatContM for maps

In this chapter we consider dynamical systems generated by smooth nonlinear maps

$$x \mapsto f(x,\alpha), \quad x \in \mathbb{R}^n,\ \alpha \in \mathbb{R}^m \tag{4.1}$$

with state variable x and parameter vector α.

A sequence $(x_k)_{k\in\mathbb{Z}}$ is called a *connecting orbit* of f if

$$\begin{array}{rcll} \lim\limits_{k\to-\infty} x_k & = & x_{-\infty}, & \\ f(x_k,\alpha) & = & x_{k+1}, & \forall k \in \mathbb{Z}, \\ \lim\limits_{k\to+\infty} x_k & = & x_{+\infty}, & \end{array} \tag{4.2}$$

where $x_{-\infty}$ and $x_{+\infty}$ are (necessarily) fixed points of f. The connecting orbit is called *homoclinic* if $x_{+\infty} = x_{-\infty}$ and *heteroclinic* otherwise. From a geometric point of view, the connecting orbit lies in the intersection of the unstable manifold $W^u(x_{-\infty})$ of $x_{-\infty}$ and the stable manifold $W^s(x_{+\infty})$ of $x_{+\infty}$. These manifolds are defined as

$$W^s(x_{+\infty}) = \left\{x \in \mathbb{R}^n : f^{(J)}(x,\alpha) \to x_{+\infty} \text{ as } J \to +\infty\right\}, \tag{4.3}$$

and

$$W^u(x_{-\infty}) = \left\{x \in \mathbb{R}^n : \exists\{q_k\}, q_0 = x \text{ and } f(q_{k+1},\alpha) = q_k, \text{ and } \lim_{k\to\infty} q_k = x_{-\infty}\right\}. \tag{4.4}$$

MatContM builds on the command line code Cl_MatContM described in [50] and [55] but supports several new functionalities, as well as providing a uniform interface. Comprehensive tutorials are provided that illustrate the use of MatContM by investigating example models. The tutorials can be found on [64]

4.1 Features and functionalities of MatContM

In a typical use of MatContM, one starts with an initial fixed point or cycle, which may be obtained from analysis, simulations or previous continuations. One first computes curves

of fixed points or cycles under variation of one parameter, and may detect bifurcation points on such curves. Starting from such bifurcation points, the continuer algorithm in MatContM can compute bifurcation curves. These curves are defined by a system of equations consisting of fixed point and bifurcation conditions. With one free parameter we can also compute curves of connecting orbits. By varying two system parameters we can compute bifurcation curves of limit points, period-doubling and Neimark-Sacker points as well as tangencies of homoclinic and heteroclinic orbits. The systems that define connecting orbits and their tangencies are fairly complicated as they involve the saddle equilibria at the endpoints of the orbits, their eigenspace structures and the whole connecting orbit. The following list contains functionalities that are provided by MatContM:

- Simulation (iteration) of maps, i.e. computation and visualization of orbits (trajectories).
- Computation of the Lyapunov exponents of long trajectories.
- Continuation of fixed points of maps and iterates of maps with respect to a control parameter.
- Detection of fold (limit point), flip (period-doubling point), Neimark-Sacker and branch points on curves of fixed points.
- Computation of normal form coefficients for fold, flip and Neimark-Sacker bifurcations.
- Continuation of fold, flip and Neimark-Sacker bifurcations in two control parameters.
- Detection of all codimension 2 fixed point bifurcations on curves of fold, flip and Neimark-Sacker bifurcations.
- Computation of normal form coefficients for all codimension 2 bifurcations of fixed points.
- Switching to the period doubled branch in a flip point.
- Branch switching at branch points of fixed points.
- Switching to branches of codimension 1 bifurcations rooted in codimension 2 points.
- Automatic differentiation for normal form coefficients of codimension 1 and codimension 2 bifurcations.
- Computation of one-dimensional invariant manifolds (stable and unstable) and in the two-dimensional case computing their transversal intersections to obtain initial homoclinic and heteroclinic connections.
- Continuation of homoclinic and heteroclinic orbits with respect to a control parameter and the detection of tangencies on the curve of orbits.
- Continuation of homoclinic and heteroclinic tangencies in two control parameters.

4.2 Basic practical use of MatContM

A simple but important feature is that MatContM can simulate maps, i.e. compute orbits (trajectories). These orbits can also be visualized and their Lyapunov exponents can be computed, see §4.3.

The advanced use of MatContM relies on the ability to continue curves under variation of parameter(s) and apply advanced bifurcation theory on given examples of maps. The aim of the package is to allow a user to perform a bifurcation analysis of a map, without deep knowledge of the workings of MatContM or even the continuation software.

The user is able to enter a system, an initial fixed point and start computing with most of the settings left on default. This should lower the entry barrier for researchers from different research fields who want to investigate their models but do not want to be confronted with specification and implementation details.

The continuation curves can be visualized using the plot capabilities of MatContM; this can be done during and after the continuation.

Tools are provided to help with managing systems, diagrams and curves.

Specific features have been implemented to make branch switching between continuation curves fast and easy. An initial point can be selected out of a list of special points or it can be selected by double-clicking on a graph of the computed curve. A list of available curves for computation is shown depending on the type of the initial point.

An interface is also provided that allows the exchange of information between MatContM and the MATLAB command line. This enables the user to combine MatContM with other MATLAB software or simply to use computed data on the MATLAB command line.

Figure 4.1 is an action screenshot of MatContM that gives a visual impression of the computation of a cascade of period-doubling bifurcations in a predator-prey model [1].

4.3 Lyapunov exponents in MatContM

The computation of Lyapunov exponents of maps (and in fact also of ode's) is often useful since it helps to find attractors and to compute their dimensions. However, it depends in a rather unpredictable way on the initial point of the orbit and therefore should be used with care and usually combined with other techniques. MatContM contains two routines to compute the Lyapunov exponents of a map. The class file `LyaExp.m` contains the routine from [7] to compute all Lyapunov exponents of a map. The class file `LyaExp2Dlargest.m` from [95] is a more restricted but efficient algorithm to compute the largest Lyapunov exponent in the case of planar systems. The directory `GUI` contains two further files related to the computation of Lyapunov exponents. One of them is the file `FieldsModel.m` which acts as the *Model* part in a MVC setting. It contains the list of input variables that can be called by either of the Lyapunov algorithms. It also contain the conditions that check their validity. The *Controller* in the same MVC setting is the Starter window through which the user sets the values of the input variables. The other file is `Branch_LyaExp.m` which

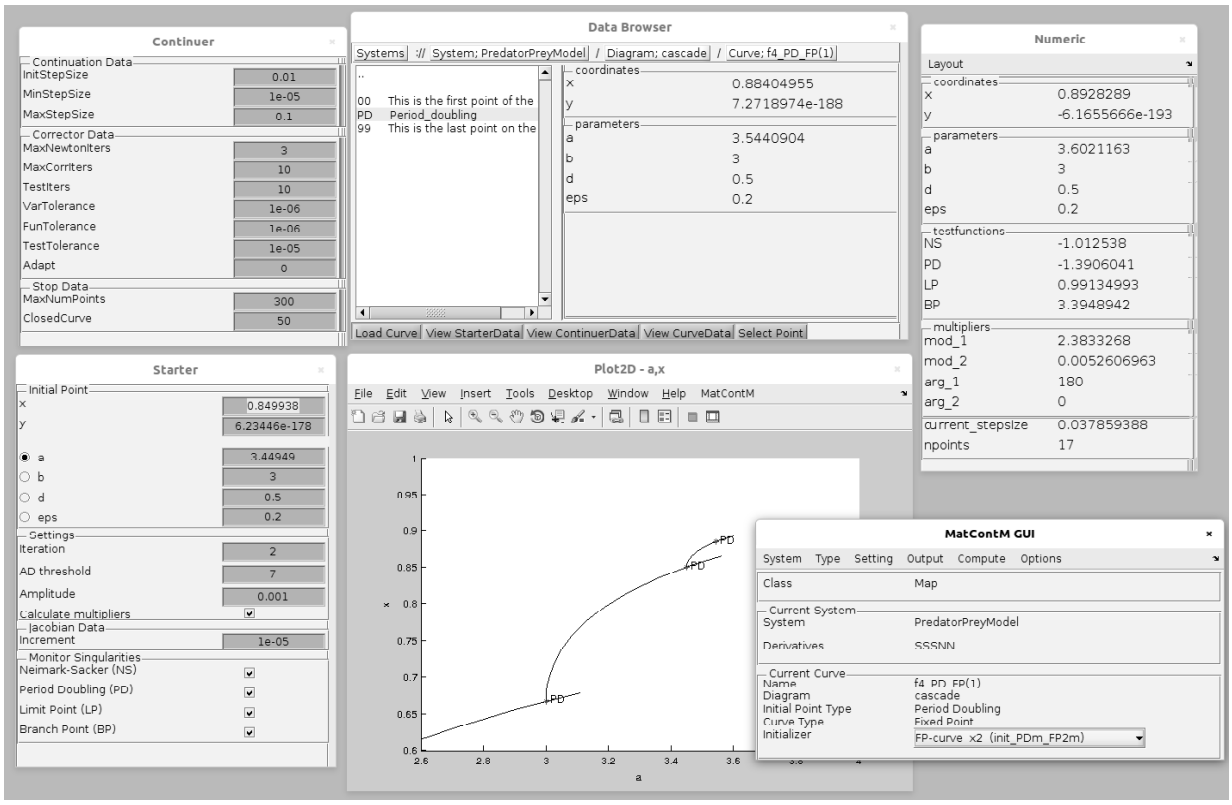


Figure 4.1: This action screenshot of MatContM presents the main windows that are opened during the computation of a cascade of period-doubling bifurcations in a predator-prey model [1]. Bottom right is the MatContM main window which contains general information on the model that is being used and the top-level commands. Top left is the Continuer window which shows the continuation variables. The fields of this window are independent of the particular curve that is being computed, though the numeric values can vary. Bottom left is the Starter window whose fields strongly depend on the curve that is being computed. Top middle is the Data Browser which allows to inspect all introduced systems, computed curves, etcetera. Top right is the Numeric window which during computation provides numerical values of computed quantities. Bottom middle is the 2D plot window which in this case plots several bifurcations curves; the horizontal axis presents a parameter of the system and the vertical axis is a state variable. The PD points are flip bifurcation points; the period-doubling cascade is clearly visible.

acts as the driver routine for the computations; it also acts as the *Viewer* in the MVC. The name of the file is derived from the fact that the computation of Lyapunov exponents is implemented as the computation of a new branch in a bifurcation point.

Both Lyapunov computation routines are implemented in a generalized way, in fact by computing the exponents for a vector of values of a user-chosen system parameter. In the

case where all Lyapunov exponents are computed the input fields in the Starter window are the following:

- Lyapunov steps : number of applications of the (possibly iterated) map.
- norm steps : number of steps between two consecutive normalizations; not present in the case of the `LyaExp2Dlargest`.
- report every x normalizations : output is sent to the workspace after that number of normalizations
- transient iterations : computing the Lyapunov exponents starts after that number of map applications
- reuse latest computed state : if this boolean variable is 'true', then the next computation of Lyapunov exponents starts with the latest computed state (for the previous value of the parameter in the vector of parameters) as initial state.
- parametervalues : a MATLAB expression that generates a vector of floating point numbers.

If only the largest Lyapunov coefficient is computed then the input field "norm steps" is not present and the field "report every x normalizations" is replaced by "report every x iterations".

The option of reusing the latest computed state is mainly useful if a series of Lyapunov exponents is computed for varying values of a system parameter. If the option is "on" then there is a better chance to stick to a (varying) stable attractor of the system, while with the "off" option there is a better chance to find "new" attractors. Both can be useful.

4.3.1 Example: a monopoly model

We consider the monopoly model introduced in [76] and recently studied in [3]. It can be reformulated as a planar map

$$M : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} y \\ y + \delta P(x, y) \end{pmatrix}, \tag{4.5}$$

where δ is a positive parameter and x, y are the quantities of a commercial good that are produced at two consecutive time points in the search of optimal profit. Furthermore,

$$P(x, y) = 3.6 - 2.4(x + y) + 0.6(x^2 + xy + y^2) - 0.05(x^3 + x^2y + xy^2 + y^3). \tag{4.6}$$

We will compute the Lyapunov exponents for a range of values of $\delta \in]0, 3.5]$. To start the computations we choose the point type "point" and as curve type one of the two provided algorithms; in the case of Figure 4.2 it is the Benettin algorithm [7] to compute all Lyapunov exponents.

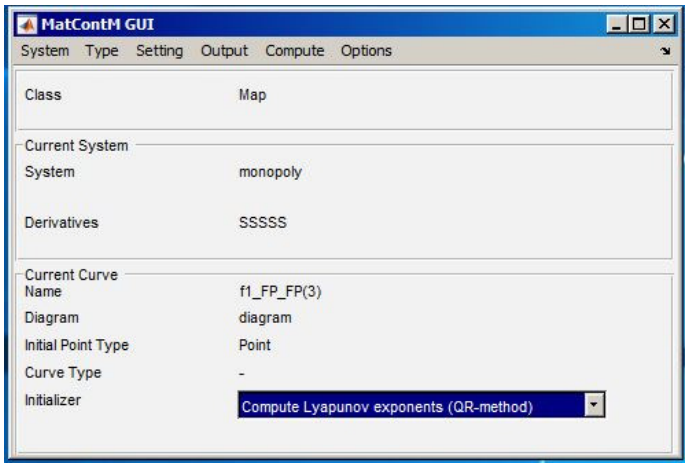


Figure 4.2: The main window when starting the computation of all Lyapunov exponents.

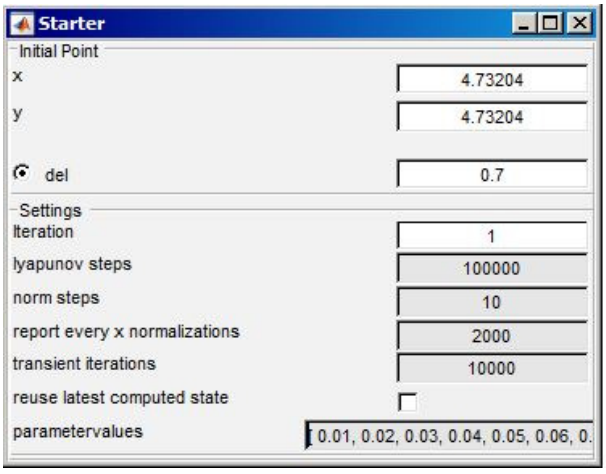


Figure 4.3: The Starter window in the case of the Benettin algorithm.

We then open the **Starter** window and fill it as in Figure 4.3.

We note that all orbits start from the same point in state space which is computed as $(3+\sqrt{3}-10^{-5}; 3+\sqrt{3}-10^{-5})$. This is a small perturbation of the point $(3+\sqrt{3}; 3+\sqrt{3})$ which is a fixed point for all $\delta > 0$, cf. [3]. We also note that the parameter δ is named "del" in the code. The field "parametervalues" is filled by typing $[0.01 : 0.01 : 3.5]$. When the **Enter** button is hit, this MATLAB expression generates the vector $[0.01; 0.012; \ldots; 3.49; 3.50]$ and the Lyapunov exponents will be computed for all entries of this vector as parameter values for δ.

The computations are started by clicking **Compute**|**Forward**. During the computations, the output is shown in an **Output** window, see Figure 4.4.

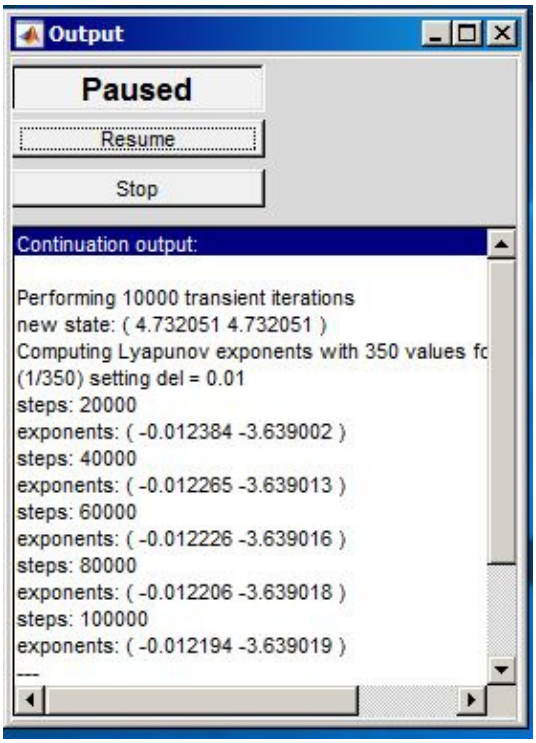


Figure 4.4: The Output window in the case of the Benettin algorithm.

At the end of the computations the main results are stored in the MATLAB workspace in a structure called `lyapunovExponents`. This structure has two fields, namely `lyapunovExponents.del` which returns the vector of used parametervalues (the extension "del" refers to the name of the parameter) and a 2×350 matrix named `lyapunovExponents.exponents`. Each column of this matrix corresponds to a parameter-value and gives the values of the Lyapunov exponents in decreasing order.

With standard MATLAB commands we can then generate Figure 4.5. Note that for a considerable range of $\delta-$ values the two Lyapunov exponents are equal.

In practice the largest Lyapunov exponent is usually the most important one since it indicates when a fixed point or cycle is stable. In the case of planar maps it can be computed separately without normalizations. In MatContM this is done by using the "Compute largest Lyapunov exponent (2D-only)" field instead of the "Compute Lyapunov exponents (QR-method)" field in Figure 4.2. In the corresponding **Starter** window (Figure 4.6) the field "norm steps" is missing while the field "report every x normalizations" is replaced by "report every x iterations".

In Figure 4.7 we show the largest Lyapunov exponent for $\delta \in]0, 3.5]$. Figure 4.8 is a zoom of Figure 4.7 with $\delta \in [2.400, 2.401, 2.402, \ldots, 2.999, 3.000]$.

We add the following observations:

1. It is known [3] that for $\delta \in]0, \frac{5}{3}[$, (4.5) has a stable fixed point with $x = y = 3$. This

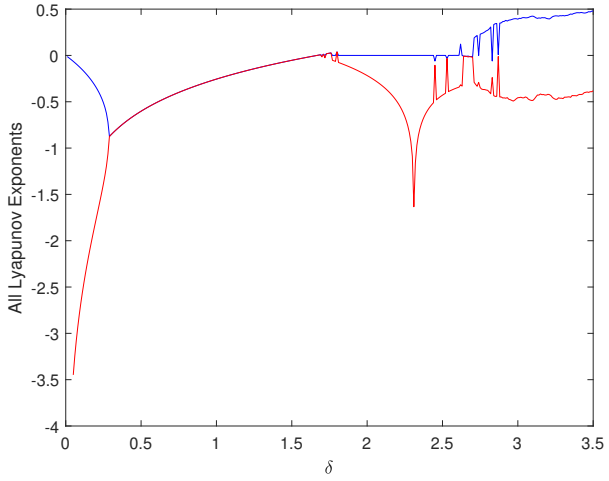


Figure 4.5: The largest (blue) and second (red) Lyapunov exponents for a range of δ values.

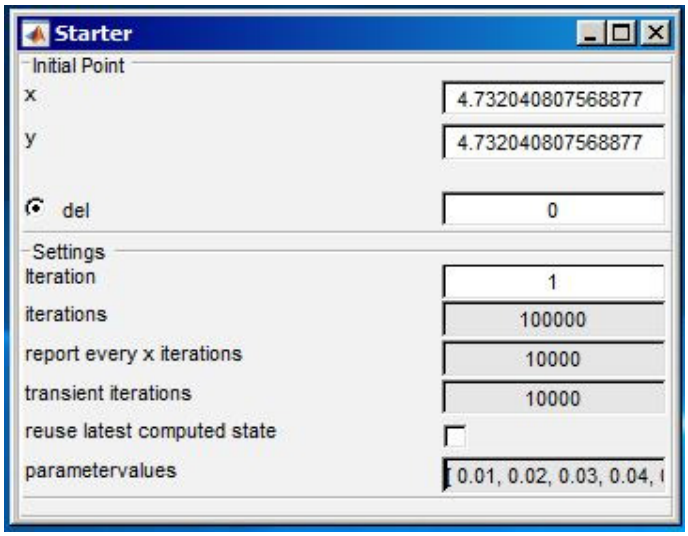


Figure 4.6: The Starter window when computing the largest Lyapunov exponent with the 2D algorithm.

is confirmed by Figure 4.7 where the largest Lyapunov exponent is negative in that range.

2. The computed largest Lyapunov exponent is close to zero for $\frac{5}{3} < \delta < 2.615$ though we know from [3] that around $\delta = 2.45$ there is a stable cycle of period 17.

3. There is a range of negative values for δ between 2.62 and $\delta = 2.7$. This is consistent

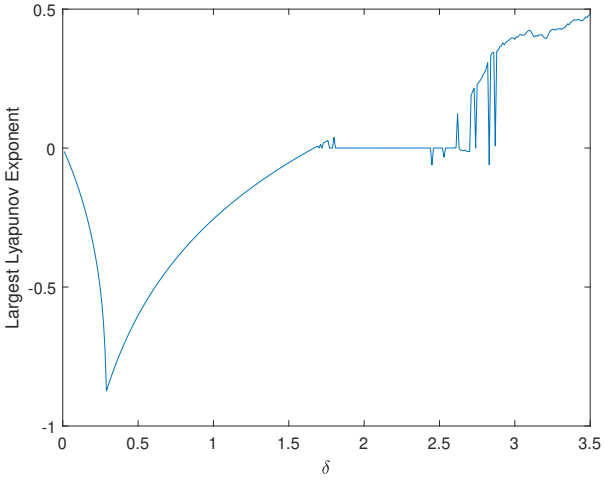


Figure 4.7: The largest Lyapunov exponent for $\delta \in]0, 3.5]$.

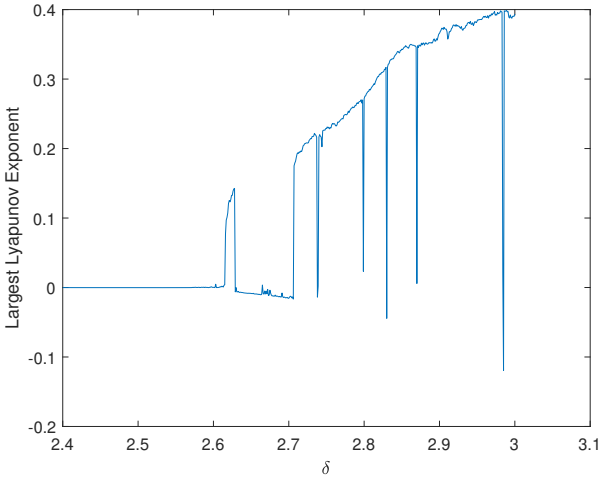


Figure 4.8: Zoom of Figure 4.7.

with [3] where stable periodic behaviour with period 5 is found in that region.

4. For $\delta > 2.7$, the largest Lyapunov exponent increases and the system becomes more and more chaotic. However, there are three isolated values where the largest Lyapunov exponent is negative, namely for $\delta = 2.7380$ with coefficient -0.0138, for $\delta = 2.8300$ with coefficient -0.0444 and for $\delta = 2.9850$ with coefficient -0.1193. Only the value $\delta = 2.8300$ is reported in [3] and explained by the presence of a stable periodic orbit with period 13. The other values remain to be investigated.

4.4 Growing one-dimensional unstable and stable manifolds

4.4.1 One-dimensional unstable manifolds

To compute one-dimensional unstable manifolds we use a slightly improved version of the algorithm described in [57] (the improvement is mainly in a better bookkeeping of the metadata of the algorithm). The algorithm starts with a saddle fixed point. Near the fixed point, the algorithm uses the unstable eigenspace (which is assumed to be one-dimensional) to approximate the unstable manifold.

Two issues must be kept in mind. First, both directions along the unstable manifold can possibly lead to a connecting orbit and should be tried. Second, if the unstable multiplier is negative, then the second iterate of the map is automatically used, so that the orientation along the manifold is preserved.

The first part is to choose a point in the unstable eigenspace close to the saddle point and add points along the unstable manifold by applying the map. This procedure is applied as long as the distance between two consecutive points is within some interval and an angle condition is satisfied.

The main part consists of adding new points in the direction away from the fixed point which are at each step at most a prescribed distance from the last added point. The distance changes from step to step with the curvature of the manifold; many points are added when encountering sharp folds while few points are added during locally straight lines. To avoid cutting of sharp folds, a restriction is placed on the angle of the new point and the last two points. Our implementation of this algorithm (an adaptation of the implementation in [54]) also keeps track of the segment of the manifold approximation that contains the preimage of each newly added point.

4.4.2 One-dimensional stable manifolds

England et al. [32] describe an algorithm for growing one-dimensional stable manifolds for planar maps in which the inverse is not explicitly or implicitly (i.e. by a Newton-like iteration) used. However, their approach, called circle search, does not generalize to higher-dimensional spaces. We therefore use a slightly improved version of an algorithm by C. Bruschi which essentially uses the inverse in the form of Newton-like iterations, implicitly assuming that the manifold is locally unique (the improvement is mainly in a better bookkeeping of the metadata of the algorithm). The algorithm starts with a saddle fixed point. Near the fixed point, the algorithm uses the stable eigenspace (which is assumed to be one-dimensional) to approximate the stable manifold.

The same issues as for unstable manifolds must be kept in mind.

In the first part of the algorithm we choose a point in the stable eigenspace sufficiently close to the saddle point and add points along the stable manifold in the direction towards the fixed point by applying the inverse map (implicitly, by Newton iteration) as long as the distance between two points is below a threshold and an angle condition is satisfied.

The main part consists of adding new points in the direction away from the fixed point which are at each step at a distance approximately equal to or smaller than δ_k, which is one of the threshold parameters in the code. Let p_{end} be the last computed point on the stable manifold.

The first step of the procedure is sketched in Figure 4.9. It is first checked that $\|f^{-1}(p_{end}) - p_{end}\| > \delta_k$, where $f^{-1}(p_{end})$ is computed by Newton iterations. If this is not the case, then δ_k is decreased. Next, by tracing back the computed points along the stable manifold two consecutive points q_L and q_R are identified such that $\|f^{-1}(q_R) - p_{end}\| > \delta_k$ and $\|f^{-1}(q_L) - p_{end}\| < \delta_k$. In the process, $p_R = f^{-1}(q_R)$ and $p_L = f^{-1}(q_L)$ are identified. It is not excluded that $p_L = p_{end}$.

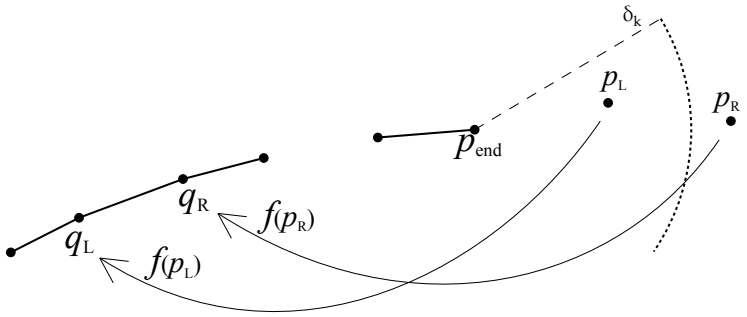


Figure 4.9: Computing a new point on the stable manifold: first step. The dotted arc represents a sphere of radius δ_k around p_{end}. This figure is adapted from a figure by C. Bruschi.

Next, the zero-order approximation $\bar{p}$ to the sought next point p_k on the stable manifold is computed as

$$\bar{p} = p_{end} + \delta_k \frac{p_R - p_{end}}{\|p_R - p_{end}\|},$$

see Figure 4.10. The further aim is to obtain p_k as the solution to the system that consists of

$$f(p_k) = q_L + \tau(q_R - q_L) \tag{4.7}$$

and

$$\langle p_k - \bar{p}, \bar{p} - p_{end} \rangle = 0. \tag{4.8}$$

The geometric meaning of these conditions is clear from Figure 4.10. Together, (4.7) and (4.8) constitute a system of $n+1$ equations in $n+1$ variables, namely the components of p_k and the scalar τ, where n is the dimension of the state space. To obtain a first-order approximation of (p_k, τ) we solve the linearized system

$$\begin{bmatrix} J|_{\bar{p}} & -q_R + q_L \\ (\bar{p} - p_{end})^T & 0 \end{bmatrix} \begin{bmatrix} p_k \\ \tau \end{bmatrix} = \begin{bmatrix} q_L - f(\bar{p}) + J|_{\bar{p}}\bar{p} \\ (\bar{p} - p_{end})^T \bar{p} \end{bmatrix}, \tag{4.9}$$

which is equivalent to a single Newton step, starting with $(\bar{p}, 0)^T$ as initial guess.

It turns out that in many cases the Jacobian matrix of the system (4.7)-(4.8) is ill-conditioned at the solution point. Therefore, in practice we keep τ fixed and solve (4.7) by a Newton iteration. The code contains several further checks, e.g. an angle condition (to handle the case of sharp fold points) and the requirement that τ is positive and smaller than a chosen threshold. If one of these requirements is not satisfied, then δ_k is decreased and the process is started again.

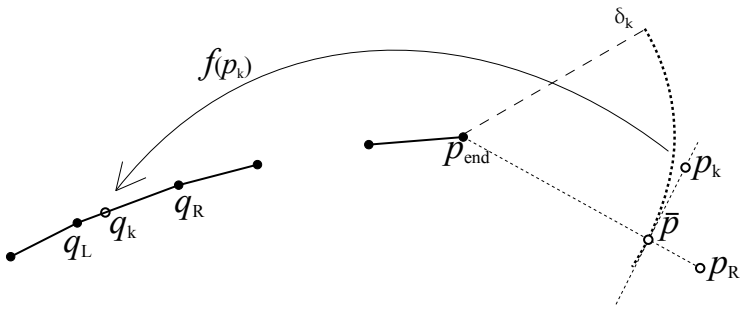


Figure 4.10: Computing a new point on the stable manifold: second step. This figure is adapted from a figure by C. Bruschi.

4.5 Projection algorithm for intersecting manifolds

In this section we restrict to planar maps and the computation of the intersection points of two manifolds (in fact, a stable and an unstable manifold). This is the key step towards the computation of homoclinic and heteroclinic connections and tangencies.

Since each manifold is approximated by line segments this intersection problem is a special case of the computation of the intersection points of a collection of line segments. The latter problem is a well-studied one and the standard solution is the *line sweep algorithm*, see [19]. But so far there is no efficient way to implement this algorithm in MATLAB, in spite of several attempts, see [92].

In fact, the standard solution so far is to simply consider each pair of line segments (a stable one and an unstable one) and check whether they intersect [11]. We call this the algorithm of Bruschi. It is obviously rather inefficient but it is still acceptable since its cost is typically much smaller than the cost of computing the manifolds.

We present an algorithm based on the idea of first looking into one projection of the two manifolds. In a first round we select the pairs of intervals whose $x-$ projections overlap. Typically this is a small fraction of all pairs. In the second round we select from this fraction those pairs which also have overlapping $y-$ projections. For the remaining pairs, usually a tiny fraction of all pairs, we check whether they have an intersection in 2D-space. We present two versions of this algorithm, which we call the first one and the improved one, respectively. The first version is coded in the m-file `Projectie.m` and listed in Appendix E. The improved version is coded in the m-file `Projectie2.m` and listed in Appendix F. For

comparison purposes both versions are available in MatContM but only the improved version is called in the GUI.

4.5.1 First version of the projection algorithm

The first part of this algorithm might be called a *line sweep* algorithm though it is different from the 'classical' line sweep algorithm in [19]. First the starting points of all intervals (both from the stable and unstable manifolds) are ordered in a list L by increasing values of their $x-$ coordinates. For each point one keeps track of the intervals to which it belongs, whether it is a starting point or an endpoint in the $x-$ projection and to which manifold the interval belongs.

We then start with an empty list A and successively consider all values in the list L in increasing order. At each point of the list L, say x_0, we consider all intervals that have x_0 as a starting point in the $x-$ projection. These are stored in another list C as couples with all intervals in the list A that belong to the other manifold. Then all intervals that have x_0 as an endpoint in the $x-$ direction are removed from list A.

When the last point of list L is taken into account we have a list C of all couples of intervals that belong to different manifolds and have an overlap in the $x-$ projection. In the next step we go through C and check for each couple if they also have an overlap in the $y-$ direction. If not, that couple is removed from list C. Finally, for each remaining couple of intervals we check if they really have an intersection in $(x, y)-$ space.

As an example we consider the generalized Hénon map [39]:

$$F : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_2 \\ \alpha - \beta x_1 - x_2^2 + R x_1 x_2 + S x_2^3 \end{pmatrix}, \tag{4.10}$$

where α, β, R and S are parameters.

We consider the saddle fixed point $(-1.621146385, -1.621146385$ for the parameter values $\alpha = -0.4$, $\beta = 1.03$, $R = -0.1$, $S = 0$

In Figure 4.11 we show the stable (blue) and unstable (red) manifolds of this point as created by the function `growman.m` in MatContM with 20000 points each. In Figure 4.12 we consider a zoom of Figure 4.11.

In Figure 4.12 the $x-$ projections of the line segments on the stable and unstable manifolds are presented separately. Suppose that all points to the left of the (green) sweep line (which does not contain any line segment endpoint of one of the two manifolds) have been considered. Then the list A contains exactly all intervals that cross the green line. The next point to be treated is x_0, the point to the right of the green line with smallest $x-$ coordinate, see Figure 4.13.

The green line moves to that point. The (red) line segment l that has x_0 as a starting point is added to list A. Every line segment in A that belongs to the other manifold (the blue intervals in Figure 4.13) can possibly have an intersection in 2D-space with l. So all couples that consist of a blue interval and the interval l are added to the list C. Then all

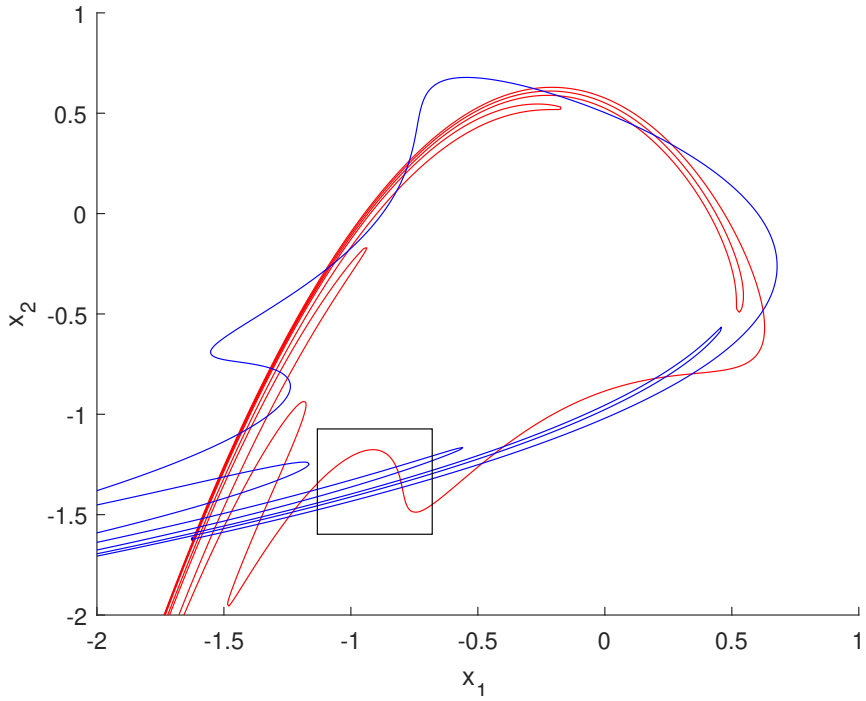


Figure 4.11: Stable (blue) and unstable (red) manifolds of the generalized Hénon map (4.10).

intervals in A that have x_0 as an endpoint in the $x-$ direction are removed from list A (in the present case there are none).

Next, all couples of intervals are checked in the $y-$ direction. In the case of Figure 4.13 there are no intersections in the $y-$ direction, so these couples are removed from list C. Finally, the remaining couples in C are checked for intersections in (x, y) space which gives us the final list of intersection points.

4.5.2 Improved version of the projection algorithm

In the version of §4.5.1 the list A contains intervals from both the stable and unstable manifolds and this has to be checked each time when an interval is considered. In the next, improved version we make two lists, say A_u and A_s for the unstable and stable manifolds, respectively. Suppose that a new point x_0 is considered. Then each interval l that has x_0 as a starting point in the $x-$ direction is added to its own $A-$ list. If l belongs to the stable manifold, then it is coupled in the $C-$ list with each interval in the A_u- list and vice versa. If this is done for all intervals that have x_0 as a starting point in the $x-$ direction, then we remove all intervals in both A_u and A_s that have x_0 as an endpoint in the $x-$ direction.

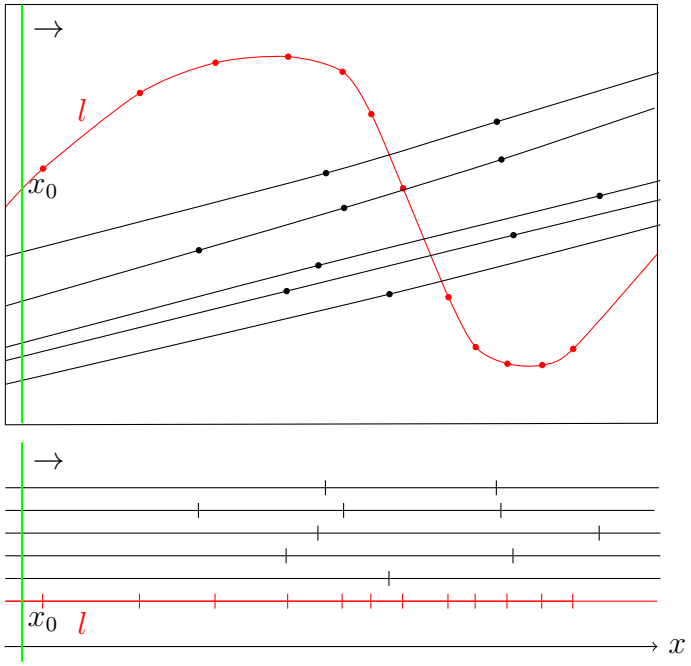


Figure 4.12: Position with no interval starting point on the sweep line (green).

In Table 4.1 we show the time (in seconds) spent in computing the intersection points of a stable and an unstable manifold for a given number of points (the same number in each manifold). We use the manifolds created in the test example `Manifolds2DHom.m`, in `Testruns/InvManifolds` in MatContM. In this code the number of computed points in the manifolds is set by the user. We use 25000 to 350000 points. The computations were done on a Macbook Pro (2GHz Intel Core i5 - 8GB 1867MHz) MATLAB *R*2017*a*. In Table 4.1 the algorithm of Bruschi is compared with the projection algorithm in §4.5.1 and its improved version in §4.5.2. Clearly the new algorithms are much faster than the algorithm of Bruschi. This is also clear from Figure 4.14, where the algorithm of Bruschi is presented in red and the others in blue and green, respectively. In Figure 4.15 we see that the improved version of the projection algorithm (green) is faster than the first projection algorithm (blue).

We note that in the worst case the projection algorithm in both its first and in its improved form has a time complexity $O(n^2)$. Apparently, this does not occur in our situation. Figure 4.15 suggests that in practice the time complexity of the projection algorithms is close to linearity in the number of intervals. In Table 4.2 we give the ratio of the time spent in the computation in the case of n points in each manifold (using the improved algorithm with projection) versus $(2n+k)\log(2n)$ where k is the number of found intersection points. This ratio is practically constant, so in practice the time complexity of the improved projection algorithm is $O((2n+k)\log(2n))$, just like that of the *line sweep* algorithm [19].

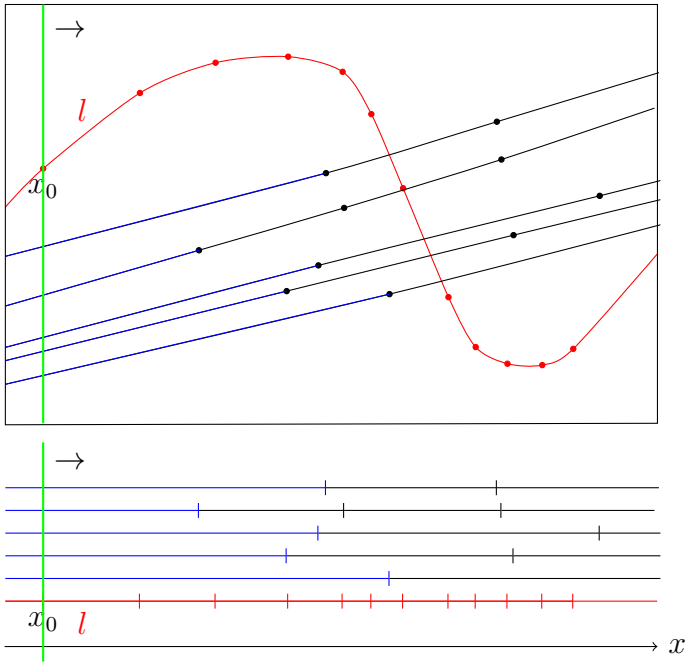


Figure 4.13: The next point met by the line sweep algorithm

4.5.3 Example: homoclinic connections

In the testrun `Manifolds2DHom.m` in the directory `Testruns/InvManifolds` of MatContM we consider the generalized Hénon map (4.10) with parameter values $\alpha = -0.4, \beta = 1.03, R = -0.1, S = 0$ and a saddle fixed point $x_0 = (-1.621146385, -1.621146385)$. In §4.5.1 we computed the stable and unstable manifolds with 20000 points each and plotted them in Figure 4.11.

In §4.5.2 we used the improved projection algorithm to compute the intersection points of the two manifolds. The aim of the procedure is to compute homoclinic connections. For this we use the code provided in Appendix G. This code is based on a code of C. Bruschi [11] which requires that the intersection points are first ordered along the stable manifold in the direction away from the saddle fixed point. The outcome is presented in Figure 4.16, where the manifolds are plotted together with two homoclinic connections (purple and green).

# points n	Bruschi time (s)	Projection time (s)	Projection2 time (s)
25000	5.2035	3.1900	1.7811
50000	18.2562	5.6752	3.8201
75000	40.5224	8.1444	5.9616
100000	69.2368	10.7070	7.9499
150000	151.2542	15.8123	12.0731
200000	266.7075	20.8992	16.4446
250000	413.8600	25.7673	20.5561
300000	596.5867	30.9975	24.5081
350000	831.9529	36.9692	29.8827

Table 4.1: Time spent by the Bruschi algorithm, the projection algorithm and the improved projection algorithm when computing the intersection points of an unstable and a stable manifold both approximated by n points.

4.5.4 Example: heteroclinic connections

To compute heteroclinic connections we consider again the generalized Hénon map (4.10) but now with parameter values $\alpha = 0.3, \beta = -1.057, R = -0.5, S = 0$. We use the testrun `Manifolds2DHet.m` in the directory `Testruns/InvManifolds` of MatContM.

Unlike in the case of homoclinic connections in §4.5.3 we now use the second iterate of the map. We compute the stable manifold rooted in the saddle equilibrium $x_1 = (-0.4286, -0.4286)$ and the unstable manifold rooted at another saddle fixed point $x_0 = (0.4666, 0.4666)$. These manifolds have 15000 points each and are shown in Figure 4.17.

To compute heteroclinic connections we first need the intersection points of the stable and unstable manifolds, which we compute by using the Bruschi algorithm, the projection algorithm in §4.5.1 and the improved projection algorithm in §4.5.2. A survey of the computational results is given in Table 4.3. We see that the improved projection algorithm is the most efficient one. Now the heteroclinic connections can be computed using the code in Appendix G (with itnumber = 2). This code is based on a code of C. Bruschi [11] which we have adapted to the case of an iteration number larger than 1. The heteroclinic connections are presented in Figure 4.18 in green and purple.

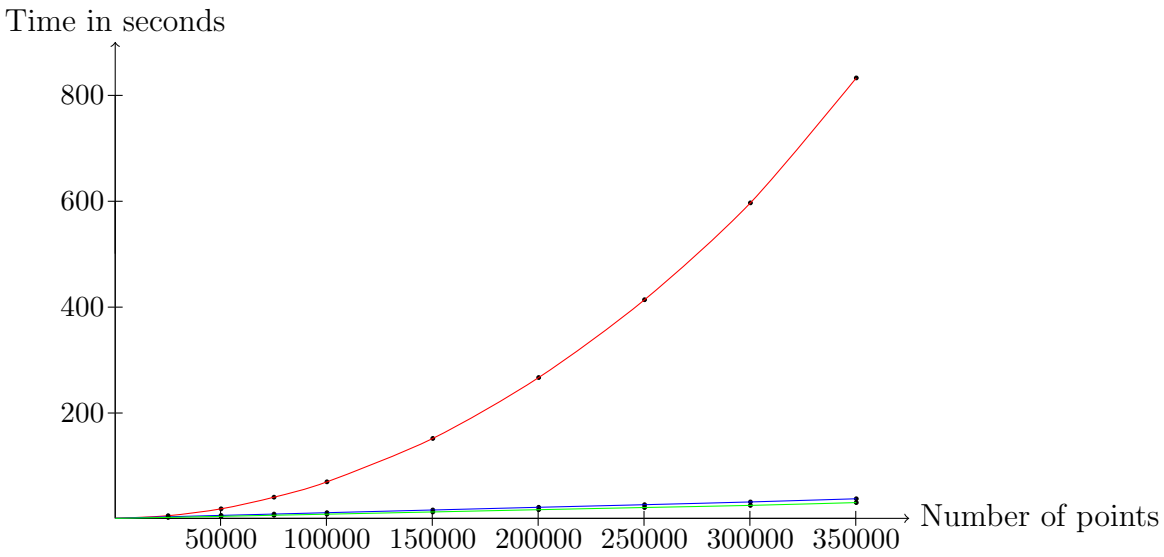


Figure 4.14: Time spent by the algorithm of Bruschi (red), the projection algorithm (blue) and the improved projection algorithm (green) when computing the intersection of a stable and an unstable manifold approximated by the same number of points.

4.6 Initialization of a homoclinic orbit from a one-dimensional manifold

In this section we return to the general case, i.e. with state dimension two or higher. But we restrict to the case where either the unstable or the stable manifold are one-dimensional. This section was published as part of [73]

4.6.1 Initialization from a one-dimensional unstable manifold

Logically, four parts can be distinguished in the initialization algorithm. We discuss them separately.

Step 1: Growing the unstable manifold. We use the algorithm described in §4.4.1 to obtain an unstable one-dimensional manifold.

Step 2: Intersecting the stable eigenspace. If a point on the unstable manifold also lies on the stable manifold of $x_{\pm\infty}$, it is a point of a connecting orbit. However, computing manifolds of dimension higher than one is a difficult problem; it is practically unfeasible if the dimension is higher than two, but see [96] for recent advances. We avoid this difficulty by only considering the stable eigenspace near $x_{\pm\infty}$. If an intersection point of the unstable manifold with the stable eigenspace is close enough to $x_{\pm\infty}$, then it serves as an approximation of an intersection point with the stable manifold. To find intersection points with the stable eigenspace, we consider a left eigenvector v_u^l of the (unique) unstable eigenvalue. The inner product function with v_u^l generically changes sign when the unstable

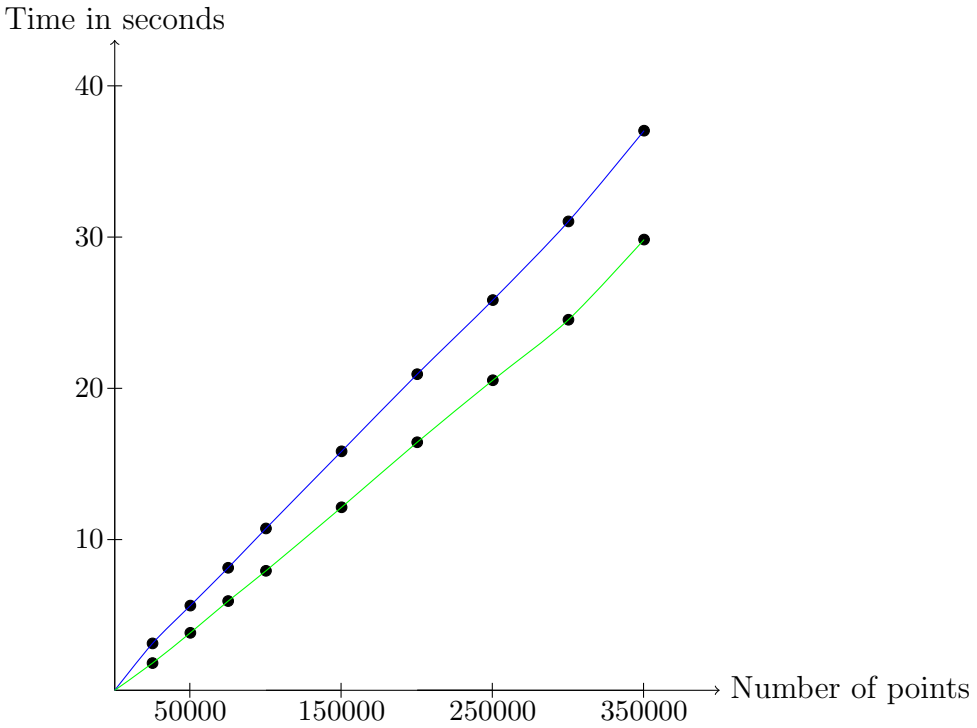


Figure 4.15: Projection algorithm (blue), improved projection algorithm (green)

manifold crosses the stable eigenspace and the exact intersection point is approximated by a linear interpolation in the line segment where the sign change happens.

Step 3: Tracing back the orbit to $x_{-\infty}$. The intersection point P found in **Step 2** is only an approximation to a point on the homoclinic orbit and needs corrections by a Newton method applied to a defining system which is a truncation of (4.2) with projection boundary conditions, as discussed in [54]. This orbit has to be approximated first. We start with the segment information to trace back the orbit:

First, we locate the segment on the piecewise linear approximation that contains the point P. This segment is a line, defined by two points P_1 and P_2. These points were added at some stage by the algorithm. We retrieve two segments, one that contains the preimage of P_1 and one that contains the preimage of P_2. These two segments can be the same, they can be consecutive or there can be several segments between them. The preimage of P lies on one of these segments. Bisection is applied on these segments to find an accurate value for the preimage of P. Once the preimage of P has been located, we repeat the process with the preimage as the new P.

At some stage, we have traced the orbit back to the part of the manifold approximation where points were added along the unstable eigenspace during the initial phase. We can then decide to stop or add extra points. For continuation purposes, we need to get sufficiently close to the fixed point.

Step 4: Adding points towards $x_{+\infty}$. The computation of the unstable manifold is

# points n	Ratio
50000	$7.6319e-5$
100000	$7.4944e-5$
150000	$7.3449e-5$
200000	$7.3366e-5$
250000	$7.2124e-5$
300000	$7.0679e-5$
350000	$7.3023e-5$

Table 4.2: Ratio of the time (s) spent by the improved projection algorithm in the computation of the intersection points of an unstable and a stable manifold approximated each by n points versus $(2n+k)\log(2n)$. k is the number of found intersection points.

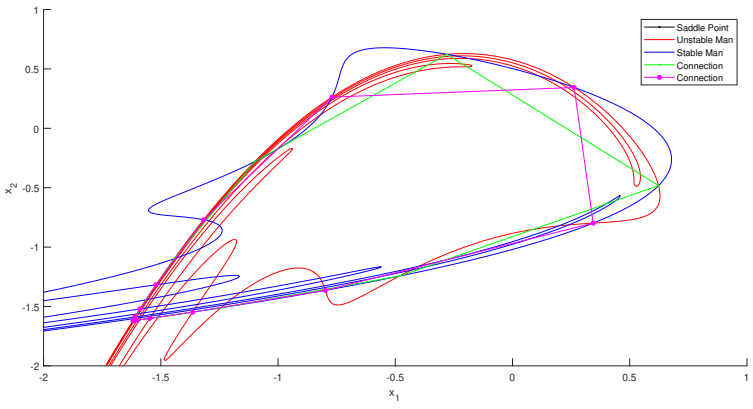


Figure 4.16: Stable (blue) and unstable (red) manifolds of the generalized Hénon map (4.10).

usually quite time-consuming and the computation of each subsequent intersection point with the stable eigenspace requires more effort. At some stage we have to decide that we are close enough to the stable manifold. We then add new points simply by applying the map and then projecting back on the stable eigenspace to avoid being thrown off due to the presence of the unstable direction.

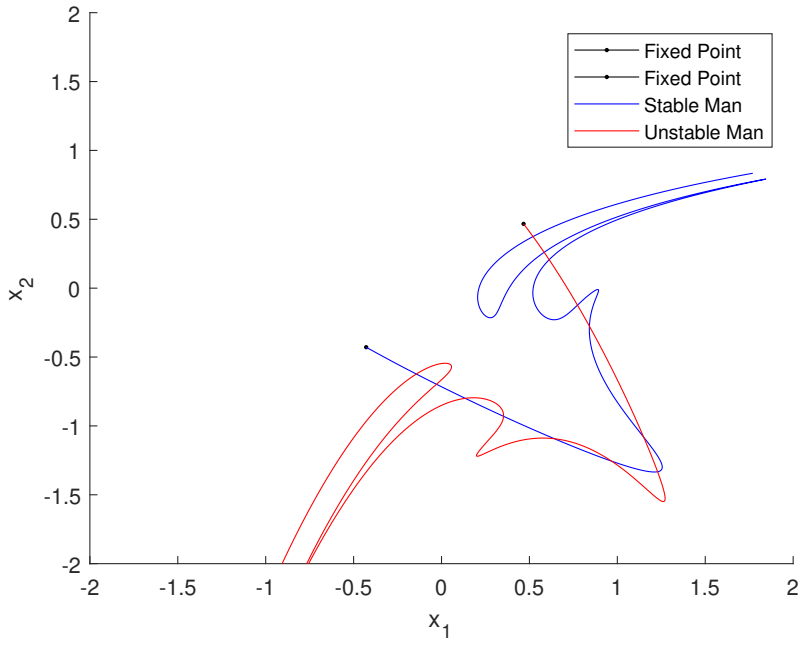


Figure 4.17: Stable (blue) and unstable (red) manifolds of the generalized Hénon map (4.10).

4.6.2 Initialization from a one-dimensional stable manifold

Essentially, this is done like in the case of an unstable manifold. Two other issues have to be taken into account. First, tracing back the orbit is simpler in this case since we can apply the map and then project to the corresponding segments of the already computed stable manifold and choose the closest point. Second, adding points to $x_{-\infty}$ is now done by applying the inverse map (in the form of a Newton iteration procedure) and project to the unstable eigenspace.

4.6.3 Case Study: Adaptive Control Map

We illustrate the algorithm described in in §4.6 in the study of the *adaptive control map* to obtain a rather complete bifurcation diagram of the resonance horn in a 1:5 Neimark-Sacker bifurcation point, revealing new features.

The adaptive control map

$$f : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} y \\ bx + k + zy \\ z - \frac{ky}{c+y^2}(bx + k + zy - 1) \end{pmatrix} \tag{4.11}$$

# points	Bruschi	Projection	Improved Proj
25000	4.5290	2.4828	1.8340
50000	17.4327	4.5864	3.7405
75000	41.9410	6.6198	5.7264
100000	68.4937	8.5738	7.5180
150000	163.4358	12.5669	10.4461
200000	278.2382	16.0458	13.2632
250000	442.3803	20.1670	16.5891
300000	640.9995	24.2855	20.2986
350000	856.5199	28.1474	23.9518

Table 4.3: Time comparison (in seconds) of three algorithms to find the intersection points of a stable and an unstable manifold.

was introduced in [38] and studied in [36]. The parameters are k, b and c. In [36] an example of a subcritical period-5 resonance horn is given. Near the tip of the horn, the authors detect and compute a homoclinic tangency curve. This tangency curve is shown in the (k, b) parameter space, the parameter c is kept at 0.1.

For $(k, b) = (0.824, -0.47)$, we obtain a saddle of $f^{(5)}$ near $(1.56, 0.25, 0.72)^T$ using Newton's method on $f^{(5)}(X) - X = 0$. The saddle values are $x = 1.566650994666793$, $y = 0.253929996002914$, $z = 0.725117829106035$. The multipliers of this saddle are 0.0674, 0.1957 and 1.7483, so we have a one-dimensional unstable manifold and a two-dimensional stable manifold.

We use our algorithm to find a homoclinic orbit. We start this by growing the unstable manifold, following the procedure in §4.4.1. The resulting unstable manifold is shown in Figure 4.19 as the red line that emanates from the saddle, then turns back towards it while showing slight oscillations before converging to a stable fixed point. This indicates that we are nearby a homoclinic connection.

The second step is to intersect the stable eigenspace as described in §4.6.1. The tangent plane is shown in Figure 4.20 as the grey plane. We intersect this plane with the unstable manifold (red) and take the point that is closest to the saddle (green cross). The third step is taking preimages of this point in the unstable manifold (blue crosses in Figure 4.20), as described in §4.6.1. This backward computation of the homoclinic orbit ends when we are close enough to the saddle. The fourth step is adding points at the stable side of the homoclinic orbit. For this we use the method described in §4.6.1. This is shown in Figure 4.21 where we start from the initial intersection point (green cross) and add new points (magenta crosses).

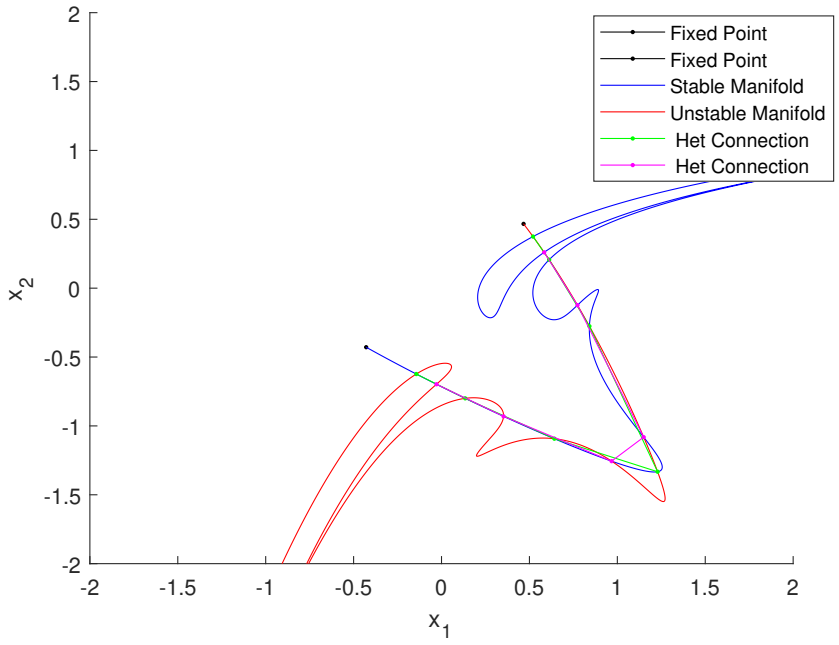


Figure 4.18: Stable (blue) and unstable (red) manifolds of the generalized Hénon map (4.10) with two fixed points and two heteroclinic connections.

This approximation of a homoclinic orbit is a sufficiently accurate starting point for Newton's method applied on the defining system for a homoclinic connection.

Initial Newton corrections are always applied in MatContM when starting a continuation, so we can use this approximation as the initial orbit for a homoclinic continuation. We select k as the free parameter. Figure 4.21 shows the approximation to the homoclinic orbit when our algorithm is applied to the data presented in Figure 4.20 and an orbit obtained after Newton corrections by varying the parameter k, as done by MatContM.

Figure 4.22 shows the correction in the (k, b) parameter space and the continuation of the homoclinic connection by varying the parameter k. This is depicted by a black line emanating from the corrected orbit. We encounter two distinct homoclinic tangencies during the continuation. These tangencies are limit points of the homoclinic connection curve and therefore denoted as LP_HO (Limit Point of a Homoclinic Orbit). The continuation moves back and forth between these two tangencies several times.

From both homoclinic tangencies, we start homoclinic tangency continuations in both directions using k and b as free parameters. The curves of homoclinic tangencies are seen in Figure 4.22 as red curves. Robust homoclinic connections exist between those two red lines. This reveals that our starting point for obtaining an approximation to the homoclinic orbit was poor since there was no homoclinic connection for these parameter values. Choosing a saddle near homoclinic connections can thus suffice to obtain a homoclinic connection

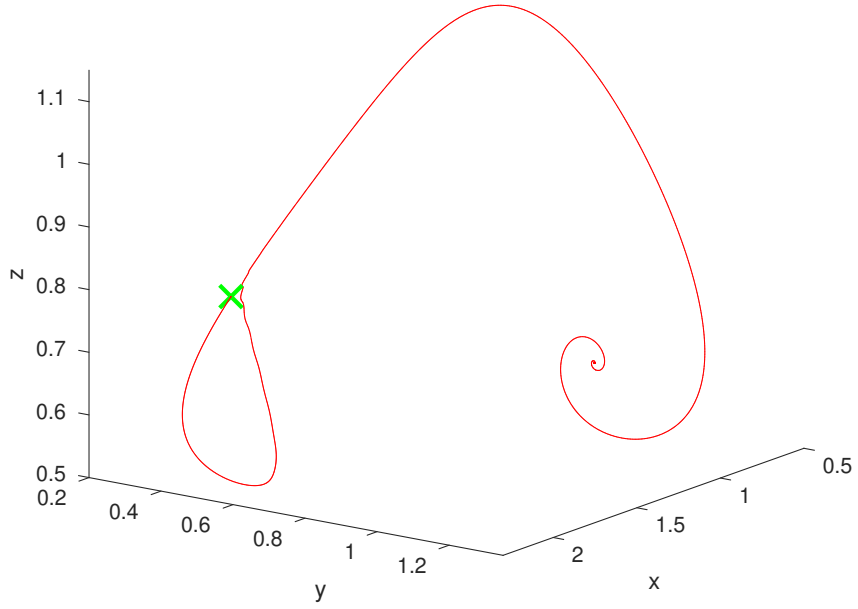


Figure 4.19: The unstable manifold (red curve) of the saddle (green cross) of the fifth iterate of (4.11). The unstable manifold turns back towards the saddle and then away from the saddle towards another fixed point.

for continuation. This is very useful since areas in the parameter space where connecting orbits exist are usually very narrow.

We now provide a more detailed picture filling in gaps in [36]. Figure 4.23 extends Figure 4.22 and shows the full result of both homoclinic tangency continuations. The black curve containing the two tangencies from Figure 4.22 is marked on Figure 4.23 using a black +. In one direction the red tangency curves move towards the upper Resonance 1:1 (R1) point where the continuation process fails if tangency curves are too close. Figure 4.24 shows homoclinic tangencies in phase space on the left red line in Figure 4.22. The blue orbit is the initial homoclinic tangency for $k = 0.82445996, b = -0.47$. The smallest red one is closest to the R1 point.

Figure 4.25 shows the full resonance horn and Figure 4.26 provides a zoom showing a remarkable interplay of local and global bifurcation curves. We have detected a HT_SF bifurcation point (big red star) at $(k, b) = (0.80179, -0.4866)$ and four HT_NS bifurcation points (small magenta stars) at $(k, b) = (0.789729826599800, -0.490045493902434)$, $(0.788197353086524, -0.492122114091179)$, $(0.787017631285026, -0.493787062295342)$, and $(0.786651451614289, -0.494323335473866)$, respectively. These bifurcations and their test-functions are described in §4.7.

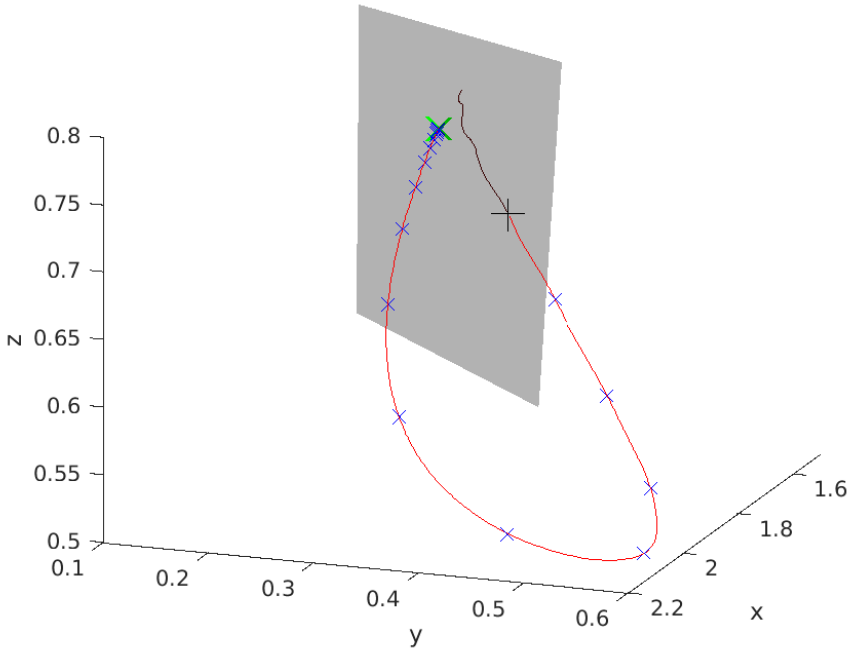


Figure 4.20: The unstable manifold (red line), the stable eigenspace (grey plane), their intersection point closest to the saddle (black +), points of the approximate homoclinic orbit by computing preimages (blue crosses).

In addition, we have determined two other global bifurcation curves $QNS^{\pm}$ that indicate a quasi-periodic saddle-node bifurcation. Here a stable and unstable invariant curve coalesce and disappear. These curves were not obtained through continuation but were constructed using an ad hoc method in order to complete the bifurcation diagram.

Notice that the curve LP_HE^5 extends towards the period-doubling bifurcation curve PD^5 where the period-5 saddle cycle has two unstable and one stable direction (the superscripts refer to the period of the cycle). Indeed the geometry of the attractor seemed to be quite complicated in that area. We did not explore this situation in more detail.

4.7 Detection of bifurcations during homoclinic continuation

In this section we list the bifurcations that MatcontM can detect during the continuation of homoclinic orbits and homoclinic tangencies.

In Figure 4.27 we display the relation graph between a general homoclinic orbit (HO, at the top), some codimension 1 bifurcations (HO_NS, HO_Dx and LP_HO at the second line)

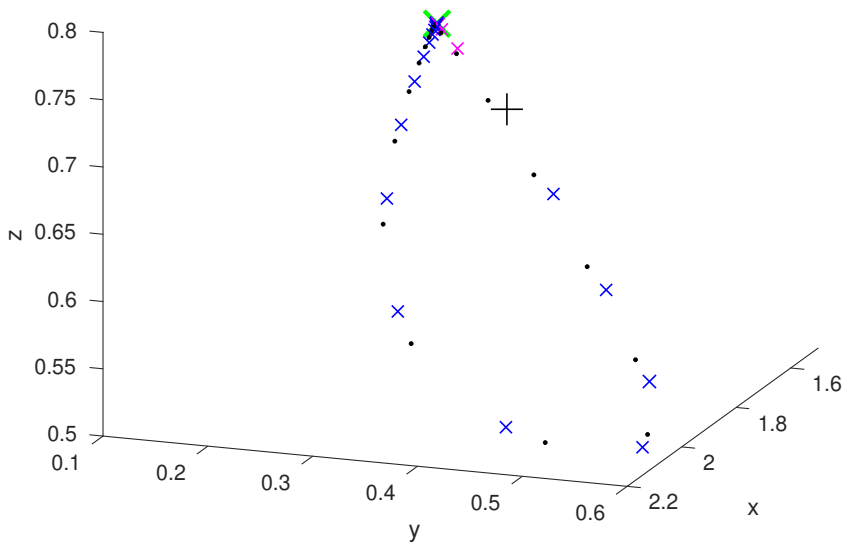


Figure 4.21: The approximation obtained by our algorithm (cf. Figure 4.20) where additional points are added between the intersection point and the saddle (magenta crosses) and a corrected orbit using Newton's method (black dots). This corrected orbit is the starting point in a homoclinic connection continuation.

and codimension 2 bifurcations (HT_NS, HT_Dx, HT_xF and HT_Ex at the bottom line).

In this graph "HO" stands for "Homoclinic Orbit", "HT" for "Homoclinic Tangency", "NS for "Neutral Saddle", "D" for "Double", "LP" for "Limit Point", "E" for "Extended", "F" for "Foliation" and "x" for either "S" (Stable) or "U" (Unstable). We note that a limit point of homoclinic orbits is called a homoclinic tangency, so in this graph "LP_HO" and "HT" are used interchangeably.

The scenario for detecting bifurcations is generic: on a computed curve of bifurcation points with codimension i one detects bifurcations with codimension $i+1$ by monitoring test functions that change sign at the occurrence of such higher codimension points. These test functions are evaluated at each step during the continuation. When a sign change is detected, a bisection-like algorithm is used to locate the higher-codimension bifurcation.

4.7.1 Codimension 1 bifurcations

Let n_s, n_c, n_u be the number of stable, critical and unstable multipliers of the fixed point of a homoclinic orbit, respectively, i.e. those with modulus less than 1, equal to 1 and

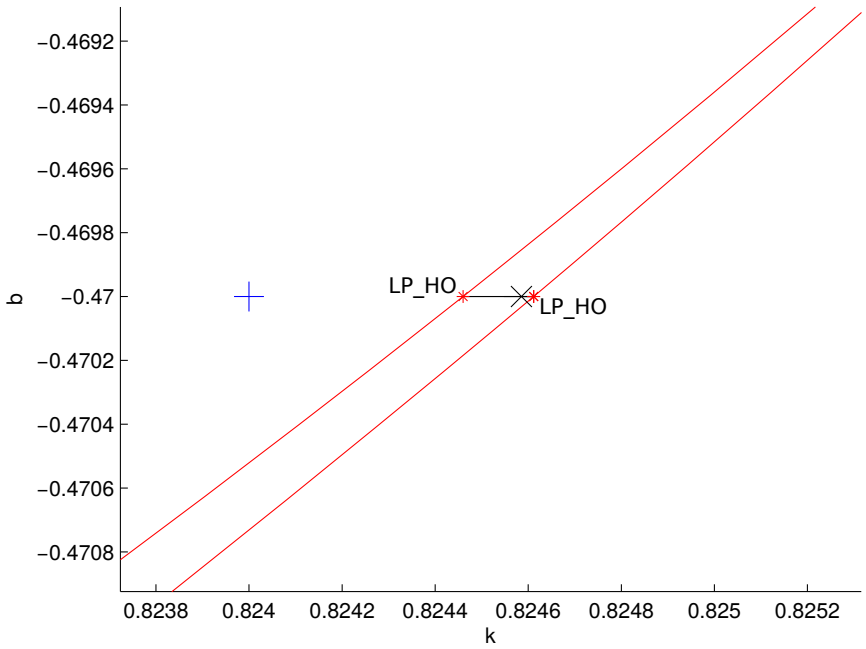


Figure 4.22: Tangencies of homoclinic connections. The initial approximation is represented by the blue +. The corrected orbit is represented by the black cross. The corrected orbit is the first point in a homoclinic connection continuation, the black line. During this continuation, two homoclinic tangencies are detected, labeled by LP_HO (limit point of a homoclinic orbit). The red lines are part of homoclinic tangency continuations started from both LP_HO points. Homoclinic connections exist between those two red curves.

larger than 1, respectively. Also, let $\mu_1^s, \mu_2^s, \ldots, \mu_{n_s}^s$ be the stable multipliers, ordered by decreasing modulus. Similarly, let $\mu_1^u, \mu_2^u, \ldots, \mu_{n_u}^u$ be the unstable multipliers, ordered by increasing modulus. The multipliers μ_1^s and μ_1^u are called the leading stable and unstable multipliers, respectively. Also, let $(x^1, x^2, \ldots, x^N)$ be the computed part of a homoclinic orbit.

Neutral Saddle Homoclinic Orbit HO_NS

This is the case where $n_c = 0$ and $|\mu_1^s \mu_1^u| = 1$. The test function is defined as $\phi_{NS} = |\mu_1^s \mu_1^u| - 1$.

Double-Stable Homoclinic Orbit HO_DS

This is the case where $n_c = 0$ and $\mu_1^s = \mu_2^s$. The test function ϕ_{DS} is defined as $(\mu_1^s - \mu_2^s)^2$ in the case where both μ_1^s and μ_2^s are real, as $-(\Im(\mu_1^s - \mu_2^s))^2$ if they form a conjugate

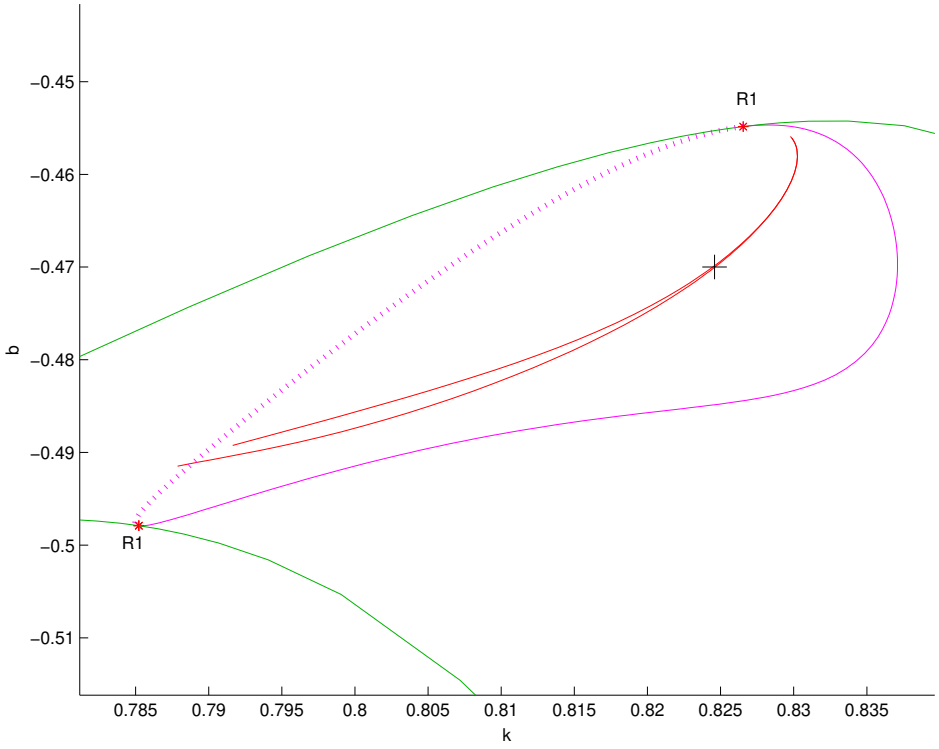


Figure 4.23: Green: limit point curves; red: curves of homoclinic tangencies; magenta: Neimark-Sacker curve; dashed magenta: Neutral Saddle curve; black +: starting homoclinic tangency. The R1 points are 1:1 resonance points of the 5th iterate.

complex pair, and undefined in all other cases.

Double-Unstable Homoclinic Orbit HO_DU

This is the case where $n_c = 0$ and $\mu_1^u = \mu_2^u$. The test function ϕ_{DU} is defined as $(\mu_1^u - \mu_2^u)^2$ in the case where both μ_1^u and μ_2^u are real, as $-(\Im(\mu_1^u - \mu_2^u))^2$ if they form a conjugate complex pair, and undefined in all other cases.

Homoclinic Tangency LP_HO

This is the case of a limit point of homoclinic orbits. The test function is simply the parameter component of the tangent vector to the curve of approximate homoclinic orbits.

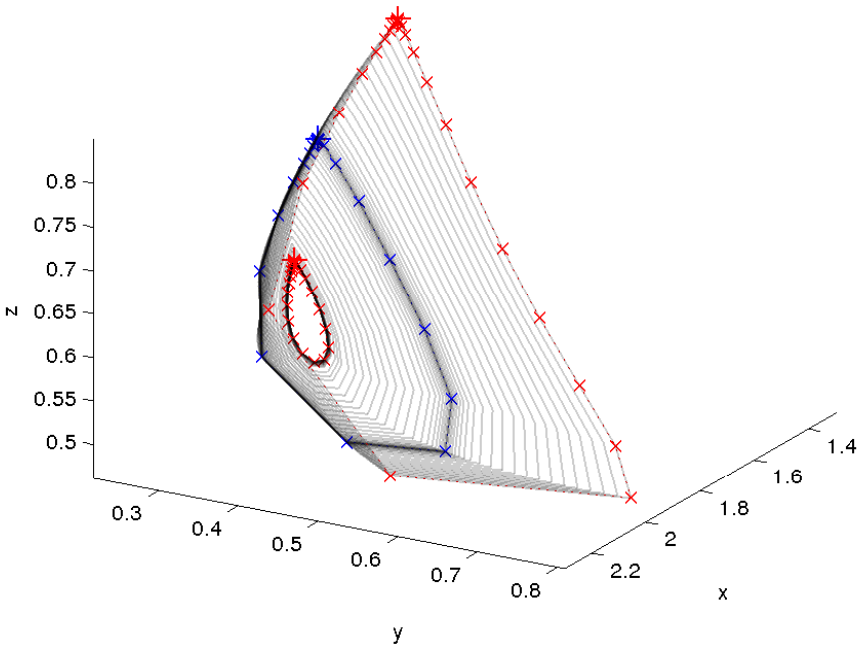


Figure 4.24: Blue orbit: starting homoclinic tangency for $k = 0.82445996, b = -0.47$; smallest red orbit: homoclinic tangency close to R1 for $k = 0.82990325, b = -0.45615685$; largest red orbit: homoclinic tangency far away from R1 for $k = 0.79163754, b = -0.48923588$.

4.7.2 Codimension 2 bifurcations

The following codimension 2 bifurcations occur when in a tangency another condition is satisfied.

Neutral Saddle Homoclinic Tangency HT_NS

This is the case where $n_c = 0$ and $|\mu_1^s \mu_1^u| = 1$. The test function is $\phi_{NS} = |\mu_1^s \mu_1^u| - 1$.

Double-Stable Homoclinic Tangency HT_DS

This is the case where $n_c = 0$ and $\mu_1^s = \mu_2^s$. The test function ϕ_{DS} is defined as $(\mu_1^s - \mu_2^s)^2$ in the case where both μ_1^s and μ_2^s are real, as $-(\Im(\mu_1^s - \mu_2^s))^2$ if they form a conjugate complex pair, and undefined in all other cases.

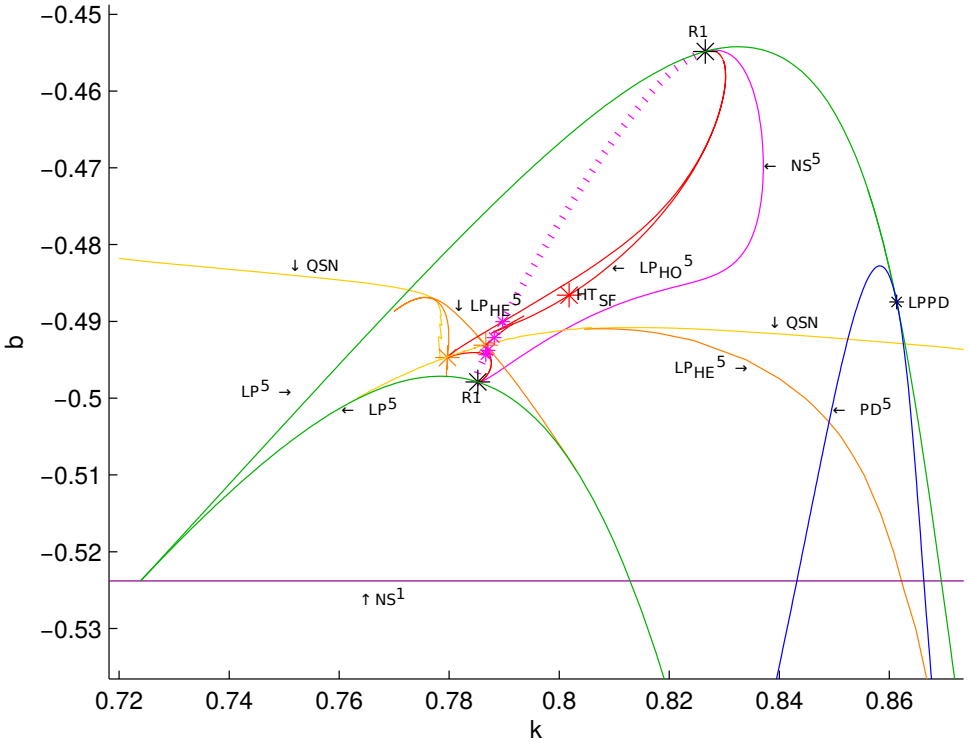


Figure 4.25: The full resonance horn. The dark magenta curve (straight horizontal line) is a NS1 curve, all other curves are bifurcation curves of the 5th iterate. Green=LP5 curves, Magenta solid=NS5 curve, Magenta dashed=neutral saddle fixed points. The magenta curves touch the green curves in R1 points. Red=Homoclinic tangency curves LP_HO, rooted in the R1 points. Orange and yellow: heteroclinic tangencies. Blue=PD5. Note the presence of a HT_SF bifurcation point on the LP_HO curve, cf. Figure 4.27.

Double-Unstable Homoclinic Tangency HT_DU

This is the case where $n_c = 0$ and $\mu_1^u = \mu_2^u$. The test function ϕ_{DU} is defined as $(\mu_1^u - \mu_2^u)^2$ in the case where both μ_1^u and μ_2^u are real, as $-(\Im(\mu_1^u - \mu_2^u))^2$ if they form a conjugate complex pair, and undefined in all other cases.

Stable Foliation Homoclinic Tangency HT_SF

Gonchenko, Gonchenko and Tatjer [40] studied the dynamics due to a generalized homoclinic tangency for maps. This generalized tangency is defined as a tangency of the unstable manifold to the strongly stable foliation of the stable manifold at the homoclinic orbit. Hence, this bifurcation may occur in (at least) three-dimensional maps and we restrict to the case where the unstable manifold is one-dimensional. Then for neighborhoods of the connecting orbit, there exist return maps for each sufficiently large iteration number such that the dynamics on this neighborhood is given by a Generalized Hénon Map. See

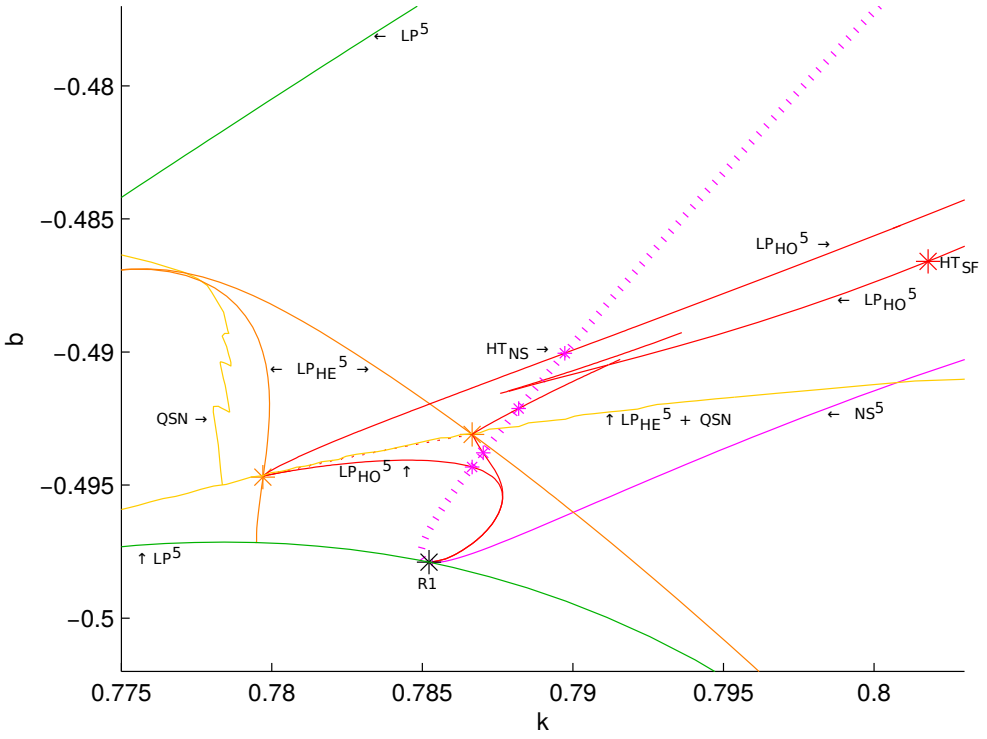


Figure 4.26: Zoom of Figure 4.25. The four magenta crosses on the dashed magenta curve are HT_NS bifurcation points, cf. Figure 4.27. Note that the form of one of the LP_HO curves suggests the nearby presence of a swallowtail bifurcation of homoclinic orbits.

Figure 4.28 for a sketch of the orbit structure.

The construction of the test function ϕ_{SF} is based on the observation that in a generalized homoclinic tangency, the tangent vector to the unstable manifold near the saddle is orthogonal to the leading stable left eigenvector v_s^l which we assume to be real.

The tangent vector along the unstable manifold during continuation of a homoclinic tangency is not available as only the homoclinic orbit and not the full manifold is computed during continuation. We therefore need to approximate the tangent vector of the unstable manifold. We start with the unstable eigenvector v_u at the saddle point. Let the computed internal points of the homoclinic orbit (different from the saddle point) be denoted by $x_1, x_2, \ldots, x_N$ in that order. Also, let $v_u^1, v_u^2, \ldots, v_u^N$ be the corresponding tangent vectors to the unstable manifold. Now as tangent vectors are mapped to tangent vectors by the linearization, i.e. $v_u^{m+1} \approx D_f(x_m)v_u^m$, we can map v_u along the connecting orbit so that the tangent vector at the unstable manifold at any point x_m of the connecting orbit is given by

$$v_u^m = D_f(x^{m-1})D_f(x^{m-2})\cdots D_f(x^1)v_u.$$

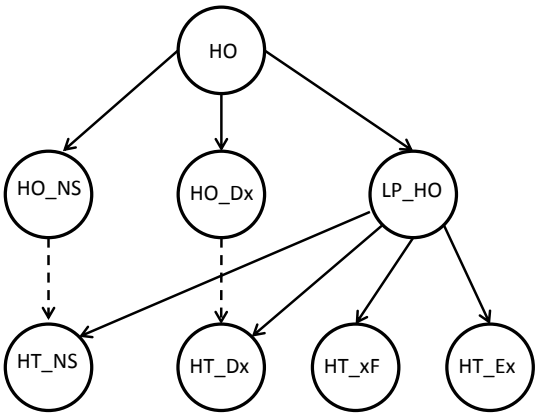


Figure 4.27: Detection graph of homoclinic bifurcations. "x" stands either for S (stable) or U (unstable). Dashed curves indicate that such a bifurcation exists, but its detection is not supported.

Then we can choose $\phi_{SF} = {v_s^l}^T v_u^N$ as a test function.

Extended Stable Homoclinic Tangency HT_ES

In the case of a one-dimensional unstable manifold the unstable extended manifold W^{ue} depicted in Figure 4.29 is a non-unique invariant manifold that contains the unstable manifold and the leading (assumed to be one-dimensional) stable manifold. Its tangent plane at each orbit point x_m is spanned by the tangent vector v_u^m to the unstable manifold (the green arrow in Figure 4.29) and the mapped leading stable eigenvector v_s^m (the red arrow in Figure 4.29) which is assumed to be real. The HT_ES bifurcation occurs if the unstable extended manifold is tangential to the stable manifold. Since v_u^m is already tangential to the stable manifold, the condition is that v_s^m is also tangential to the stable manifold, i.e. orthogonal to the left unstable eigenvector at the saddle point v_u^l. As in the preceding case this bifurcation may occur in (at least) three-dimensional maps. The test function ϕ_{ES} is defined in a dual way to ϕ_{SF} and given by $\phi_{ES} = {v_u^l}^T v_s^N$ where v_s^N is defined as in the previous case, replacing v_u by the leading stable eigenvector v_s.

Unstable Foliation Homoclinic Tangency HT_UF

This generalized tangency is defined as a tangency of the stable manifold to the strongly unstable foliation of the unstable manifold at the homoclinic orbit. Hence, this bifurcation may occur in (at least) three-dimensional maps and we restrict to the case where the stable manifold is one-dimensional and the leading unstable eigenvalue is real and unique. So the tangent vector to the stable manifold near the saddle is orthogonal to the leading unstable left eigenvector v_u^l.

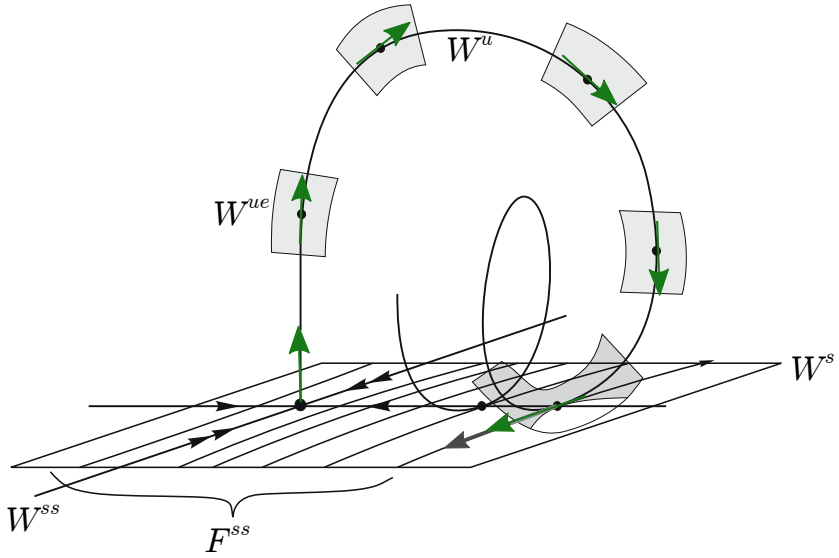


Figure 4.28: An instance of a HT_SF bifurcation. The green arrows are the tangent vectors of the unstable manifold. W^s, W^u, W^{ss} and W^{ue} denote the stable, unstable, strongly stable and unstable-extended manifolds, respectively. F^{ss} is the strongly stable foliation. Upon returning in the stable manifold the tangent vectors are tangential to the strongly stable foliation. The Figure is adapted from [40].

We map v_s along the connecting orbit so that the tangent vector at the stable manifold at x_m is given by

$$v_s^m = D_f^{-1}(x^{m+1})D_f^{-1}(x^{m+2})\cdots D_f^{-1}(x^N)v_s.$$

Then we choose $\phi_{UF} = {v_u^l}^T v_s^1$ as a test function.

Extended Unstable Homoclinic Tangency HT_EU

As in the preceding case this bifurcation may occur in (at least) three-dimensional maps and we restrict to the case where the stable manifold is one-dimensional and the leading unstable eigenvalue is real and unique.

The test function ϕ_{EU} is defined in a dual way to ϕ_{UF} and given by $\phi_{EU} = {v_s^l}^T v_u^1$ where v_u^1 is defined as in the preceding case.

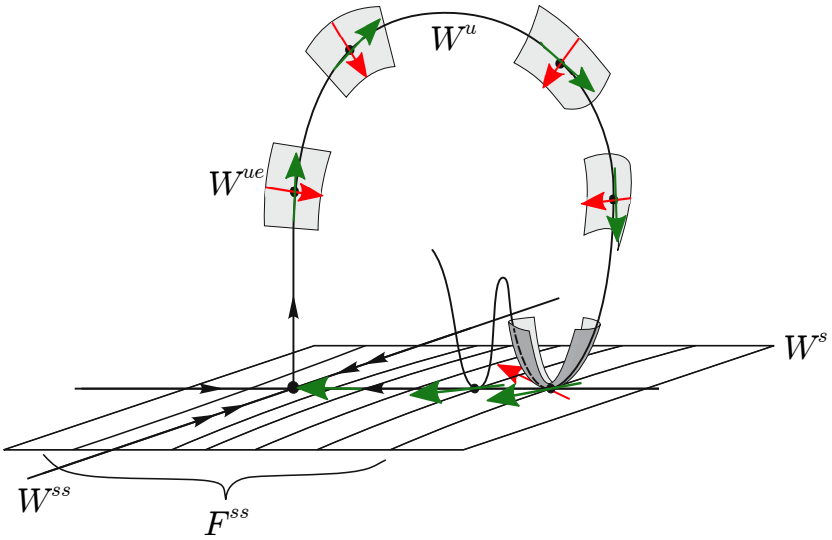


Figure 4.29: An instance of a HT_ES bifurcation. W^s, W^u, W^{ss} and W^{ue} denote the stable, unstable, strongly stable and unstable-extended manifolds, respectively. F^{ss} is the strongly stable foliation. The red arrows are mapped from the leading stable eigenvector at the saddle point. Together with the tangent vectors of the unstable manifold, they span the tangent plane to the W^{ue} manifold. Upon returning in the stable manifold the unstable-extended manifold becomes tangential to the stable manifold. The tangent vector to the unstable manifold (green arrow) converges to a vector in the leading stable direction. The Figure is adapted from [40].

Chapter 5

Front end features of the new MatCont GUI.

5.1 The MatCont database

The database of MatCont consists of an archive of systems one of which is the *current system*.

A system is internally characterized by a system m-file, a system mat-file and a system directory, all with the name of the system. They are all in the directory **Systems** of MatCont.

The system m-file is the *odefile* of the system which is described extensively in §3.6 and in the manual [45].

The mat-file of the system contains the information that is necessary to restart the computations on a previously studied system at the stage where it was left, including the position and contents of all windows. However, computed data have to be redrawn.

The system directory has at least one default subdirectory called `Diagram`. This and other subdirectories of the system directory are called *diagrams*.

Each diagram contains a number of mat-files and each mat-file describes a computed curve with enough information to reproduce the computation exactly if desired.

Each computed curve contains a number of special points, which includes the first point, the last point, bifurcation points, and zeros of userfunctions but other entities may also be defined as special points. An important example of this is the case of an orbit where a **Select Cycle** object is identified as a special point. Selecting this special point as an initial point opens a subpanel with two fields that allow to choose a convergence criterion and a number of mesh intervals for initializing a curve of periodic orbits that starts from the periodic orbit found by time integration (the number of collocation points is always the default 4, cf. §5.4.1).

The MatCont GUI provides a special tool called the **Data Browser** to navigate through this database. The **Data Browser** can be accessed through the main MatCont panel but also from the **Output** window of a computed curve.

5.2 The main MatCont panel

Figure 5.1 is a screenshot of a typical MatCont main panel.

The MatCont GUI has a tab bench (line at the top of the screen) with six tabs, which each correspond to a panel. The panels are named as follows:

1. Tab 1: Select
2. Tab 2: Type
3. Tab 3: Window/Output
4. Tab 4: Compute
5. Tab 5: Options.
6. Tab 6: Help

The main panel is vertically divided in three fields, namely the *Class field,* the *Current System field* and the *Current Curve field.*

The *Current System field* displays the name of the system that is loaded (if any) and also which derivatives are available symbolically (S) or have to be approximated by finite differences (N). More precisely, a string of five letters from {S,R,N} is displayed. If $i \in \{1,2,3,4,5\}$ then the derivatives of order i are obtained from symbolic derivatives if the $i-$th letter is "S", from a user-provided script if the letter is "R", and by finite difference approximations if the letter is "N". Normally the string only contains "S" and "N" with the "S" preceding the "N". Derivatives of order higher than 3 are only needed if codim 2 bifurcations are involved.

The `Current Curve` field displays the name of the current curve, i.e. the curve that is loaded (if any), the name of the diagram in which it is kept, the `Initial Point` type (if any), the `Curve` type (if any) and the name of the initializer that was used or will be used to start the computation of the curve.

The main panel can be used from the tab bench. For quick access to the **Data Browser** one can also right-click on either the `Current System` field or the `Current Curve` field. In the last two cases location-dependent menus are opened that allow to manage the diagrams and curves of the system. In the `Current System` case one can choose between `Load New System, Create New System, Edit Current System,` and `ManageUserfunctions.` In the `Current Curve` case the choice is between `View Curve, View Diagram, Rename Curve, New Diagram, Continuer/Integrator,` and `Starter.` Renaming a curve allows to preserve it permanently in spite of the restriction on the number of untitled curves, cf. the use of the Archive Filter button in §2.9.3.

Figure 5.2 is a screenshot of a MatCont main panel where `Select/System` is pressed. The meaning of the further tabs `New`, `Load/Edit/Delete Systems` and `Manage Userfunctions` is obvious.

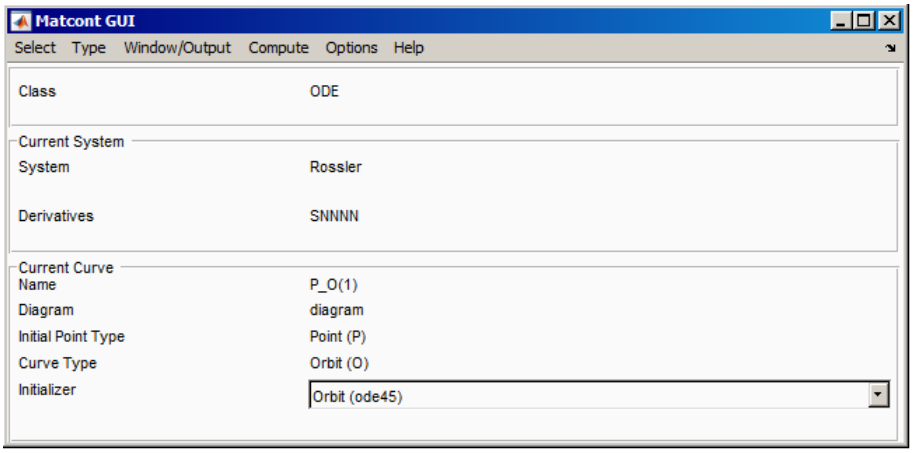


Figure 5.1: The main MATCONT panel.

Clicking **Select|Diagram** opens the **Data Browser** with a panel that shows all diagrams. Further tabs allow to `Load, Rename,` or `Delete` an existing diagram or to create a `New` one.

Clicking **Select|Curve** opens the `Data Browser` with a panel that shows all curves in the current diagram. Further tabs allow to `Load, Rename,` or `Delete` an existing curve or to create a `New` (empty) one. There is also a `View` tab that opens a spreadsheet-type view of the data in the current curve. If a curve is loaded, then the first point is automatically selected as Initial Point.

Clicking `Select|Initial Point` opens the `Data Browser` with a panel that shows all special points in the current curve (if no curve is loaded, then it is not possible to use this functionality.) Further tabs allow to see the settings with which the curve was computed, to get a spreadsheet view of the data in the current curve and to select one of the special points to be used as initial point for a new computation.

Clicking **Select|Organize Diagrams** opens a tool that allows to move curves from one diagram to another one, but also to view, rename, plot and delete curves, see Figure 5.3.

Clicking **Select|Exit** closes the MATCONT session. The main panel disappears. Depending on the operating system it may be necessary to close other windows separately.

Clicking **Type|Initial Point** opens a list box window with the list of allowed Initial Point types. By selecting one of these the user declares that the loaded Initial Point has that type, see Figure 5.4.

Clicking **Type|Curve** opens a list box window with the list of Curve types which are allowed for the type of the loaded Initial Point. By selecting one of these the user declares that a curve of that type is to be computed (but it does not start the computation), see Figure 5.5.

Clicking **Window/Output** allows to open graphic windows (2D or 3D), a **Numeric** window, a **Starter** window or a **Continuer/Integrator** window, see Figure 5.6. Multiple

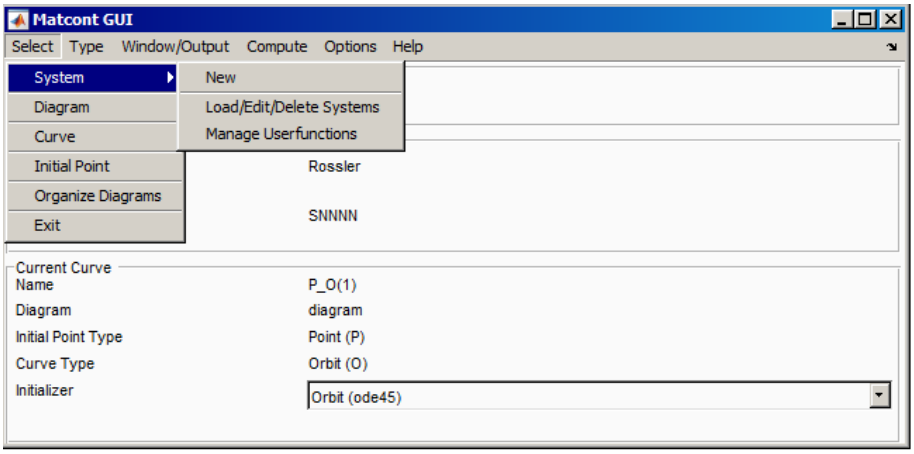


Figure 5.2: Clicking **Select|System** in the main MATCONT panel.

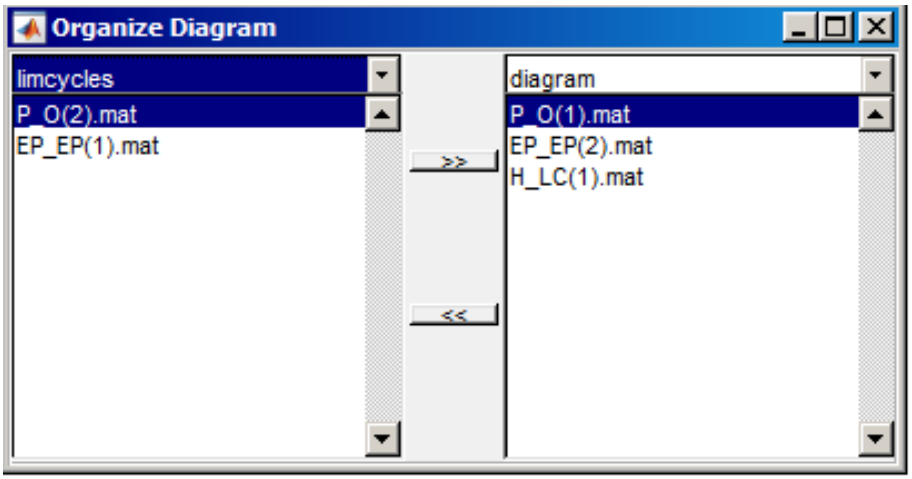


Figure 5.3: Clicking **Select|Organize Diagrams** in the main MATCONT panel.

graphic windows can be open at the same time, but there cannot be more than one **Numeric** window, **Starter** window or **Continuer/Integrator** window. We note also that **Continuer** and **Integrator** are mutually exclusive. In fact, they are internally the same window.

The **Compute** tab opens a subpanel that allows either to start a continuation **Forward** or **Backward** or to **Extend** a finished continuation or integration, see Figure 5.7.

The **Options** tab opens the subpanel displayed in Figure 5.8. It allows to set the seven options already mentioned in §2.9.3. More details are given in §5.7.

The **Help** tab opens a subpanel that allows to consult the MATCONT documentation or to unlock the GUI-session in an emergency situation, see Figure 5.9.

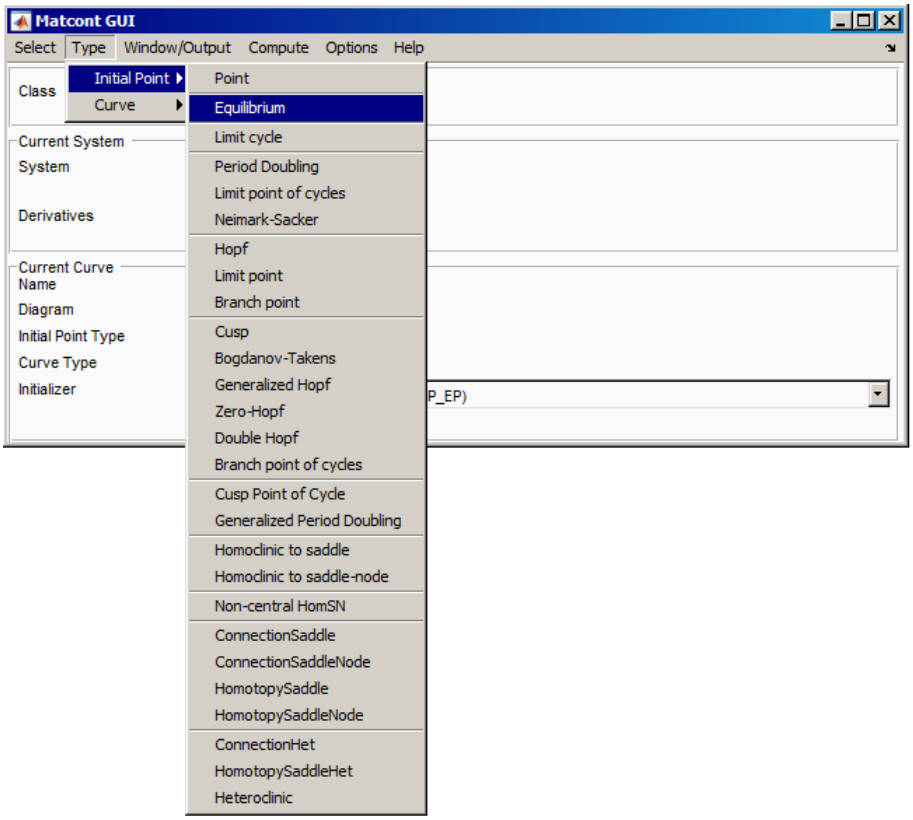


Figure 5.4: Clicking **Type|Initial Point|Equilibrium** in the main MatCont panel.

5.3 The matfiles of curves

In the case of a continuation curve, such a mat-file contains the following data:

- x, v, s, h, f, i.e. the standard output of the continuation run, cf. §3.5.1 and [45],§3.2. This information allows to export the in MatCont computed results to the general MATLAB environment, e.g. to use them in other software environments, sophisticated graphical packages etcetera.
- A structure `globals` with two fields. One field is always `cds`, the *continuation descriptor structure* which contains most of the settings used in the computation of the curve, cf. [45], §3.8. The other substructure is more specific and depends on the type of the computed curve, so it can be one of `eds, lpds, hds, bpfd, lds, homds,` or `hetds`. We refer to [45] §3.2 for more details.
- A character string called `initUsed` which contains the name of the initializer that was used in the continuation, e.g.

  ```
  'cont_init_EP_EP'
  ```

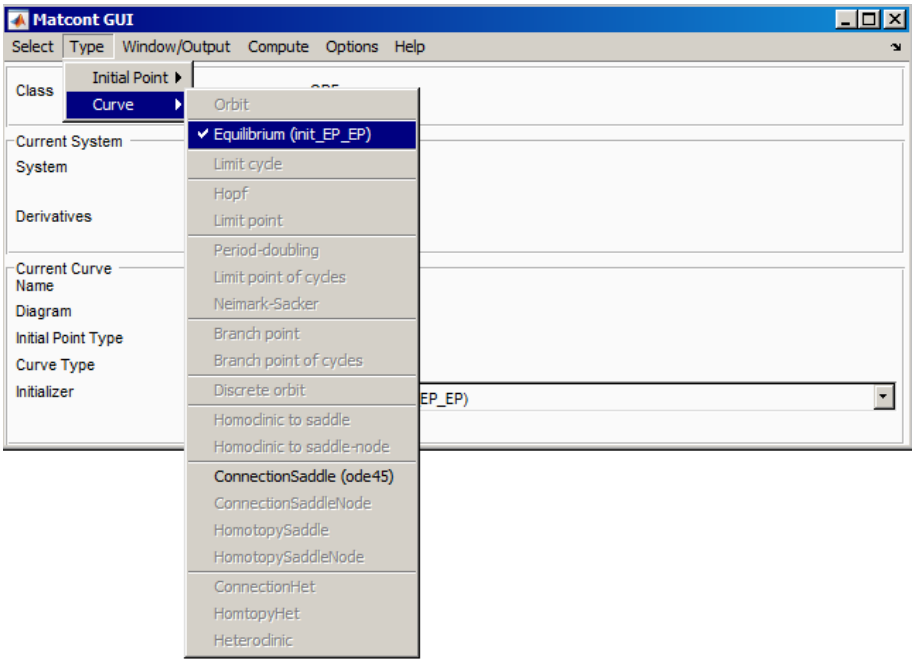


Figure 5.5: Clicking **Type|Curve|Equilibrium (init_EP_EP)** in the main MATCONT panel.

in the case of an equilibrium curve started from an equilibrium point.

- A structure `gui` that contains information for the GUI. It contains three fields, namely `CLSettings, compbranch` and `loader.` This information is not typically useful for the MATCONT user.

In the case of an orbit, the following is stored:

- `connectData`: this field is used to store additional information needed to perform the homotopy method.
- `method`: name of the integrator, e.g. `ode45`.
- `param`: the row vector of parameters
- `x0`: the row vector of the initial point
- `gui`: a structure that contains metadata utilized only by the GUI
- t: the column vector of time points where the orbit was computed (not including the *Event* values if any)
- `tspan`: the time interval

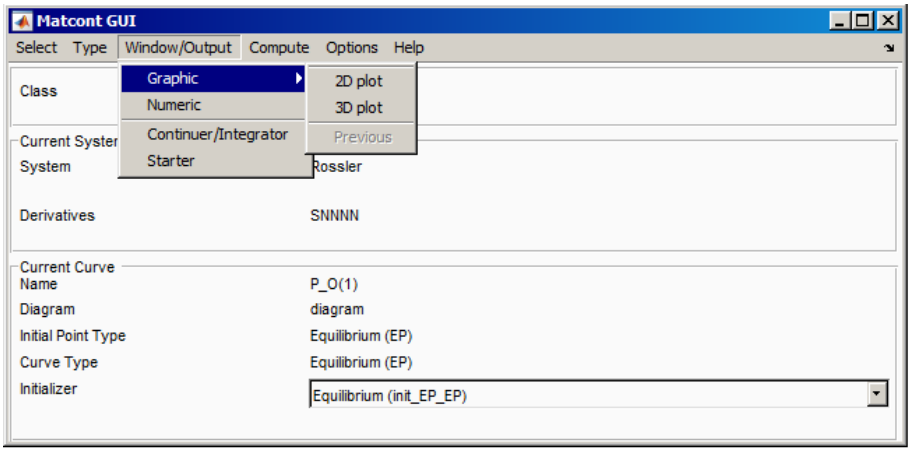


Figure 5.6: Clicking **Window/Output|Graphic** in the main MatCont panel.

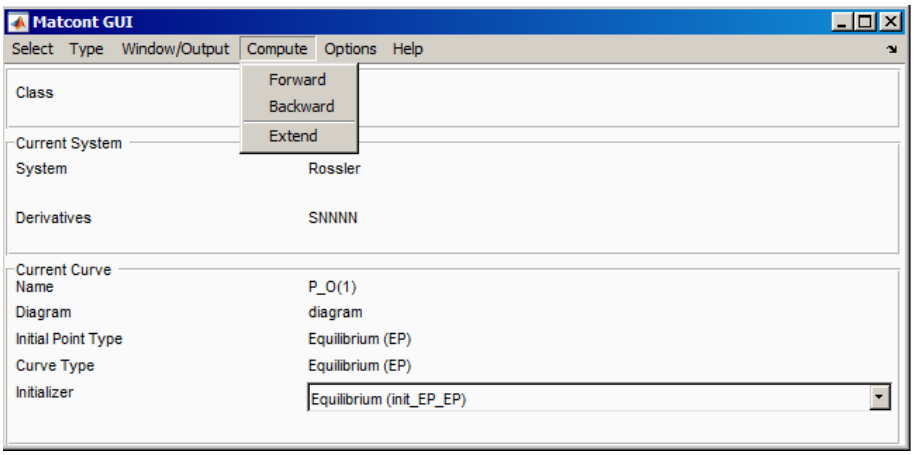


Figure 5.7: Clicking **Compute** in the main MatCont panel.

- y: the matrix whose rows are the state values at the time points.
- `options`: the options used in the time integration, e.g. *AbsTol*, *RelTol*, name of the *Event* function if any, *Normcontrol* and *Refine*.
- `tE`: if an *Event* function was present, then `tE` is the column vector of time points where events were detected. Otherwise `tE` is empty.
- `yE`: if an *Event* function was present, then `yE` is the matrix whose rows are the state values at the time points in `tE`.
- `iE`: if an *Event* function was present then `iE` is the column vector of the indices of the events detected at the time points present in `tE`.

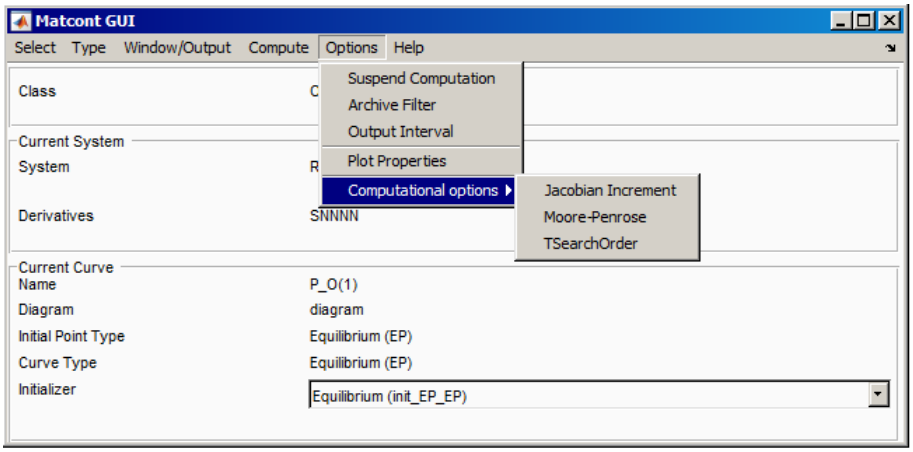


Figure 5.8: Clicking **Options|Computational Options** in the main MATCONT panel.

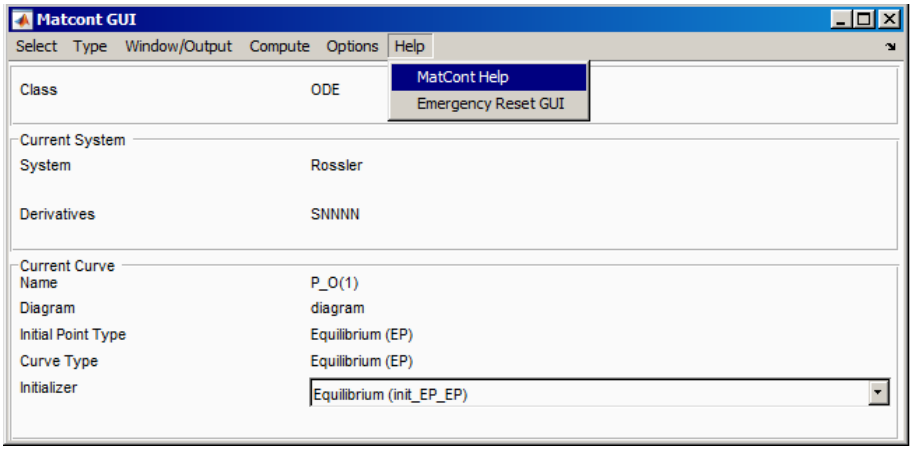


Figure 5.9: Clicking **Help** in the main MATCONT panel.

We note that userfunctions can be plotted during time integration but the data are not stored in the mat-file.

5.4 Input and Control panels

As a general remark, the CONTINUER and INTEGRATOR windows cannot be visible at the same time. Opening one automatically closes the other one (in fact, internally they are the same window).

5.4.1 Detailed description of the panels

1. Continuer. This panel is open only if continuation is to be performed. It has a fixed structure and always consists of the following numeric input fields which correspond to a subset of the settings which can be used in MatCont:

1. Continuation Data
   - InitStepsize
   - MinStepsize
   - MaxStepsize
2. Corrector Data
   - MaxNewtonIters
   - MaxCorrIters
   - MaxTestIters
   - VarTolerance
   - FunTolerance
   - TestTolerance
   - Adapt
3. Stop Data
   - MaxNumPoints
   - ClosedCurve

2. Integrator. This panel is opened only if integration is to be performed. It has a fixed structure and always consists of the following input fields:

1. Method (choice between standard MATLAB integrators and two additional integrators `ode78.m` and `ode87.m`)
2. Interval
3. EventFunction (is only enabled if Events are to be monitored during time integration; this functionality is typically used to compute Poincaré sections)
4. InitStepsize
5. MaxStepsize
6. RelTolerance
7. AbsTolerance

8. Refine
9. Normcontrol

Details on these choices can be found in the MATLAB help page (`doc odeset`). In particular, in the case of `ode45`, the default for `Refine` is 4. This means that only every fourth step in the output is a real Runge-Kutta step. The default for `Refine` for all other solvers is 1.

3. Starter. This subpanel is opened if either integration or continuation is to be performed. Its structure depends on the curve type but in the continuation case also on the dimension of the state space since for each bifurcation type a sufficient number of state variables is needed.

- In the case of time integration the initial values of `time` and of the state variables and parameters have to be introduced in that order.
- In the case of an equilibrium continuation from init_EP_EP the **Starter** panel consists of the following input fields:
  1. State variables
  2. Parameters; exactly one parameter must be chosen as the active one, in all other cases an error message is issued when the continuation is started.
  3. Monitor Singularities, i.e. indicate which of the three bifurcations "Branching", "Hopf", "Limit Point" are to be monitored. "Branching" and "Limit Point" require only one state variable, "Hopf" requires two.
  4. Calculate eigenvalues, i.e. indicate whether the eigenvalues of the computed equilibrium points are to be computed and stored in the output of the continuation
- In the case of a fold (Limit Point) continuation from init_LP_LP the **Starter** panel consists of the following input fields:
  1. State variables
  2. Parameters; exactly two parameters must be chosen as the active ones, in all other cases an error message is issued when the continuation is started.
  3. Monitor Singularities, i.e. indicate which of the three bifurcations "Bogdanov-Takens", "Zero-Hopf", "Cusp" are to be monitored "Cusp" requires one state variable, "Bogdanov-Takens" requires two, "Zero-Hopf" requires three.
  4. Calculate eigenvalues, i.e. indicate whether the eigenvalues of the computed fold points are to be computed and stored in the output of the continuation

  Note. In this Starter there are two columns of radio boxes for the system parameters. To activate a parameter one clicks in the right column. The left column is optional. If a parameter in this column is activated then branching points with respect to this parameter will be detected along the curve.

- In the case of a Hopf continuation from init_H_H the **Starter** panel consists of the following input fields:

  1. State variables
  2. Parameters; exactly two parameters must be chosen as the active ones, in all other cases an error message is issued when the continuation is started.
  3. Monitor Singularities, i.e. indicate which of the four bifurcations "Bogdanov-Takens", "Zero-Hopf", "Generalized Hopf", "Double Hopf" are to be monitored. "Bogdanov-Takens" and "Generalized Hopf" require two state variables, "Zero-Hopf" requires three, "Double Hopf" requires four.
  4. Calculate eigenvalues, i.e. indicate whether the eigenvalues of the computed Hopf points are to be computed and stored in the output of the continuation

- In the case of a limit cycle continuation from init_H_LC the **Starter** panel consists of the following input fields:

  1. State variables.
  2. Parameters; exactly one parameter and the Period must be chosen as active, or two parameters. In all other cases an error message is issued when the continuation is started.
  3. Period; cannot be given an initial value and no value is displayed. But it has be chosen as active if only one system parameter is active (this is the natural situation).
  4. Switch Data, i.e. 'amplitude', a measure for the amplitude of the initial limit cycle.
  5. Discretization Data, i.e. the number *ntst* of test intervals used in the discretization and the number *ncol* of collocation points, which is also the degree of the piecewise polynomials that are used to approximate the limit cycle.
  6. Monitor Singularities, i.e. indicate which of the four bifurcations "Branching", "Period Doubling", "Limit Point of Cycles" and "Neimark-Sacker" are to be monitored. "Branching", and "Limit Point of Cycles" require two state variables, "Neimark-Sacker" and "Period Doubling" require 3.
  7. Calculate multipliers, i.e. indicate whether the multipliers of the computed limit cycles are to be computed and stored in the output of the continuation.
  8. Phase response curve. Requires three input fields, i.e. a field *PRC* with entry either 0 or 1, a field *dPRC* with entry either 0 or 1, and a field *Input* with entry either a scalar or a column vector with a scalar value for each state variable.

- In the case of a limit cycle continuation from init_LC_LC the **Starter** panel consists of the following input fields:

1. Parameters; either two parameters must be active, or one parameter and the Period. All other cases will generate an error message when the continuation is started.
2. Period; cannot be given an initial value but has to be active if only one parameter is active (this is the natural situation).
3. Discretization Data, i.e. the number *ntst* of test intervals used in the discretization and the number *ncol* of collocation points, which is also the degree of the piecewise polynomials that are used to approximate the limit cycle.
4. Monitor Singularities, i.e. indicate which of the four bifurcations "Branching", "Period Doubling", "Limit Point of Cycles" and "Neimark-Sacker" are to be monitored. "Branching", and "Limit Point of Cycles" require two state variables, "Neimark-Sacker" and "Period Doubling" require 3. Default is none.
5. Calculate multipliers, i.e. indicate whether the multipliers of the computed limit cycles are to be computed and stored in the output of the continuation.
6. Phase response curve. Will require three input fields, i.e. a field *PRC* with entry either 0 or 1, a field *dPRC* with entry either 0 or 1, and a field *Input* with entry either a scalar or a column vector with a scalar value for each state variable.

- In the case of a limit cycle continuation from init_PD_LC the **Starter** panel consists of the following input fields:

1. Parameters; one or two parameters must be set to be active. In all other cases an error message is issued when the continuation is started.
2. Period; cannot be given an initial value but has to be active if only one system parameter is active (this is the most natural situation).
3. Discretization Data, i.e. the number *ntst* of test intervals used in the discretization and the number *ncol* of collocation points, which is also the degree of the piecewise polynomials that are used to approximate the limit cycle.
4. Monitor Singularities, i.e. indicate which of the four bifurcations "Branching", "Period Doubling", "Limit Point of Cycles" and "Neimark-Sacker" are to be monitored. "Branching", and "Limit Point of Cycles" require two state variables, "Neimark-Sacker" and "Period Doubling" require 3. Default is none.
5. Switch Data, i.e. 'amplitude', a measure for the distance from the PD to the initial (period doubled) LC.
6. Calculate multipliers, i.e. indicate whether the multipliers of the computed limit cycles are to be computed and stored in the output of the continuation.
7. Phase response curve. Will require three input fields, i.e. a field *PRC* with entry either 0 or 1, a field *dPRC* with entry either 0 or 1, and a field *Input* with entry either a scalar or a column vector with a scalar value for each state variable.

- In the case of a limit cycle continuation from init_BPC_LC the **Starter** panel consists of the following input fields:

  1. Parameters; one or two parameters must be set to be active. In all other cases an error message is issued when the continuation is started.
  2. Period; cannot be given an initial value but has to be active if only one system parameter is active (this is the most natural situation).
  3. Discretization Data, i.e. the number *ntst* of test intervals used in the discretization and the number *ncol* of collocation points, which is also the degree of the piecewise polynomials that are used to approximate the limit cycle.
  4. Monitor Singularities, i.e. indicate which of the four bifurcations "Branching", "Period Doubling", "Limit Point of Cycles" and "Neimark-Sacker" are to be monitored. "Branching", and "Limit Point of Cycles" require two state variables, "Neimark-Sacker" and "Period Doubling" require 3.
  5. Switch Data, i.e. 'amplitude', a measure for the distance from the BPC to the initial LC on the new branch.
  6. Calculate multipliers, i.e. indicate whether the multipliers of the computed limit cycles are to be computed and stored in the output of the continuation.
  7. Phase response curve. Will require three input fields, i.e. a field *PRC* with entry either 0 or 1, a field *dPRC* with entry either 0 or 1, and a field *Input* with entry either a scalar or a column vector with the dimension of the state space.

Starter windows for other continuation curves are structured along the same lines and are self-explaining by now. In the case of a curve of limit point of cycles the same **Note** applies as in the case of a limit point of equilibria.

5.5 Output panels

The essential results computed by MatCont are stored in its database and can afterwards be accessed by its **Data Browser**. During the computations the results can be output through the following windows:

1. **2D Plot** windows (any number).
2. **3D Plot** windows (any number).
3. A **Numeric** window.
4. An **Output** window (is automatically opened during all continuation and integration runs).
5. The MATLAB command window.

5.5.1 The case of continuation curves

For continuation curves the **2D plot**, the **3D plot** and the **Numeric** windows can draw or display the values of:

- State variables (not in the numerical window for periodic orbits)
- Parameters
- Period (for periodic orbits and codim 1 bifurcations of periodic orbits only)
- Testfunctions
- Userfunctions
- Eigenvalues, in the following cases:
  - Equilibria
  - Homoclinic orbits, of the equilibria associated to the orbits.
  - Heteroclinic orbits, of the endpoint equilibria (different colors).
- Multipliers (for periodic orbits and codim 1 bifurcations of periodic orbits only)
- Current stepsize
- Point number.

If in a **2D plot** the $x-$axis displays a parameter or a point number and the $y-$ axis displays a state variable, then for periodic orbits it is possible to display the minimum and maximum values of the state variable along the orbit.

5.5.2 The case of orbits

For orbits, the **2D plot**, the **3D plot** and the **Numeric** windows can draw, as desired by the user, the values of:

- Time
- State values
- Parameters (although constant during simulation)

5.6 Event functions and Poincaré maps

A Poincaré section is a (in general, curved) surface in phase space that cuts across the flow of a dynamical system. The Poincaré map transforms the Poincaré section onto itself by relating two consecutive intersection points. We note that only those intersection points count, which come from the same side of the section. In this way, a Poincaré map turns a continuous-time dynamical system into a discrete-time one. If the Poincaré section is carefully chosen no information is lost concerning the qualitative behaviour of the dynamics. For example, if the system is being attracted to a limit cycle, one observes dots converging to a fixed point in the Poincaré section.

When computing an orbit $(t, y(t))$ in MATLAB an event can be defined as going through a zero of a scalar event function $G(t, y)$. If G does not explicitly depend on time in an autonomous dynamical system, this functionality can be used to see the Poincaré map in action.

For the sake of generality MATLAB allows to define vector-valued event functions whereby each component function defines its own event. This is done by setting the Events property to a handle to a function, e.g. `@events` with the syntax

[value,isterminal,direction] = events(t,y,varargin)

where y is a state vector. The input variable `varargin` is a cell array that contains the values of the parameters. If parameters are explicitly used in the definition of the event function, then `varargin` should be replaced by an explicit list of parameter names.

If there are k event functions then for $i \in \{1, \ldots, k\}$:

- `value(i)` is the value of the $i-$th event function.
- `isterminal(i)` $= 1$ if the integration is to terminate at a zero of the $i-$th event function and 0 otherwise.
- `direction(i)` $= 0$ if all zeros of the $i-$th component are to be computed (the default), $+1$ if only the zeros are needed where the event function increases, and -1 if only the zeros are needed where the event function decreases.

The use of event functions to compute Poincaré maps in Cl_MatCont is discussed in the manual [45]. We now illustrate the GUI implementation by example. Consider the *Rössler* system:

$$\begin{cases} \dot{x} &= -y - z \\ \dot{y} &= x + Ay \\ \dot{z} &= Bx - Cz + xz, \end{cases}$$

where (x, y, z) are the phase variables, and (A, B, C) are the parameters (this system is also studied in Tutorial I in Appendix A)

Suppose that we want to compute an orbit and detect two events along it, namely $x = 0.2$ and $y = 0.3$ We need only the events where x, respectively y, are increasing and the integration will not be terminated if an event is detected

We then define an event function testEV as follows:

```
function [value,isterminal,direction]= testEV(t,y,varargin)
value=[y(1)-0.2;y(2)-0.3];
isterminal=zeros(2,1);
direction=ones(2,1);
end
```

The function `testEV.m` is placed in the MATCONT main directory (or anywhere else on the MATCONT path)

We integrate the *Rössler* system from 0 to 100 starting from the point $[-5; 5; 10]$ with parameter values $(0.25, 0.4, 4.5)$ and input the name 'testEV' (without quotes) in the EventFunction field of the **Integrator,** see Figure 5.10.

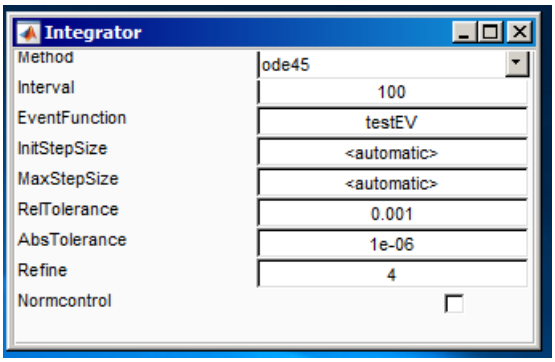


Figure 5.10: Integrator with an Event function.

We also open a MATCONT **Plot3D** window to display the state variables in the range $\{-9 \leq x \leq 9, -9 \leq y \leq 9, -2 \leq z \leq 11\}$. The 3D output is shown in Figure 5.11. We note that zeros of the first event function are marked as 'E1', those of the second as 'E2' etcetera.

To see the convergence of the Poincaré iterates to a fixed point open a **Plot2D** window with $\{4.22 \leq x \leq 4.32, 2.3 \leq z \leq 2.8\}$, see Figure 5.12.

The computed output is stored in the mat-file of the computed curve, see §5.3, from where it can be recovered for further use. Event points can also be selected as special points in the data browser and by double-clicking on the labels in the plots.

5.7 Options window

This panel was mentioned in §5.2 and its typical appearance on the screen is shown in Figure 5.8. The seven options were briefly discussed in §2.9.3. It is a modal window with seven buttons, namely **Suspend Computation, Archive Filter, Output Interval, Plot properties, Jacobian Increment, Moore-Penrose** and **TSearchorder**. This

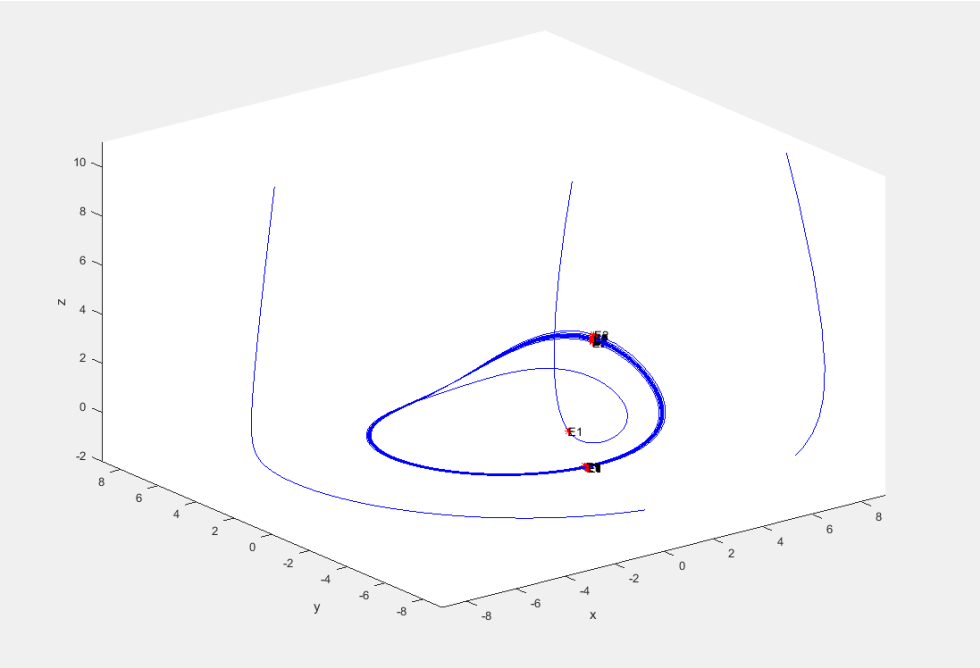


Figure 5.11: Time integration with event points that converge to fixed points.

panel is normally closed and cannot be opened when computations are being done. When it is open, no actions in other windows can be performed. Here we give some more details on the four non-computational options.

Clicking the **Suspend Computation** button opens a list box selection panel which offers the choice of three options to suspend computations, namely at each point, at special points (default) or never. The user then has the possibility to stop and change to a different type of computation, or just to think before proceeding. This functionality applies only to continuation routines.

Clicking the **Archive Filter** button opens a panel with a numerical input window. In this field a positive integer number can be input and the default is 2. It means that only that number of untitled curves of a particular type is preserved. After that number the oldest curve of that type is removed and replaced by a new curve. To preserve a computed curve "permanently", it is necessary to rename it, cf. §5.5.

Clicking the **Output Interval** button opens a panel with a numerical input window. In this field a positive integer number can be input and the default is 1. It means that only after that number of computed points the output to graphical or numerical windows is generated. This functionality applies to continuation curves and orbits. Since output is a very time-consuming part of most computations, this functionality can be used to speed up computations.

Clicking the **Plot Properties** button opens an edit box panel **Plot properties** which allows to assign plot properties (color, linestyle, ...) to each type of computed curve.

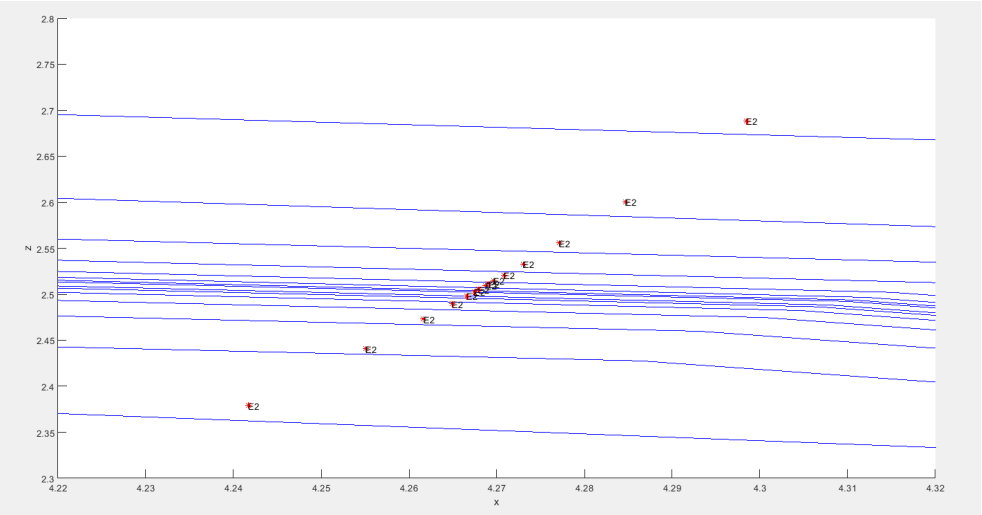


Figure 5.12: Convergence of Poincaré iterates to a fixed point.

Defaults are provided for all types of computed curves, which can be overwritten using MATLAB code in the edit boxes. For curves of equilibria and curves of limit cycles, it is possible to differentiate between stable and unstable parts of the curves by assigning different colors and/or linestyles. Advanced MATLAB users can edit the file *GUICurveModifications.m* to add new differentiations of the plot properties within a curve.

Clicking the Enter key on the keyboard closes the **Options** window and its subwindows.

5.8 The tutorials

Four tutorials are available to introduce the users to practical work with MatCont. For consistency and ease they treat essentially the same material as similar tutorials for the previous versions of MatCont, first written by Yu. A. Kuznetsov and later adapted many times by himself and others. For the new MatCont the practical details are again different.

Tutorial I (Appendix A) deals with the input of a new system, time integration, selecting initial data, 3D plots, starting a computation, inspecting a computed curve in the **Data Browser** and in a spreadsheet form, qualitative changes of orbits under parameter variation, exporting a figure, 2D plots and archiving solutions.

Tutorial II (Appendix B) deals with the location of equilibria by time integration, selection special points on a computed curve via the **Data Browser**, computing a curve of equilibria, using the **Numeric** window, detecting fold and Hopf points and interpreting their normal form coefficients, detecting branch points and continuing a new branch of equilibria, introduction and use of userfunctions.

Tutorial III (Appendix C) deals with the initialization of limit cycles by time integra-

tion, starting a limit cycle continuation from a Hopf point, detecting a period doubling bifurcation, detecting a limit point of cycles bifurcation and a Neimark-Sacker (torus) bifurcation, modulated oscillations (movement on a stable torus), detecting a homoclinic orbit.

Tutorial IV (Appendix D) deals with the continuation of fold and Hopf points under variation of two parameters, two-parameter bifurcation diagrams, continuation of a limit point of cycles curve by starting from a Generalized Hopf (Bautin) point and continuation of a Neimark-Sacker (torus) curve under variation of two parameters.

Chapter 6

Internal working of the new MatCont GUI

In this chapter we discuss the inner workings of the MatCont GUI. We focus on the core mechanism of MatCont, i.e the inner layer between CL_MatCont and the GUI as seen by the user. Important other parts, which can be used semi-autonomously are not discussed here. By this we mean the generator of the system m-files (*SysGUI.m*), the spreadsheet viewers (*GUISimCurveTable.m* and *GUIContCurveTable.m*) and the GUI subsystems *Data Browser* and *Diagram Organizer*, which are stored in subfolders of the *GUI* folder.

The internal documentation of all files is accessed by performing '`doc filename.m`', a reference page is generated based on the comments in the source files. The main driver files are `matcont.m`, `GUI/MATCONTGUI.m` and `GUI/Session.m`.

6.1 Object oriented programming in MATLAB and MatCont

By default, in MATLAB objects are passed along by value (if used as an argument, a copy is made). However, MATLAB contains a superclass called handle with the special property that an object is passed by reference if the class of that object inherits from handle. This is not to be confused with the function handles which are used in Cl_MatCont, cf. [45]. More information can be found in Chapter 5 of the MATLAB OOP manual [67]: *Value or Handle Class – Which to Use.*

In programming for GUI's it is common to inherit from graphical user interface classes (which may be related to, e.g. a specific button) to add a specific functionality. Another technique is 'encapsulation' which is used extensively in the MatCont GUI. This means that a MatCont object can contain as fields one or more MATLAB GUI objects, i.e. objects from the GUI library of MATLAB. We often use `varargin` in the class constructor of a MatCont object to pass along extra arguments to the internal MATLAB GUI object.

The new MATLAB OOP language [67] makes it very easy to use certain popular design patterns for graphical user interface programming, like Model-View-Controller (MVC). Sometimes the controller and view are merged into one.

MATLAB has already implemented a base model, including the adding of listeners (observers) and dispatching of events (signals). This base model is provided in the class handle and inheriting from it. Events declarations are included in the language as a clause:

```
classdef Session < handle  %inherits from handle
...
events
    settingsChanged
end
...
```

One can then add listeners to an object:

```
session.addlistener('settingsChanged',@(obj,event) myfunction(obj,event));
```

One can trigger an event by using:

```
sessionobj.notify('settingsChanged');
```

The `addlistener` command generates an object and should be captured if the user wants to undo the listener at later times. One major difference between the MATLAB OOP language and other OOP languages such as C++ and Java is that the object itself is always passed as an argument to its own methods. This style is also used in the Python programming language.

Consider the following function header of a method.

```
function changeSystem(obj, system)
```

The variable `obj` is the object on which the method is invoked. Invoking the method can be done in two ways:

```
>> changeSystem(session, system);
>> session.changeSystem(system);
```

We use the syntax of the second line, because it resembles the syntax used in Java and C++. In the next section we will describe the main classes in MatCont.

6.2 Classes in MatCont

The four most important classes in MatCont are

- `Session.m` (Session)

  This class maintains the state of the GUI and is considered the 'driver' of the GUI. We discuss this class in more detail in §6.4.

- `CLSettings.m` (Settings)

  An object of this class contains a collection of settings (objects of `CLSetting.m`) required to start a certain computation. We discuss this class in §6.3 and §6.5.

- `CompConf.m` (Computation Configuration)

  This class represents the computations performed in the GUI. An object of this class contains a listing of the required settings and the code for executing the computation. Simulations and continuations are coded in separate class files with `CompConf.m` as base class. An overview of all computations is given in Appendix I.

- `CompSolution.m` (Computation Solution)

  Each object of this class is the outcome of a computation. It will be referred to as a *Solution object* in §6.3.

6.3 General description of the workflow

In MatCont every action involves a **System,** i.e. a dynamical system with state variables and parameters, coded in a *system m-file*.

A typical action starts with a *settings object* containing an `Initial Point` and various other settings most of which are visible in the **Continuer/Integrator** or **Starter** windows. A full list is given in Appendix H.

Depending on the type of the initial point a list of *computation objects* is available. These computations are usually either continuations or simulations. The computation can be chosen by the user; the first one on the list is selected by default (typically, it is the most natural or most often used choice). In Appendix I we provide a full list of all computations.

The computation requires starting conditions to be fulfilled by the user. For example, an equilibrium continuation requires the options in the **Continuer** to be available but it also requires an initial point with parameters, a setting whether eigenvalues need to be computed and it also requires a selection of testfunctions to be monitored. Internally, this selection of testfunctions is derived from the list of bifurcations that the user has selected for detection. Other continuations like `init_BP_EP` require an additional setting called amplitude, a positive double to quantify the distance from the initial point to the first point on the new curve.

The computation object configures the settings object and checks if all necessary settings are present and satisfy the input restrictions. Settings that are already used by

previous computations are re-used. If the settings are not present, then they are created with default values. Once the settings are configured by the computation object, the **Continuer/Integrator** and **Starter** windows are rendered and the user can again configure the settings object.

The computation object has three available actions referred to as **Forward, Backward** and **Extend**. When the action is selected the computation object gathers all relevant settings. In the case of continuations the relevant settings are passed to the initializer and the initializer is called. Then the options structure is built and passed to the continuer.

The file *ContConf.m* contains the base class for all continuation computations and contains the core code (`compute`) for generically processing all continuations. We now display the lines of code of the `compute` function which is the most important function for continuation.

```
function [solution, errormsg, overwrite] = compute(obj, settings, forward)
    solution = [];  %default return value
    overwrite = 0; %default return value
    settings.setValue('forward', forward);
    parametersmodel = settings.getSetting('parameters');
    activeParams = parametersmodel.getActive();
    system = settings.system;
    x0 = settings.coord;
    x0 = x0(:); %make column vector
    param = settings.parameters;
    param = param(:); %make column vector
    ...
```

The first lines retrieve settings (active parameters, current system, initial point, parameter values) from the `settings` object that was passed along. The direction of continuation is stored in `settings` (forward or backward).

```
    ...
    %restore global variables of source curve
    initialPoint = settings.getSetting('IP');
    if ~isempty(initialPoint.source)
       initialPoint.source.restoreGlobals();
    end
    ...
```

If the selected point came from a previous continuation curve, the global variables associated with that curve are restored before the continuation starts (`cds`, `eds`, ...).

```
    ...
    [valid, errormsg] = obj.sanityCheck(system.handle, x0, param, ...
                                        activeParams, settings);
```

```
if ~valid; return; end
[x0, v0, errormsg] = obj.performInit(system.handle, x0, param, ...
                                     activeParams, settings);
if isempty(x0)
    if isempty(errormsg); errormsg = 'init failed'; end
    return;
end
...
```

A sanity check is performed on the current state before any computation starts. Errors might include a wrong number of active parameters or missing data to start a branch. This check can differ depending on the type of continuation computation. After the sanity check, the initializer is called. This initializer can also report an error and shut down the computation.

```
...
options = contset;
options.ActiveParams = activeParams;
options = obj.constructContOptions(options, settings);

[x, v, s, h, f] = obj.contfunction(obj.curvedefinition, x0, v0, options);
if isempty(x); errormsg = ''; return; end

s = obj.postProcess(s);
solution = ContCurve(settings, obj, x, v, s, h, f);
```

After the initializer is called, `contset` is called to provide a new clean options structure for continuation. The active parameter selection is placed in this new structure and a function is called to place the relevant settings from `settings` to this structure. The continuer function `cont.m` gets called and the result is stored in a solution object for continuation (`ContCurve`).

In the case of time integration another version of `compute` is called from the file *SimConf.m* which contains the base class for all simulation (time integration) computations. We now display the lines of code of the `compute` function which is the most important function for time integration.

```
function [solution, errmsg, overwrite] = compute(obj, settings, forward)
    %set default:
    solution = []; errmsg = []; overwrite = 0;
    settings.setValue('forward', forward);

    [tspan, optionsODE] = obj.buildOptions(settings);
```

```
    system = settings.system;
    handles = system.handle();
    x0 = settings.coord;
    x0 = x0(:); %make column vector
    param = num2cell(settings.parameters);
    method = str2func(obj.methodname);
```

After storing the forward setting, the relevant settings are collected into a MATLAB ODE options structure. The initial point and parameter values are extracted from the settings object.

```
    if isempty( optionsODE.Events )
        [t, y] = method(handles{2}, tspan, x0, optionsODE, param{:});
        tE = []; yE = [];
    else
        [t, y, tE, yE, iE] = method(handles{2}, tspan, x0,...
                                              optionsODE, param{:});
    end
    solution = SimCompSolution(settings, obj, t, y, tE, yE, ...
                    method, tspan, x0', optionsODE, cell2mat(param));
end
```

The selected integrator method is called. If a MATLAB *Event* function is present, the call to the method has to be done slightly differently. The simulation results are stored in a solution object for time integration (`SimCurve`).

In all cases the results of the computation are stored in a *Solution object*. The Solution object is then displayed as the current curve in the main MatCont panel.

The user now sees a newly computed current curve. Usually there are special points displayed in the plots, numeric window or data browser which the user can select. On selection a new settings object is generated which inherits the settings of the Solution object but is configured for the type of the new initial point. Typically the initial point is changed and matches the type of the detected point. The initial point data (coordinates and parameters) also match the coordinates and parameter values of the detected point. The transition to a new Settings object is internally called a *switch*.

At this point the workflow cycle is closed. The new settings replace the old settings and a new list of computations becomes available.

6.4 The Session class

In §6.3 we described the general workflow of the core-mechanism in the MatCont GUI. The class Session in `Session.m` implements and executes this mechanism.

This class has a property named `settings` which stores the current settings used in the workflow. The selected computation (usually simulation or continuation) is stored in the `computation` property of Session. The current solution, after computation, is stored in a property called `solutionhandle` (class name: `CLSolutionHandler`) which contains a reference to the current solution and the current diagram.

We will now describe some of the inner workings of the sessions object. We will start by describing the *Events* that are triggered by this object.

- settingsChanged:
  This event is activated whenever a new settings object is loaded as current settings object or whenever a settings object gets reconfigured. For example, whenever a new computation is selected the settings object will be reconfigured in order to fulfill the demands of this computation. A settingsChanged event is fired which causes the `Starter` window and the `Continuer` window to reconfigure.

- settingChanged:
  This event is activated whenever a setting within a settings object gets a new value assigned. This applies to the current settings object. Whenever a new settings object is loaded, the `settingslistener` property in the class Session gets updated so it will react to the newly loaded settings object.

- computationChanged:
  This event is activated whenever a new computation is selected. For example, this event is activated when the user selects a simulation or a continuation branch (in fact, an initializer) based on the type of initial point.

- solutionChanged:
  This event is activated whenever a computation of a new solution is finished or an old solution is loaded. When this event is triggered the main window will update the labels. The menu item to 'Extend' the current solution must also listen to this event to verify if a solution is available for extension.

- initpointChanged:
  This event is activated whenever the user changes the type of the initial point. It is actually a settingChanged-event but due to many components that are dependent on the type of the initial point this special event was introduced.

- lockChanged:
  During computation, most of the GUI components remain locked so that no adjustments can be made by the user. A lockChanged-event notifies those components of changes in the lock-status of the session.

- shutdown:
  This event is activated when the method `shutdownGUI` is called and is used by the various GUI objects to perform a cleanup and shut down. This event occurs when the user closes the main window.

We already discussed the properties `settingslistener`, `settings`, `computation` and `solutionhandle`. Other properties of the Session objects are

- The property `globalsettings` contains the settings that are independent of the current system and computations, they contain the settings for the windows and plot properties.

- The property `systemspath` contains the path to the directory `Systems` which contains the systems and the computed data. This property is set during the startup of the GUI. The user is allowed to change the current directory of the workspace because the GUI uses the `systempath` property to locate the systems and the data. However, the user should never move the source directory while running the GUI.

- The Boolean property `locked` keeps track if the GUI is locked during computation. Other GUI components are notified of changes to this variable through the lockChanged event.

- The property `pointloader` contains a CLPointLoader object. The MATCONT GUI allows for the user to double-click on plots and select special points to load in as the new initial point. However, problems occur when a plot exists but the solution that is depicted gets renamed or deleted from the hard-drive. Solutions are often renamed after computation and this should not lead to a crash whenever an existing plot is used to select a new initial point by double-clicking. Some plots are generated during computation when the solution does not yet exist on the hard-drive. The `pointloader` solves these problems by maintaining an internal name for each (to be) computed solution. This internal name is passed to the plots. Whenever the user interacts with the plot and loads a new initial point, the `pointloader` is called with the internal name. The `pointloader` will map the internal name onto the real name of the solution on the hard-drive. If a solution gets deleted or renamed, the `pointloader` gets informed and adjusts its mappings of the labels.

Because Session is the main object in the GUI, the class-definition file contains a lot of code. To break up the code into multiple files and to keep this important object maintainable, a number of features are delegated to 'submodels'. Each submodel specializes in a certain task maintained by the session object. The three submodels are `branchmanager`, `windowmanager` and `outputmanager`.

- The property `branchmanager` contains a CLBranchManager object. This objects keeps track of the available computations depending on the selected initial point type. The available computations are visible in the main window. Whenever the session triggers an initpointChanged event, the `branchmanager` object will iterate over all possible computations and select those that are compatible with the initial point. The list of available computations gets sorted based on their priority and the computation with the highest priority gets automatically loaded as the initial computation of the

new initial point. The `branchmanager` can activate two events. The newlist event is activated whenever a new list of computations is generated based on an initial point change. A selectionChanged event is activated when the selected computation has been changed (by the user).

- The property `windowmanager` contains a GUIWindowManager object. This object serves as a submodel for the session model. This object stores and restores the positions of the windows, keeps track of windows that have been opened and makes sure no window is opened twice. The windowmanager can be called upon to close all windows or restore certain windows. The windows managed by this object are the main window, the Numeric window, the Starter and Continuer/Integrator windows, the `plotlayout` windows, the diagram organizer window, the plot configuration window and the (d)PRC windows.
- The property `outputmanager` contains a GUIOutputManager object. This object maintains the plot windows and the numeric window. Plot windows and numeric windows are generated by this object and get placed in a list of windows that require computation-output. Before a computation gets started, the list is called upon to prepare itself for computational output and this list is then passed along to the computation.

The Session object has several methods; we mention only the most important ones. Methods that are not mentioned are the usual getter/setter methods and methods for obtaining the current status of the session object.

- `changeState(obj, system, diagram, solution, newsettings)`

  This function changes the state of the Session object and is called upon mainly by the *Data Browser*. The session object will adopt the new system, new diagram, new solution or new settings. Arguments are left empty if no change is needed. For example, by leaving `solution` and `newsettings` empty, one can load in a new system with a given diagram using a blank solution and a default settings object.

- `function changeInitPoint(obj, point)`

  This function changes the initial point in the current settings objects and triggers an initpointChanged event. This causes the `branchmanager` to recompile a new list of available computations.

- `function selectCompConf(obj, compconf)`

  This function changes the computation (simulation or continuation) and triggers computationChanged event.

- `function computeSolution(obj, action)`

  When called upon, this function will use the current computation with the current settings to compute a new solution. This fires a solutionChanged event. The new

solution is stored in `solutionhandle`. The argument `action` determines if the computation is 'forward', 'backward' or 'extend'. Other actions could be specified but this option is not utilized in the current implementation.

- `function shutdownGUI(obj)`

  This function is called when the user closes the main window. The session will call the `saveToFile` method to save the current state to a file in `System/`. The session will call a similar routine for its submodels `windowmanager` and `outputmanager`. The windowmanager and outputmanager will close all their windows and will make sure the positions are stored for recovery. A `shutdown` event is generated to notify other GUI components.

- `function loadFromFile(obj, local)`

  This functions loads a stored session object from a file into the current session objects. The submodels `windowmanager` and `outputmanager` are called upon to restore their windows.

- The methods `lock` and `unlock` are called upon by computation objects to freeze the GUI during computation. This blocks the user from making changes while computing. A `resetState` method is available to reset the GUI if it were to malfunction during computation. For example, this situation occurs when the user debugs code during computation and exits debug mode before the unlocking occurs. This action causes the GUI to remain in a locked state. The reset function can be called to restore the session. This function is available to the user through the 'Help' menu.

When starting the GUI, the session object is stored as the global variable `session`. Global variables are frowned upon in object oriented programming. However, the session object is passed along where needed, as it should be. The global variable `session` is only used to access it from the command line by the developer. It is useful for debugging, one can always access the session object by declaring it global and view its content. One can then check it for errors while the GUI is still active.

6.5 The Settings class

The large majority of routines in MatCont is directly related to numerical continuation, including initialization and branch switching, or directly related to time integration, including Poincaré maps. Internally, the MatCont GUI treats these routines in a unified way in which each routine uses a collection of *settings*. The interaction of these settings with the user and with the computational routines is crucial for the performance of MatCont. The object containing these settings is defined in the file *CLSettings.m*.

In Appendix H we provide a listing of a settings object that contains all possible settings.

6.5.1 The settings of a continuation or integration routine

The MatCont directory **GUI** contains a subdirectory **SettingModel** which collects the code related to the use of *settings* in MatCont.

In particular it contains the class file `CLSettings.m` which defines the class of objects of type ClSettings.

It contains also the class file `ClSetting.m` which defines the class of objects of type ClSetting.

A ClSettings object is a key value collection in which each key (an internal name) refers to an object of the class ClSetting.

A ClSetting obect has a number of properties, namely:

- a 'displayname', for example 'Amplitude'
- a 'value', must be a MATLAB object
- a 'validitycheck'. The validity checks are contained in a file. For each key there is a set of functions whose argument is the 'value' of the ClSetting object and whose output is a boolean value.
- a 'visibility'. This is a boolean value which indicates whether the ClSetting object is visible to the end user.
- a 'groupid'. Can be one of $0, 1, 2, 3$ where 1 refers to the **Starter** window, 2 refers to the **Continuer** window and 3 refers to the **Integrator** window. A ClSetting object with 'groupid' 0 is not rendered in one of these three windows but may appear elsewhere, e.g. in the MatCont main window in an options menu.
- a 'subgroupid' determines in which part of the 'groupid' window the object is to be rendered, e.g. in the 'Continuation Data', 'Corrector Data', or 'Stop Data' of the **Continuer** window.
- a 'itemid', determines the ordering in the display of the 'subgroupid'.
- a 'help' i.e. a tooltip, a text that appears when the cursor moves over the displayname.
- a boolean 'editable' which indicates whether the 'value' of the ClSetting object can be changed

The class ClSetting further contains a subclass ClSettingBlank with an extra property 'Blankmessage'. If its (boolean) value is empty then the 'value' is not subject to the 'validitycheck'.

Preparing a new type of computation e.g. a limit cycle curve from a Hopf point (initializer `init_H_LC`,) includes the following steps:

- The visibility of all settings is set to zero.

- The existing required settings are re-activated, i.e. become visible and are not overwritten. The non-existing required settings are created, filled with default values and made visible.
- The user can overwrite the values except for the rare cases that they are not 'editable'.

6.6 The command line interface

The structure of the MatCont GUI allows to use practically all GUI functionalities from the command line in an interactive way. The collection of MATLAB commands to perform this is called the Command Line Interface (cli).

This has a number of advantages. For example, if the number of state variables or parameters is large and provided by an external file, then it is possible in this way to load them as vectors directly into MatCont. Also, it is possible to write a MATLAB script that uses MatCont in a semi-automatic way, for example to perform a bifurcation analysis for a range of values of external parameters, i.e. parameters which are fixed in each particular bifurcation analysis.

The first step in the use of the cli is to set one or more of the class variables `settings`, `session` or `solution` global. These classes form the middle layer of MatCont, i.e. the layer between Cl_MatCont and the GUI as seen by the user. In the cli we address directly this intermediate layer which also contains the protection against e.g. nonsense input in the Cl_MatCont routines.

For simplicity, we start with the use of the `settings` class. We introduce the *Rössler* system with state variables x, y, z and parameters A,B,C, as in §5.6 and in Tutorial I in Appendix A. To avoid confusion we introduce it under the new name *ROESSLERTest*. After selecting the initial point type 'Point' and introducing some initial values we turn to the MATLAB command line to reset the values:

```
>> global settings
>> settings
settings =
              system: ROESSLERTest
                  IP: Point (P)

        option_pause: At Special Points
      option_archive: 2
       option_output: 1
 option_tsearchorder: true
 option_moorepenrose: true
    option_increment: 1e-05
============================
                time: 4.55359945382384
                co_x: 1
```

```
                co_y: 2
                co_z: 3
               coord: [ 1, 2, 3 ]

          parameters: [ 0.25, 0.4, 4.5 ]
                pa_A: 0.25
                pa_B: 0.4
                pa_C: 4.5
============================
            Interval: 200
       eventfunction: <disabled>
    InitStepSize_sim: <automatic>
     MaxStepSize_sim: <automatic>
        RelTolerance: 0.001
        AbsTolerance: 1e-06
              Refine: 4
         Normcontrol: false

>> settings.co_x
ans =
     1
>> settings.co_y
ans =
     2
>> settings.coord
ans =
     1     2     3
>> settings.coord.set([-5 5 10])
>> settings.parameters.set([0 0.4 4.5])
>> settings
settings =
              system: ROESSLERTest
                  IP: Point (P)

        option_pause: At Special Points
      option_archive: 2
       option_output: 1
 option_tsearchorder: true
 option_moorepenrose: true
    option_increment: 1e-05
============================
                time: 4.55359945382384
                co_x: -5
```

```
                co_y: 5
                co_z: 10
               coord: [ -5, 5, 10 ]

          parameters: [ 0, 0.4, 4.5 ]
                pa_A: 0
                pa_B: 0.4
                pa_C: 4.5
============================
            Interval: 200
       eventfunction: <disabled>
    InitStepSize_sim: <automatic>
     MaxStepSize_sim: <automatic>
        RelTolerance: 0.001
        AbsTolerance: 1e-06
              Refine: 4
         Normcontrol: false
```

We turn to the GUI and check that these changes are visible in the **Initializer** window; then we perform the computation in the GUI. The outcome of the computation is stored in the `solution` class. We inspect it in the MATLAB command window:

```
>> global solution
>> solution
solution=
   SimCompSolution with properties:
             t: [1477x1 double]
             y: [1477x3 double]
        method: @ode45
         tspan: [4.5536 204.5536]
       options: [1x1 struct]
         param: [0 0.4000 4.5000]
            tE: []
            yE: []
            iE: []

>> solution.y
ans=
-5.0000 5.0000 10.0000
-5.0763 4.9739  9.5078
...
```

We return to the GUI, select the last point of the computed orbit and declare it to be an equilibrium. By default the Curve Type will now be 'Equilibrium'. We again inspect the `settings`:

```
>> settings
settings =
              system: ROESSLERTest
                  IP: Equilibrium (EP)

        option_pause: At Special Points
      option_archive: 2
       option_output: 1
 option_tsearchorder: true
 option_moorepenrose: true
    option_increment: 1e-05
============================
        InitStepsize: 0.01
         MinStepsize: 1e-05
         MaxStepsize: 0.1

      MaxNewtonIters: 3
        MaxCorrIters: 10
        MaxTestIters: 10
        VarTolerance: 1e-06
        FunTolerance: 1e-06
       TestTolerance: 1e-05
               Adapt: 3

        MaxNumPoints: 300
         CheckClosed: 50
============================
                co_x: -0.000335576810360529
                co_y: -0.000180608885605843
                co_z: -3.20127988121691e-05
               coord: [ -0.00033557681, -0.00018060888, -3.2012798e-05 ]

          parameters: [ 0, 0.4, 4.5 ]
                pa_A: 0
                pa_B: 0.4
                pa_C: 4.5
         pa_A_select: false
         pa_B_select: false
         pa_C_select: false
```

```
          test_EP_BP: true
           test_EP_H: true
          test_EP_LP: true

         eigenvalues: true
============================
```

We need to select an active parameter in the equilibrium continuation. This can be done via the command line:

```
>> settings.pa_A_select.set(true)
```

We execute the continuation in the GUI and get the output:

```
first point found
tangent vector to first point found
label = H , x = ( 0.000000 0.000000 0.000000 0.086282 )
First Lyapunov coefficient = -2.884394e-03

elapsed time  = 1.8 secs
npoints curve = 49
```

The solution data can be obtained in the MATLAB Command window:

```
>> solution
solution=
ContCurve with properties:

x: [4x49 double]
v: [4x49 double]
s: [3x1 struct]
h: [5x49 double]
f: [3x49 double]

>> size(solution.x)
ans=
     4      49
```

So far all driving steps were executed in the GUI, where they were relegated to the `session` class. In a more advanced use of the cli we can also perform these steps from the command line by declaring the `session` class global. We then add the commands:

```
>> global session
>> session
```

The `session` output recalls the settings and then offers a choice of three buttons, labeled

```
View Computations  View Actions  View Switches
```

Clicking the first button is equivalent to executing the command

```
>> session.select()
```

It provides the numbered list of computations (usually time integrations or continuation runs) that can be selected from the given Initial Point type. They can be selected by clicking, or by typing

```
>> session.select(k)
```

where k is the list number.

Clicking the second button is equivalent to executing the command

```
>> session.compute()
```

It provides a fixed list of options, namely **Forward, Backward,** and **Extend**. The list can be handled as in the previous case.

Clicking the third button is equivalent to executing the command

```
>> session.switches()
```

It provides the list of objects that can be chosen as new Initial Points. The list can be handled as in the previous cases. One can change the type of an initial point from the command-line by using the instruction

```
>>> session.changeInitPoint('H')  %force IP type to be 'Hopf (H)'
```

We note that it is always possible to get information on the current system:

```
>> settings.system
ans=
CLSystem with properties:

               name: 'ROESSLERTest'
        coordinates: {'x'  'y'  'z'}
         parameters: {'A'  'B'  'C'}
                dim: 3
               time: 't'
             handle: @ROESSLERTest
```

```
    userfunctions: []
          ufdata: []
 diagramlocation: '/home/nineiryn/repo/MatCont/Systems/ROESSLERTest'
          derstr: 'SSSNN'
       equations: [3x16 char]
```

6.7 Output Interpreters

A computation provides a solution with an *Output Interpreter* object. There are two main types of interpreter objects, one for time integration (simulation) and one for continuation. There is also a special interpreter (a subclass of the continuation interpreter) for connecting orbits.

An output interpreter takes a solution and provides a mapping that is used in the Numeric window, the Plot Layout windows and the spreadsheet view of the data. This mapping translates the raw output data into concepts meaningful for the user. In the case of continuation, the interpreter object maps the coordinates, parameters, testfunctions, userfunctions, multiplier/eigenvalues, stepsizes, period (if present) onto the (`x,v,s,h,f`) output of the continuer (*cont.m*).

For example, a system with 2 coordinates (u, v) and one parameter (d) has the following interpretation for an equilibrium curve:

```
x(1,:): u
x(2,:): v
x(3,:): d

h(1,:): stepsize
h(2,:): correction
h(3,:): BP
h(4,:): H
h(5,:): LP

f(1,:): eig
f(2,:): eig
```

Note that along this curve three bifurcations (BP, H and LP) were monitored for detection.

The output from a time integration is easy to interpret and is performed by the class in *CLSimOutputInterpreter.m*. The (`t,y`) output in the case of the above example will lead to `y(1,:)` being interpreted as u and `y(2,:)` being interpreted as v. The `t` vector is labeled as time.

The output from a continuation is far more complex and is performed by the class in *CLContOutputInterpreter.m*. There exist subclasses like *CLContOutputInterpreterHom.m* who add a few exceptions in the interpretation for connecting orbits. The complexity of the interpretation is caused by the many settings that alter the interpretation:

- Working with points, cycles or connecting orbits
- The enabling or disabling of certain bifurcation detections. Some bifurcations require two testfunctions
- The enabling or disabling of userfunctions if present
- The selection of the active parameter.
- The period is always given in the output for codim 1 limit cycle continuations; for limit cycle continuation the period is given in the output only if selected as active.
- The options to select the computation of eigenvalues or multipliers or (d)PRC data.

A change of computation or a change in the current settings can cause the interpretation to change. The *Output Manager*, which maintains the plot windows and the numeric window, monitors any relevant changes through the Event mechanism in `session` and calls upon the current interpretation to produce new selections for the numeric window and plot windows. For example, if one selects to compute eigenvalues for an equilibrium continuation the option to plot eigenvalues or to monitor eigenvalues in the numeric window becomes available.

The output interpreters have as main function the function `interpret`:

```
function [outputmap, numconfig, plotsel] = ...
                    interpret(obj, solution, settings, computation)
```

Based on the `settings` and the `computation`, the interpreter produces three outputs. If the `solution` argument is non-empty, the `settings` and `computation` will be taken from the solution. The `outputmap` is a simple mapping used in the spreadsheet data viewer and is also used for debugging. The `numconfig` provides a list of categories with values which is used as the selection in the numeric window. The `plotsel` provides the menu in the plot layout windows.

Chapter 7

Future work

The present list of tasks for future work was written up at the beginning of October 2018. It is roughly ordered by what we consider as priority.

1. As mentioned in §2.9.2 and §6.6 a Command Line Interface (cli) allows direct interaction between the MATLAB workspace and the new GUI version of MatCont. However, this is not yet documented or discussed in a tutorial.

2. The new MatCont GUI supports the use of **Event functions** during time integration and the computation of Poincaré maps, see §5.6, but so far this is not yet documented or discussed in a tutorial.

3. The restrictions on the names of auxiliary variables (§2.8.1) in the system m-files generator should be lifted as the names for these variables can come into conflict with built-in MuPAD names. The current version of the generator will produce an error message and force the user to change the names of the variables.

4. Initializers for the two Hopf branches that are rooted in a Double Hopf (HH) point (similar to the two NS branches that are rooted in an NSNS point of maps).

5. Similarly, an initializer for the two NS branches rooted in an NSNS point of limit cycles.

6. Computation of Lyapunov exponents of orbits for ODEs. A combination of the normal form coefficients in MatCont with computation of Lyapunov exponents is a powerful tool to investigate unfoldings of codim 2 bifurcation, [23].

7. Plotting two variables along the y-axis as a function of the same variable along the x-axis in 2D plots in the MatCont GUI. At present, a replacement strategy is to open two windows simultaneously.

8. Vectorization of the continuation algorithms for limit cycles and their codimension 1 bifurcations. In the case of limit cycles this is done in COCO [17].

9. Introduction of vector state variables. This might in particular be useful in the case of discretized PDEs.

10. Import of Systems Biology Markup Language (SBML) systems into MatCont (provided in the pre-2018 MatCont GUI but not often used).

Chapter 8

Summary

8.1 English summary

The mathematical background of MatCont is bifurcation theory which is a field of hard analysis. Bifurcation theory treats dynamical systems from a high-level point of view. In the case of continuous dynamical systems this means that it considers nonlinear differential equations without any special form and without restrictions except for differentiability up to a sufficiently high order (in the present state of MatCont never higher than five.) The number of equations is not fixed in advance and neither is the number of variables or the number of parameters, some of which can be active and others not. The aim of bifurcation theory is to understand and classify the qualitative changes of the solutions to the differential equations under variation of the parameters. This knowledge cannot be applied to practical situations without numerical software, except in some artificially constructed situations.

'MatCont' stands for 'MATLAB Continuation'. Its counterpart for discrete time systems generated by iterated maps is called 'MatContM'. Both packages can be used either from the command line or by using a GUI. The command line use is referred to as Cl_MatCont or Cl_MatContM, respectively. The GUI versions are more user-friendly and are probably used more often. The command line versions are more flexible and powerful but require more work and more insight in the underlying mathematics and numerical methods.

Both MatCont and MatContM are MATLAB successor packages to CONTENT but were developed from scratch with many new functionalities. The project is lead jointly by W. Govaerts (Ghent University, Belgium) and Yu.A. Kuznetsov (Utrecht, The Netherlands) and more recently also by H.G.E. Meijer (University of Twente) who has been a long-time co-developer.

On October 6, 2016 in the Web of Science core collection 462 papers cited the first (2003) paper on MatCont. By September 22, 2018 this number increased to 617. The follow-up paper (2008) was then cited 49 times. The citing papers cover most fields of quantitative science, e.g.

- Rayleigh-Bénard convection;
- Bacteria-phage interaction in a chemostat;
- Design of cell cycle oscillators;
- Control of rotating blade vibrations;
- Vehicle systems dynamics;
- Electronic circuits;
- Population dynamics of Xenopus tadpole;
- Bottom fishing;
- Dynamics of landscapes;
- Neural models;
- Pattern storage in neural networks;
- Insulin secretion and hepatitis;
- Chemical reaction engineering;
- Climate warming;
- Magnetic Resonance Force Microscopy;
- Harvesting piezoelectric vibration energy;
- Onset and dynamics of bicycle shimmy;
- Aeronautical engineering.

The software related to the MatCont project, including the manuals and tutorials, is freely available from www.sourceforge.net. The main other general purpose packages for continuation and bifurcation software PyDSTool, AUTO-07P, and COCO are also available on www.sourceforge.net. On August 6, 2018 the number of weekly downloads was recorded as 13 for PyDSTool, 43 for AUTO-07P, 6 for COCO and 390 for MatCont.

A key ingredient of MatCont is numerical continuation, whereby curves of objects of a given type (for example, equilibria, periodic orbits, Hopf bifurcation points, homoclinic orbits ...) are computed under variation of one or more system parameters.

In the present thesis we describe our contributions to the MatCont project. This includes several new numerical algorithms, improvements to many existing algorithms, many software improvements and, most importantly, the development of a completely new environment for MatCont with the following features:

- A clear separation of computational and control routines to increase flexibility, readability and maintainability.
- The workflow is consistently organized along the lines initializer — computation — solution. The notion of continuation is replaced by the more general notion of computation.
- A better handling of the generated data. These data are represented and managed by the diagram organizer, the data browser and the spreadsheet viewer.
- The internal working of the software is documented in Chapter 6. This chapter provides a general overview. More details are obtained through the internal documentation in the code which can be accessed online.
- The software contains automatic tests to check if a new MATLAB version produces the same results as the previous version.
- Error handling of plots is improved so that plot errors caused by e.g. command line interference, of by GUI interference when computations are suspended, do not crash the computations.
- Each input field has input restrictions and these are checked to minimize input errors. So for example it will not be possible to input a float or a question mark if a positive integer number is required. Errors are reported in the MATLAB command line. On the other hand, numerical fields can be filled with MATLAB expressions, provided they can be evaluated in the command line. So one can insert $2 * Pi$ instead of its decimal expansion 6.283184...
- A Command Line Interface (cli) allows a direct interaction between the command line and the GUI version of MatCont.

The thesis is structured as follows. After an introductory Chapter 1 we discuss in Chapter 2 (Preliminaries) general aspects of MatCont and the mathematical background of bifurcation theory for ODEs and for maps with survey tables of bifurcations and branch switchings. In the last two sections of Chapter 2 we give an overview of our own contributions to the development of the MatCont and MatContM software.

One of them is the merging of MatCont and Cl_MatCont. In MatCont5.4 (September 2014) and earlier versions of MatCont the command-line and GUI versions were separate. This was inconvenient from the point of algorithmic development since all algorithmic changes had to be input twice. We merged the two packages which required several important changes, since the continuer `cont.m` now runs in two different ways, depending on how the MATLAB session is started.

Another contribution is an improved code for generating the system m-files. These files are sometimes called odefile or mapfile to indicate whether odes or maps are being studied. They are an essential part of the whole software since they provide the handles to the dynamical system that is being studied.

We developed an algorithm for switching to two different NS curves in a Double Neimark-Sacker (NSNS) point in MatContM. We implemented this algorithm and it is remarkably simple and efficient, and quite different from the idea that is traditionally used for switching to a second branch of equilibria when a branch point of equilibria is detected on an equilibrium curve. The continuation variables in the continuation of a NS curve consist not only of the state variable x and the free parameter p but also of the scalar variable $k = \cos(\alpha)$ where the Neimark-Sacker eigenvalues of the Jacobian are $e^{\pm i\alpha}$. Hence the NSNS point corresponds in fact to two different points in (x, p, k) space with the same x and p but different k values. Therefore the two Neimark-Sacker branches can simply be started from these two points. In MatContM this corresponds to the initializers `init_NSNS_NS_same` and `init_NSNS_NS_other` where 'same' correspond to the curve on which the NSNS point was detected. We note that it is not necessary to compute tangent vectors and that it is even possible to change the choice of the free parameters, which is not the case in a branch point of equilibria.

In Chapter 3 (Numerical continuation: the algorithmic basis) we discuss the (numerical) algorithms which form the computational core of MatCont and MatContM. Few numerical details are given here since they can be found elsewhere; we focus on the aspects that have to be understood by users and/or developers.

In Chapter 4 (MatContM for maps) we discuss some of our own contributions to the MatContM software for maps and applications thereof. This involves Lyapunov exponents for maps, the growing of stable and unstable manifolds, the initialization of connecting orbits and the detecting of codimension 1 and codimension 2 bifurcations of homoclinic orbits. The joint work with L. Vanhulle on the intersection of a stable and an unstable manifold is new and unpublished. This work is restricted to planar maps and forms the key step towards the computation of homoclinic and heteroclinic connections and tangencies.

Since each manifold is approximated by line segments this intersection problem is a special case of the computation of the intersection points of a collection of line segments. The latter problem is a well-studied one and the standard solution is the *line sweep algorithm*, but so far there is no efficient way to implement this algorithm in MATLAB.

In fact, the standard solution so far is to simply consider each pair of line segments (a stable one and an unstable one) and check whether they intersect. We call this the algorithm of Bruschi. It is obviously rather inefficient but it is still acceptable since its cost is typically much smaller than the cost of computing the manifolds.

We present an algorithm based on the idea of first looking into one projection of the two manifolds. In a first round we select the pairs of intervals whose $x-$ projections overlap. Typically this is a small fraction of all pairs. In the second round we select from this fraction those pairs which also have overlapping $y-$ projections. For the remaining pairs, usually a tiny fraction of all pairs, we check whether they have an intersection in 2D-space. We present two versions of this algorithm, which we call the first one and the improved one, respectively. The first version is coded in the m-file `Projectie.m` and listed in Appendix E. The improved version is coded in the m-file `Projectie2.m` and listed in Appendix F. For comparison purposes both versions are available in MatContM but only the improved

version is called in the GUI.

MatContM now contains two routines to compute the Lyapunov exponents of a map. One computes all Lyapunov exponents; the other is a more restricted but efficient algorithm to compute the largest Lyapunov exponent in the case of planar systems. As an example application we study a monopoly model in which we detect stable behaviour in two small parameter intervals (length less than 2×10^{-3}).

Chapter 5 (Front end features of the new MatCont GUI.) deals with the user-oriented features of the new MatCont GUI. It discusses the MatCont database, the *Data Browser* to access it, and the structure of the mat-files in which the data of computed curves are stored. It shows how to obtain a spreadsheet view of a computed curve and how the MatCont data can be exported to other environments. A survey of the input and output panels and their functionalities is also given.

Chapter 6 (Internal working of the new MatCont GUI) is developer-oriented. This is, in fact, the first time that the inner working of a version of the MatCont software is explicitly described.

The new GUI of MatCont is based on the MVC design principle. We start with a description of the "Model" which is based on the Class file `Session.m`, a subclass of the `handle` class of MATLAB.

Compared to the pre-2018 version of the GUI there is a restructuring and redistribution of the information in the mat-file of the system over a number of submodels to make the working of the software more logical and transparent. It also allows for more control when reloading a system, for example if a diagram was removed. This data model stores all the GUI related data; its fields have to belong to specific classes, some of which are defined specifically for MatCont, namely System, Curve, CurveType, PointType, Branch.

Chapter 7 (Future work) mentions some topics with varying degrees of complexity for further investigation.

This thesis further contains the Appendices A-I. A-D are tutorials to make the user familiar with the basic functionalities of the new version of **MatCont**.

Tutorial I (Appendix A) deals with the input of a new system, time integration, selecting initial data, 3D plots, starting a computation, inspecting a computed curve in the **Data Browser** and in a spreadsheet form, qualitative changes of orbits under parameter variation, exporting a figure, 2D plots and archiving solutions.

Tutorial II (Appendix B) deals with the location of equilibria by time integration, selection of special points on a computed curve via the **Data Browser**, computing a curve of equilibria, using the **Numeric** window, detecting fold and Hopf points and interpreting their normal form coefficients, detecting branch points and continuing a new branch of equilibria, introduction and use of userfunctions.

Tutorial III (Appendix C) deals with the initialization of limit cycles by time integration, starting a limit cycle continuation from a Hopf point, detecting a period doubling bifurcation, detecting a limit point of cycles bifurcation and a Neimark-Sacker (torus) bifurcation, modulated oscillations (movement on a stable torus), detecting a homoclinic orbit.

Tutorial IV (Appendix D) deals with the continuation of fold and Hopf points under variation of two parameters, two-parameter bifurcation diagrams, continuation of a limit point of cycles curve by starting from a Generalized Hopf (Bautin) point and continuation of a Neimark-Sacker (torus) curve under variation of two parameters.

The appendices E-G are related to the joint work with L. Vanhulle. Appendices E, F were mentioned in the discussion on Chapter 4. Appendix G (Banen) is the listing of the routine which extracts connecting orbits from the set of intersection points of an unstable and a stable orbit. The appendices H-I provide the complete lists of settings and computations in MatCont.

8.2 Nederlandstalige samenvatting

De wiskundige achtergrond van MatCont is bifurcatietheorie, een deelgebied van de wiskundige analyse. In bifurcatietheorie worden dynamische systemen beschouwd vanuit een hoog niveau van algemeenheid. In het geval van continue dynamische systemen betekent dit dat men stelsels van normale niet-lineaire differentiaalvergelijkingen beschouwt zonder speciale vorm en zonder restricties behalve een voldoend hoog order van differentieerbaarheid (in de huidige vorm van MatCont niet hoger dan vijf.) Het aantal vergelijkingen is niet bij voorbaat vastgelegd, evenmin als het aantal variabelen en het aantal parameters, waarvan er sommige actief kunnen zijn en andere niet. Het doel van bifurcatietheorie is het begrijpen en classificeren van de kwalitatieve veranderingen van de oplossingen van de differentiaalvergelijkingen onder variatie van de parameters. Deze kennis kan niet toegepast worden in concrete situaties zonder numerieke software behalve in sommige artificieel geconstrueerde gevallen.

'MatCont' staat voor 'MATLAB Continuation'. De tegenhanger voor discretetijd systemen die gegenereerd worden door geïtereerde afbeeldingen wordt 'MatContM' genoemd. Beide pakketten kunnen ofwel gebruikt worden via de command line of via een GUI. We verwijzen naar de command line vorm als Cl_MatCont of Cl_MatContM, respectievelijk. De GUI versies zijn meer gebruiksvriendelijk en worden waarschijnlijk meer gebruikt. Anderzijds zijn de command line versies flexibeler en krachtiger maar vereisen meer werk en inzicht in de onderliggende wiskundige en numerieke methoden.

MatCont en MatContM zijn MATLAB opvolgers van CONTENT maar werden volledig nieuw ontwikkeld met vele nieuwe functionaliteiten. Het project wordt geleid door W. Govaerts (Gent) en Yu.A. Kuznetsov (Utrecht, Nederland) en meer recent door H.G.E. Meijer (Universiteit Twente) die ook zeer lang actief was als co-ontwikkelaar.

Op 6 oktober 2016 bevatte de core collection van de Web of Science 462 artikels die het eerste artikel (2003) over MatCont citeerden. Op 22 september 2018 was dit aantal gestegen tot 617. Het follow-up artikel (2008) werd toen 49 keer geciteerd. De citerende artikels komen uit ongeveer alle domeinen van de kwantitatieve wetenschappen, bijvoorbeeld:

- Rayleigh-Bénard convection;
- Bacteria-phage interaction in a chemostat;

- Design of cell cycle oscillators;
- Control of rotating blade vibrations;
- Vehicle systems dynamics;
- Electronic circuits;
- Population dynamics of Xenopus tadpole;
- Bottom fishing;
- Dynamics of landscapes;
- Neural models;
- Pattern storage in neural networks;
- Insulin secretion and hepatitis;
- Chemical reaction engineering;
- Climate warming;
- Magnetic Resonance Force Microscopy;
- Harvesting piezoelectric vibration energy;
- Onset and dynamics of bicycle shimmy;
- Aeronautical engineering.

De software met betrekking tot het MatCont project, inbegrepen de handleidingen en tutorials, is vrij beschikbaar op www.sourceforge.net. De belangrijkste andere general-purpose software pakketten voor continuatie- en bifurcatietheorie PyDSTool, AUTO-07P, en COCO zijn ook beschikbaar op www.sourceforge.net. Op 6 augustus 2018 werd het aantal wekelijkse downloads weergegeven als 13 voor PyDSTool, 43 voor AUTO-07P, 6 voor COCO en 390 voor MatCont.

Numerieke continuatie is een belangrijk ingrediënt van MatCont. Hierbij worden objecten van een gegeven type (bijvoorbeeld equilibria, periodieke banen, Hopf bifurcatiepunten, homoclinische connecties etc.) gevolgd onder de variatie van een of meer parameters van het systeem.

In dit proefschrift beschrijven we onze bijdragen tot het MatCont project. Dit omvat verschillende nieuwe algoritmen, verbeteringen aan vele bestaande algoritmen, verbeteringen aan de bestaande software en als belangrijkste bijdrage een volledig nieuwe en up-to-date software-omgeving voor MatCont met de volgende aspecten:

- Een duidelijke scheiding tussen computationele routines en controleroutines. Dit verhoogt de flexibiliteit en zorgt voor een betere leesbaarheid van de code en een eenvoudiger onderhoud van de software.
- De workflow is georganiseerd volgens de lijnen van initialisatie – berekening – oplossing. De software is niet langer zuiver gericht op continuatie maar werkt met het breder concept van een berekening.
- Een betere afhandeling van de gegenereerde data. Deze data worden weergegeven en bewerkt door de *diagram organizer*, de *data browser* en de *spreadsheet viewer*.
- De interne werking van de software documenteren we in Hoofdstuk 6. Dit hoofdstuk biedt een algemeen overzicht. Meer details vindt men terug in de documentatie binnenin de code. Deze documentatie is ook online beschikbaar.
- De software bevat automatische tests die nagaan of een nieuwe MATLAB versie dezelfde resultaten produceert als de vorige versie.
- De foutafhandeling bij het plotten is verbeterd zodat plotfouten, bijvoorbeeld als gevolg van een command line tussenkomst, of bij een GUI tussenkomst wanneer de berekeningen opgeschort zijn, de berekeningen niet doen crashen.
- Ieder inputveld heeft aangepaste restricties en die worden gecheckt om inputfouten te minimaliseren. Bijvoorbeeld zal het niet mogelijk zijn om een float of een vraagteken in te vullen als een geheel getal gevraagd wordt. Fouten worden gemeld in de MATLAB command line. Anderzijds, numerieke velden kunnen ingevuld worden met MATLAB uitdrukkingen op voorwaarde dat ze kunnen geëvalueerd worden in de MATLAB command line. Men kan dus $2 * Pi$ invullen in plaats van de decimale expansie 6.283184...
- Een Command Line Interface (cli) laat een directe interactie toe van de command line met de GUI versie van MatCont.

Het proefschrift is als volgt georganiseerd. Na een inleidend hoofdstuk ('Introduction') bespreken we in Hoofdstuk2 ('Preliminaries') algemene aspecten van MatCont en de wiskundige achtergrond van bifurcatietheorie voor ODEs en voor afbeeldingen met overzichtstabellen voor bifurcaties en takverwisselingen. In de laatste twee secties van Hoofdstuk 2 geven we een globaal overzicht van onze bijdragen tot de ontwikkeling van de MatCont en MatContM software.

Een van die bijdragen is het samenvoegen van MatCont en Cl_MatCont. In MatCont5.4 (september 2014) en vroegere versies van MatCont zijn de command-line en GUI versies gescheiden. Dat was onhandig van het standpunt van de algoritmische ontwikkeling daar alle algoritmische veranderingen tweemaal moesten ingevoerd worden. Wij voegden de twee versies samen, hetgeen verschillende belangrijke veranderingen impliceerde daar de continuer `cont.m` nu werkt in twee verschillende modussen, afhankelijk van de wijze waarop de MatCont sessie gestart wordt.

Een andere bijdrage is de verbeterde code voor het genereren van de m-files van de systemen. Die files worden soms odefiles of mapfiles genoemd naargelang ze voor ODEs of voor afbeeldingen gebruikt worden. Ze vormen een essentieel onderdeel van de hele software daar ze de handles vormen tot het dynamisch systeem dat bestudeerd wordt.

We ontwikkelden een algoritme om van een dubbel Neimark-Sacker (NSNS, torus) punt in MatContM de berekening van de twee verschillend Neimark-Sacker krommen op te starten. We implementeerden dit opmerkelijk eenvoudige en efficiënte algoritme dat heel verschillend is van het idee dat traditioneel gebruikt wordt voor bijvoorbeeld het switchen naar de tweede tak in een branch punt dat ontdekt wordt tijdens een continuatie van vaste punten. De continuatievariabelen in de continuatie van een NS kromme bestaat namelijk niet alleen uit de toestandsvariabelen en de vrije parameter maar ook uit de scalaire variabele $k = \cos(\alpha)$ waarbij de Neimark-Sacker eigenwaarden van de Jacobiaan van de vorm $e^{\pm i\alpha}$ zijn. Het NSNS punt correspondeert dus in feite met twee verschillende punten in de (x, p, k) ruimte met dezelfde x en p maar verschillende waarden van k. Daarom kunnen de twee Neimark-Sacker takken eenvoudig worden gestart van de twee punten in de (x, p, k) ruimte. In MatContM correspondeert dit met de initializers `init_NSNS_NS_same` en `init_NSNS_NS_other` waarbij 'same' correspondeert met de kromme waarop het NSNS point gedetecteerd werd. We merken op dat het dus niet nodig is om de raaklijnvectoren langs de twee takken te berekenen of te benaderen. Men kan zelfs de keuze van de vrije parameter veranderen hetgeen niet kan bij de vertakking in een branch punt van vaste punten.

In Hoofdstuk 3 (Numerical continuation: the algorithmic basis) bespreken we de (numerieke) algoritmen die de computationele kern van MatCont en MatContM vormen. We geven weinig details daar die elders beschikbaar zijn; we focussen op de aspecten die nuttig zijn voor de gebruikers en ontwikkelaars.

In Hoofdstuk 4 (MatContM for maps) bespreken we in meer detail een deel van onze bijdragen tot de ontwikkeling van MatContM en een toepassing daarvan. Het gaat om de berekening van Lyapunov exponenten voor afbeeldingen, het groeien van onstabiele en stabiele variëteiten, de initializatie van connecting orbits, en het detecteren van codimensie 1 en codimensie 2 bifurcaties van homoclinische connecties. Het gemeenschappelijk werk met L. Vanhulle over de intersectie van een onstabiele en een stabiele variëteit is nieuw en ongepubliceerd. In dit werk beperken we ons tot planaire afbeeldingen en het vormt de essentiële stap naar de berekening van homoclinische en heteroclinische connecties en tangencies.

Daar iedere variëteit benaderd wordt door lijnsegmenten is dit intersectieprobleem een speciaal geval van de berekening van alle intersectiepunten van een verzameling van lijnsegmenten. Dit laatste probleem is goed bestudeerd en de standaardoplossing is het zogenaamde *line sweep algorithm*. Maar er is geen efficiënte implementatie in MATLAB bekend voor ons probleem. In feite is de standaardoplossing tot nog toe dat men eenvoudig de intersectie zoekt van ieder onstabiel lijnsegment met ieder stabiel lijnsegment. We noemen dit het algoritme van Bruschi. Hoewel het heel inefficiënt is, is het nog aanvaardbaar omdat de kost ervan klein is in vergelijking met die van het berekenen van de variëteiten zelf.

Wij stellen een algoritme voor dat gebaseerd is op het idee om eerst te kijken naar een

eerste projectie van de twee variëteiten, zeg op de $x-$ as. In die ronde selecteren we alle paren van lijnsegmenten waarvan de $x-$ projecties overlappen. Voor deze paren, meestal een klein deel van alle paren, onderzoeken we dan of hun $y-$ projecties ook overlappen. Slechts voor deze paren, meestal een uiterst kleine fractie van alle paren, onderzoeken we dan of ze ook snijden in de globale tweedimensionale ruimte. We presenteren twee versies van dit algoritme, waarvan de tweede de verbeterde versie genoemd wordt. De eerste versie is gecodeerd in de m-file `Projectie.m` met listing in Appendix E. De verbeterde versie is gecodeerd in de m-file `Projectie2.m` met listing in Appendix F. Ter vergelijking zijn beide versies opgenomen in MatContM maar alleen de tweede wordt opgeroepen in de GUI.

MatContM bevat nu ook twee routines om de Lyapunov exponenten van een afbeelding te berekenen. De ene berekent alle Lyapunov exponenten, de andere is beperkt tot tweedimensionale systemen en berekent alleen de grootste Lyapunov exponent, maar is daarvoor ook veel efficiënter. Als voorbeeld van toepassing bestuderen we een monopoliemodel waarin we onder andere stabiel gedrag detecteren in twee zeer kleine parameterintervallen (diameter kleiner dan 0.001).

Hoofdstuk 5 (Front end features of the new MatCont GUI.) behandelt de gebruikergeoriënteerde aspecten van de nieuwe MatCont GUI. Hierin wordt de MatCont database besproken, evenals de *Data Browser* die toegang geeft tot de database, en ook de structuur van de mat-files waarin de data van de berekende krommen opgeslagen worden. We tonen hoe we een overzicht van de berekende data van een kromme in spreadsheetvorm naar buiten kunnen brengen en hoe de MatCont data geëxporteerd kunnen worden naar een andere omgeving.

Hoofstuk 5 bevat verder een overzicht van de functionaliteiten van het MatCont hoofdpaneel en van de input en output panelen.

Hoofdstuk 6 (Internal working of the new MatCont GUI) is ontwikkelaar-georiënteerd. Het is in feite het eerste document waarin de interne opbouw van de GUI van een MatCont-versie wordt besproken en dat dus een houvast kan bieden voor later verder werk.

De nieuwe GUI van MatCont is gebaseerd op het MVC ontwikkelingsprincipe. We beginnen met een beschrijving van het "Model" dat een element is van de Class file `Session.m`, een subklasse van de `handle` klasse van MATLAB.

In vergelijking met de pre-2018 versies van de MatCont GUI is er een herverdeling van de informatie in de mat-file van het systeem over een aantal submodellen van de GUI om de werking van de software meer logisch en transparant te maken. Dit vergemakkelijkt het herladen van het systeem als bijvoorbeeld een diagram weggelaten wordt. Het data model slaat alle GUI-gerelateerde data op in velden die tot klassen behoren die specifiek voor MatCont gedefiniëerd worden, namelijk System, Curve, CurveType, PointType, Branch.

Hoofdstuk 7 (Future work) vermeldt een aantal nieuwe onderwerpen die in aanmerking komen voor verder onderzoek en implementatie. Sommige daarvan zijn uitbreidingen die waarschijnlijk zonder veel moeite gemaakt kunnen worden, andere vereisen een herwerking van de hele code en zijn dus eerder lange-termijn projecten.

Dit proefschrift bevat verder de Appendices A-I. A-D zijn tutorials die de gebruiker vertrouwd moeten maken met de basisfunctionaliteiten van MatCont.

Tutorial I (Appendix A) behandelt de input van een nieuw systeem, tijdsintegratie, het selecteren van initial data, 3D plots, het starten van een berekening, inspecteren van een berekende kromme in de **Data Browser** en in spreadsheetformaat, kwalitatieve veranderingen van banen onder variatie van een parameter, het exporteren van een MATLAB figuur, 2D plots en het archiveren van een berekende kromme.

Tutorial II (Appendix B) behandelt de locatie van equilibria die gevonden worden door tijdsintegratie, selectie van speciale punten op een berekende kromme via de **Data Browser**, berekening door continuatie van een kromme van equilibria, het gebruik van het Numeric venster, het detecteren van fold en Hopf punten en het interpreteren van de coëfficiënten van hun normaalvormen, detecteren van een branch punt en continueren van een nieuwe tak van equilibria, het introduceren en gebruiken van userfunctions.

Tutorial III (Appendix C) behandelt de initializatie van periodieke banen door gebruik van tijdsintegratie, het opstarten van een kromme van periodieke banen uit een Hopf punt, detecteren van een periode-verdubbelings (flip) bifurcatie, detecteren van een limietpunt van cycli, detecteren van een Neimark-Sacker (torus) bifurcatie, gemoduleerde oscillaties, detecteren van een homoclinische baan.

Tutorial IV (Appendix D) behandelt de continuatie van fold en Hopf punten onder variatie van twee parameters, twee-parameter bifurcatiediagrammen, continuatie van een kromme van limietpunten van cycli vanuit een Generalized Hopf (Bautin) punt en de continuatie van een kromme van Neimark-Sacker (torus) bifurcatiepunten onder variatie van twee parameters.

De appendices E-G zijn gerelateerd aan het gemeenschappelijk werk met L.Vanhulle. Appendices E, F werden besproken bij de behandeling van Hoofdstuk 4. Appendix G (Banen) is de listing van de routine die connecting orbits berekent uitgaande van de verzameling van intersectiepunten van een onstabiele en een stabiele baan. De appendices H-I geven een overzicht van alle instellingen ('settings') en berekeningen ('computations') besproken in Hoofdstuk 6 in MatCont.

Bibliography

[1] H.N. Agiza, E.M. ELabbasy, H. EL-Metwally, and A.A. Elsadany, Chaotic dynamics of a discrete prey-predator model with Holling type II. Nonlinear Analysis: Real World Applications, 10(1):116 – 129, 2009.

[2] Saeed Ahmadizadeh, Philippa J. Karoly, Dragan Nesić, David B. Grayden, Mark J. Cook, Daniel Soudry, Dean R. Freestone, Bifurcation analysis of two coupled Jansen-Rit neural mass models.
PLOS ONE, March 27, 2018 (Open Access)
http://journals.plos.org/plosone/article/related?id=10.1371/journal.pone.0192842

[3] Bashir Al-Hdaibat, Willy Govaerts and Niels Neirynck, On periodic and chaotic behavior in a two-dimensional monopoly model, Chaos, Solitons and Fractals 70(2015) 27-37.
http://dx.doi.org/10.1016/j.chaos.2014.10.010

[4] E.L. Allgower and K. Georg, Numerical Continuation Methods: An introduction, Springer-Verlag, 1990 .

[5] Bidesh K. Bera, Chittaranjan Hens, Sourav K. Bhowmick, Pinaki Pal and Dibakar Ghosh, Transition from homogeneous to inhomogeneous steady states in oscillators under cyclic coupling, Physices Letters A 380 (2016) 130-134.
http://dx.doi.org/10.1016/j.physleta.2015.09.044

[6] A. Back, J. Guckenheimer, M.R. Myers, F.J. Wicklin and P.A. Worfolk, DsTool: computer-assisted exploration of dynamical systems. Notices Amer. Math. Soc., 39(4):303-309, 1992.

[7] G. Benettin, L. Galgani, A. Giorgilli, J.-M. Strelcyn Lyapunov, Characteristic Exponents for Smooth Dynamical Systems and for Hamiltonian Systems; A Method for Computing All of Them.
Meccanica, 15 (1980), p. 9-30

[8] A. Beuter, L. Glass, M.C. Mackey and M.S. Titcombe (eds.): Nonlinear Dynamics in Physiology and Medicine, Springer Verlag, New York 2003.

[9] D. Bindel, M. Friedman, W. Govaerts, J. Hughes and Yu.A. Kuznetsov, Numerical Computation of Bifurcations in large equilibrium systems in MATLAB, J. Comp. Appl. Maths 261 (2014) 232-248.
http://dx.doi.org/10.1016/j.cam.2013.10.034

[10] Borisyuk, Roman; Merrison-Hort, Robert; Soffe, Steve R.; et al., To swim or not to swim: A population-level model of Xenopus tadpole decision making and locomotor behaviour, Workshop on Neural Coding (NC) Location: Cologne, GERMANY Date: Aug 29-Sep 02, 2016 BIOSYSTEMS, Vol. 161, Special Issue, pp. 3-14, Published: Nov 2017
https://doi.org/10.1016/j.biosystems.2017.07.004

[11] Bruschi, Caterina, Growing 1D stable and unstable manifolds of n-dimentional maps, Master thesis, Utrecht University 2010.

[12] Fasoli D, Cattani A, Panzeri S (2016) The Complexity of Dynamics in Small Neural Circuits.
PLoS Comput Biol 12(8): e1004992.
https://doi.org/10.1371/journal.pcbi.1004992

[13] Samuel I. A. Cohen, Michele Vendruscolo, Mark E. Welland, Christopher M. Dobson, Eugene M. Terentjev, and Tuomas P. J. Knowles, Nucleated polymerization with secondary pathways. I. Time evolution of the principal moments. The Journal of Chemical Physics 135, 065105 (2011); doi: 10.1063/1.3608916

[14] Jeremy Collie, Jan Geert Hiddink, Tobias van Kooten, Adriaan D. Rijnsdorp, Michel J. Kaiser, Simon Jennings and Roy Hilborn, Indirect effects of bottom fishing on the productivity of marine fish, Fish and Fisheries, 2017, 18, 619-637
DOI: 10.1111/faf.12193

[15] Dan, S (Dan, Surajit); Ghosh, M (Ghosh, Manojit); Nandukumar, Y (Nandukumar, Yada); Dana, SK (Dana, Syamal K.); Pal, P (Pal, Pinaki), Bursting dynamics in Rayleigh-Benard convection, European Physical J - Special Topics 226(9) pp. 2089-2099. Jun 2017
DOI: 10.1140/epjst/e2017-70006-8

[16] Harry Dankowicz and Frank Schilder, Recipes for Continuation, SIAM Publications 2013.

[17] Harry Dankowicz and Frank Schilder, Continuation Core and Toolboxes (COCO).
https://sourceforge.net/projects/cocotools/?source=directory

[18] Jan Sieber and Yuri A. Kuznetsov, DDE-BIFTOOL, Bifurcation analysis for delay-differential equations.
https://sourceforge.net/p/ddebiftool/code/HEAD/tree/

[19] De Berg, Mark; Cheong, Otfried; Van Kreveld, Marc; Overmars, Mark, Computational Geometry: Introduction, Springer 2008.

[20] F. Della Rossa and G. Mastinu, Straight ahead running of a nonlinear car and driver model - new nonlinear behaviours highlighted, Vehicle System Dynamics, January 2018. DOI10.1080/00423114.2017.1422526

[21] D. De Schrijver, Object-georienteerd programmeren in MATLAB, Master thesis, Ghent University, June 2003.

[22] V. De Witte, W. Govaerts, Yu. A. Kuznetsov and M. Friedman, Interactive Initialization and Continuation of Homoclinic and Heteroclinic Orbits in MATLAB, ACM Transactions on Mathematical Software. Volume 38, Issue 3, Article Number: 18, DOI: 10.1145/2168773.2168776 Published: APR 2012

[23] V. De Witte, F. Della Rossa, W.Govaerts and Yu.A. Kuznetsov, Numerical Periodic Normalization for Codim2 Bifurcations of Limit Cycles: Computational Formulas, Numerical Implementation, and Examples, SIAM J. Applied Dynamical Systems 12,2 (2013) 722-788.
DOI: 10.1137/120874904

[24] A. Dhooge, W. Govaerts and Yu. A. Kuznetsov: MATCONT : A MATLAB package for numerical bifurcation analysis of ODEs, ACM Transactions on Mathematical Software 29(2) (2003), pp. 141-164.

[25] A. Dhooge, W. Govaerts, Yu.A. Kuznetsov, H.G.E. Meijer and B. Sautois: New features of the software MatCont for bifurcation analysis of dynamical systems, Mathematical and Computer Modelling of Dynamical Systems, Vol. 14(2), pp. 147-175, Published: 2008
https://doi.org/10.1080/13873950701742754

[26] E. Doedel and J Kernévez: AUTO: Software for continuation problems in ordinary differential equations with applications, California Institute of Technology, Applied Mathematics, 1986.

[27] E.J. Doedel, A.R. Champneys, T.F. Fairgrieve, Yu.A. Kuznetsov, B. Sandstede and X.J. Wang: auto97-00 : Continuation and Bifurcation Software for Ordinary Differential Equations (with HomCont), User's Guide, Concordia University, Montreal, Canada (1997-2000). (`http://indy.cs.concordia.ca`).

[28] Doedel, E.J., Govaerts W., Kuznetsov, Yu.A.: Computation of Periodic Solution Bifurcations in ODEs using Bordered Systems, SIAM Journal on Numerical Analysis 41,2(2003) 401-435.

[29] Doedel, E.J., Govaerts, W., Kuznetsov, Yu.A., Dhooge, A.: Numerical continuation of branch points of equilibria and periodic orbits, Intern. J. Bifurcation and Chaos, 15(3) (2005) 841-860.

[30] Stephen P. Ellner and John Guckenheimer: Dynamic models in biology, Princeton University Press, Princeton 2006.

[31] K. Engelborghs, T. Luzyanina, and D. Roose: Numerical bifurcation analysis of delay differential equations using DDE-BIFTOOL, ACM Trans. Math. Softw. 28 (1), pp. 1-21, 2002.

[32] J. P. England, B. Krauskopf and H. M. Osinga, Computing one-dimensional stable manifolds and stable sets of planar maps without the inverse, SIAM Journal on Applied Dynamical Systems 3(2) 2004 pp. 161-190.
http://dx.doi.org/10.1137/030600131

[33] Ermentrout, B.: Simulating, Analyzing, and Animating Dynamical Systems. Siam Publications, Philadelphia, 2002.

[34] C.P.Fall, E.S.Marland, J.M.Wagner and J.J.Tyson: Computational Cell Biology, Springer 2002.

[35] Fasoli, D; Cattani, A; Panzeri S., Pattern Storage, Bifurcations, and Groupwise Correlation Structure of an Exactly Solvable Asymmetric Neural Network Model.
Neural Comput. 2018 Mar 22:1-38.
doi: 10.1162/NECO_a_01069. [Epub ahead of print]

[36] C. E. Frouzakis, R. A. Adomaitis, and I. G. Kevrekidis: Resonance phenomena in an Adaptively-controlled System. International Journal of Bifurcation and Chaos 01, 01 (1991), 83–106.
https://doi.org/10.1142/S0218127491000075

[37] Freire, E., Rodriguez-Luis, A., Gamero E. and Ponce, E.: A case study for homoclinic chaos in an autonomous electronic circuit: A trip from Takens-Bogdanov to Hopf-Shilnikov, Physica D 62 (1993) 230–253.

[38] M.P. Golden and B.E. Ydstie: Bifurcation in model reference adaptive control systems. Systems & Control Letters 11, 5 (1988), 413-430.

[39] V.S. Gonchenko, Yu.A. Kuznetsov and H.G.E. Meijer, Generalized hénon map and bifurcations of homoclinic of homoclinic tangencies, SIAM J.

Appl. Dyn. Systems, 4,2 (2005) 407-436.
https://doi.org/10.1137/04060487X

[40] S.V. Gonchenko, V.S. Gonchenko, and J.C. Tatjer: Bifurcations of three-dimensional diffeomorphisms with non-simple quadratic homoclinic tangencies and generalized Hénon maps. Regular and Chaotic Dynamics 12, 3 (2007), 233–266.
https://doi.org/10.1134/S156035470703001X

[41] W.J.F. Govaerts, Numerical Methods for Bifurcations of Dynamical Equilibria, SIAM, 2000.

[42] Govaerts, W. and Sautois, B.: Phase response curves, delays and synchronization in MATLAB. Lecture Notes in Computer Science, 3992 (2006), 391-398.

[43] Govaerts, W. and Sautois, B.: Computation of the phase response curve: a direct numerical approach. Neural Comput. 18(4) (2006), 817-847.

[44] W. Govaerts, Yu.A. Kuznetsov, H.G.E. Meijer and N. Neirynck, A study of resonance tongues near a Chenciner bifurcation using MatcontM, Proceedings of the 7th European Nonlinear Dynamics Conference (ENOC 2011) Eds: D. Bernardini, G. Rega and F. Romeo, ISBN: 978-88-906234-2-4.
https://research.utwente.nl/en/publications/a-study-of-resonance-tongues-near-a-chenciner-bifurcation-using-m

[45] W. Govaerts, Yu. A. Kuznetsov, H.G.E. Meijer, B. Al-Hdaibat, V. De Witte, A. Dhooge, W. Mestrom, N. Neirynck, A.M. Riet and B. Sautois: MATCONT: Continuation toolbox for ODEs in MATLAB, Manual 2018.
https://sourceforge.net/projects/matcont/

[46] E. Hacker, O. Gottlieb, Application of reconstitution multiple scale asymptotics for a two-to-one internal resonance in Magnetic Resonance Force Microscopy.
International Journal of Non-Linear Mechanics 94 (2017) 174-199.
DOI: 10.1016/j.ijnonlinmec.2017.04.013

[47] Nahla Haddad, Safya Belghith, Hasséne Gritli, Ahmed Chemori, From Hopf-bifurcation to limitcycles control in underactuated mechanical systems.
International Journal of Bifurcation and Chaos, World Scientific Publishing, 2017, 27 (07),
https://doi.org/10.1142/S0218127417501048

[48] Paul J. Hurtado, Spencer R. Hall, Stephen P. Ellner, Infectious disease in consumer populations: dynamic consequences of resource-mediated transmission and infectiousness.
Theoretical Ecology 7(2) 2014 pp. 163-179.

[49] Jude Dzevela Kong, Paul Salceanu, Hao Wang, A stoichiometric organic matter decomposition model in a chemostat culture.
Journal of Mathematical Biology 76(6) (2018) pp. 609-644.
https://doi.org/10.1007/s00285-017-1152-3

[50] H.G.E. Meijer, W. Govaerts, Yu. A. Kuznetsov, R. Khoshsiar Ghaziani, N. Neirynck: MatContM, A toolbox for continuation and bifurcation of cycles of maps: Command line use.
https://sourceforge.net/projects/matcont/

[51] Kandil, A. and El-Gohary, H. Investigating the performance of a time delayed proportional - derivative controller for rotating blade vibrations, Nonlinear Dynamics, January 2018, DOI10.1007/s11071-017-4036-6.

[52] Dirk L. van Kekem and Alef E. Sterk, Wave propagation in the Lorenz-96 model, Nonlin. Processes Geophys., 25, 301–314, 2018
https://doi.org/10.5194/npg-25-301-2018

[53] A.I. Khibnik, LINBLF: A program for continuation and bifurcation analysis of equilibria up to codimension three. In D.Roose, B. De Dier and A. Spence (eds.) *Continuation and Bifurcation: Numerical Techniques and Applications,* vol. 313 of NATO Adv. Sci. Inst. Ser. C, Math. Phys. Sci., pages 283-296. Dordrecht 1990.

[54] R. Khoshsiar Ghaziani, W. Govaerts, Yu.A. Kuznetsov and H. G. E. Meijer, Numerical continuation of connecting orbits of maps in MATLAB, Journal of Difference Equations and Applications, 15 (8-9) 2009 pp. 849-875.
http://dx.doi.org/10.1080/10236190802357677
http://eprints.eemcs.utwente.nl/13858/

[55] R. Khoshsiar Ghaziani, Bifurcations of maps: numerical algorithms and applications, PhD thesis, Ghent University 2008.
http://dx.doi.org/1854/9725.

[56] Glenn E. Krasner and Stephen T. Pope, A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, Vol 1(3) Aug/Sept 1988, pp. 26-49.

[57] B. Krauskopf and H. Osinga, Growing 1D and Quasi-2D Unstable Manifolds of Maps, Journal of Computational Physics 146(1) 1998 pp. 404-419.
http://dx.doi.org/10.1006/jcph.1998.6059

[58] B. Krauskopf, H.M. Osinga and J. Galàn Vioque (Eds.) Numerical Continuation Methods for Dynamical Systems, Springer Complexity Series 2007.

[59] Yu. A. Kuznetsov, W. Govaerts, E.J. Doedel and A. Dhooge, Numerical periodic normalization for codim 1 bifurcations of limit cycles, SIAM J. Numer. Anal. 43 (2005) 1407-1435.

[60] Yu. A. Kuznetsov, Elements of Applied Bifurcation Theory, Springer-Verlag, 1998. (third edition 2004).

[61] Yu. A. Kuznetsov and V.V. Levitin, CONTENT: Integrated Environment for analysis of dynamical systems. CWI, Amsterdam 1997: `ftp://ftp.cwi.nl/pub/CONTENT`

[62] Max Lindmark, Magnus Huss, Jan Ohlberger and Anna Gardmark, Temperature-dependent body size effects determine population responses to climate warming, Ecology Letters 21(2), November 2017.
DOI10.1111/ele.12880

[63] D.W.C. Marcondes, G.F. Comassetto, B.G. Pedro, J.C.C. Vieira, A. Hoff, F. Prebianca, C. Manchein, C. Manchein, H.A. Albuquerque, Extensive Numerical Study and Circuitry Implementation of the Watt Governor Model Int. J. Bifurcation and Chaos 27(11) Oct. 2017, Article Number: 1750175
DOI: 10.1142/S0218127417501759

[64] https://sourceforge.net/projects/matcont/?source=directory

[65] D. Bindel, W. Govaerts, J. Hughes, Yu.A. Kuznetsov, M. Pekkér, and D. Veldman, CL_MATCONTL: Continuation Toolbox in MATLAB, October 2015.
https://www.uah.edu/science/departments/math/373-faculty/pekkermj/9996-software-development

[66] MATLAB, The Mathworks Inc., `http://www.mathworks.com`.

[67] MATLAB, Object-Oriented Programming (R2012a)
`www.mathworks.com/help/pdf_do/MATLAB/MATLAB_oop.pdf`

[68] Mattingly, Henry H.; Sheintuch, Moshe; Shvartsman, Stanislav Y., The Design Space of the Embryonic Cell Cycle Oscillator Biophysical Journal, Vol 113(3) pp. 743-752, Published Aug. 8, 2017.
https://doi.org/10.1016/j.bpj.2017.06.045

[69] W. Mestrom, Continuation of limit cycles in MATLAB, Master Thesis, Mathematical Institute, Utrecht University, The Netherlands, 2002.

[70] Morris, C., Lecar,H., Voltage oscillations in the barnacle giant muscle fiber,Biophys J. 35 (1981) 193–213.

[71] Niels Neirynck, Algoritmen voor de numerieke berekening van bifurcaties van afbeeldingen en hun implementatie in MATLAB, Master thesis, Ghent University 2012.

[72] Niels Neirynck, Bashir Al-Hdaibat, Willy Govaerts, Yuri A. Kuznetsov and H.G.E. Meijer, Using MatContM in the study of a nonlinear map in economics.
NOMA15 International workshop on nonlinear maps and applications, Dublin, Ireland, June 15-16, 2015.
Journal of Physics Conference Series Volume: 692 Article Number: 012013 Published: 2016, Edited by: Gelfreich, V; FournierPrunaret, D; LopezRuiz, R; et al. (eds.)

[73] N. Neirynck, W. Govaerts, Yu. A. Kuznetsov and H.G.E. Meijer, Numerical Bifurcation Analysis of Homoclinic Orbits Embedded in One-Dimensional Manifolds of Maps, ACM Transactions on Mathematical Software 44(3), Article 25, January 2018 (19 pages).
https://doi.org/10.1145/3134443

[74] Pandey, R (Pandey, Rakesh), Armitage, JP (Armitage, Judith P.) and Wadhams, GH (Wadhams, George H., Use of transcriptomic data for extending a model of the AppA/PpsR system in Rhodobacter sphaeroides. BMC SYSTEMS BIOLOGY 11 (Dec 2017),Article Number: 146.
DOI: 10.1186/s12918-017-0489-y

[75] J.D. Pryce, R. Khoshsiar Ghaziani, V. De Witte and W. Govaerts, Computation of normal form coefficients of cycle bifurcations of maps by algorithmic differentiation, Mathematics and Computers in Simulation 81 (2010) 109-119. ISSN: 0378-4754, DOI: 10.1016/j.matcom.2010.07.014.

[76] T. Puu, The chaotic monopolist, Fractals 1995:5(1), pp. 35-44.

[77] Michael Raatz, Ursula Gaedke and Alexander Wacker, High food quality of prey lowers its risk of extinction, Oikos 000: 001–010, 2017 doi: 10.1111/oik.03863.

[78] L. Razon, Stabilization of a CSTR in an Oscillatory State by Varying the Thermal Characteristics of the Reactor Vessel, International Journal of Chemical Reactor Engineering 4(1), January 2006. DOI10.2202/1542-6580.1320

[79] A. Riet, A Continuation Toolbox in MATLAB, Master Thesis, Mathematical Institute, Utrecht University, The Netherlands, 2000.

[80] D. Roose et al., Aspects of continuation software, in : Continuation and Bifurcations: Numerical Techniques and Applications, (eds. D. Roose, B. De Dier and A. Spence), NATO ASI series, Series C, Vol. 313, Kluwer 1990, pp. 261-268.

[81] Sahoo Bamadev, Panda Lokanath and Pohit Goutam, Combination, principal parametric and internal resonances of an accelerating beam under two frequency parametric excitation, International Journal of Non-Linear Mechanics 78(2016):35-44·
DOI: 10.1016/j.ijnonlinmec.2015.09.017

[82] Sandia National Laboratories, LOCA: Library of continuation algorithms for performing bifurcation analysis of large-scale applications, 2002.
http://www.cs.sandia.gov/LOCA/.

[83] Lakshmi N. Sridhar, Using Magnetic Nanoparticles to Eliminate Oscillations in Saccharomyces cerevisiae Fermentation Processes, Journal of Sustainable Bioenergy Systems, 2012, 2, 27-32
http://dx.doi.org/10.4236/jsbs.2012.23004
Published Online September 2012 (http://www.SciRP.org/journal/jsbs)

[84] Shyam Srinivasan, William R Cluett and Radhakrishnan Mahadevan, Model-based Design of Bistable Cell Factories for Metabolic Engineering, Bioinformatics, December 2017.
https://DOI10.1093/bioinformatics/btx769

[85] Straube, Ronny; Shah, Meera; Flockerzi, Dietrich; et al., Trade-off and flexibility in the dynamic regulation of the cullin-RING ubiquitin ligase repertoire Plos Computational Biology, Vol 13(11), Article Number: e1005869,Published: NOV 2017
https://doi.org/10.1371/journal.pcbi.1005869

[86] Hadi Taghfavard, Hildeberto Jardón Kojakhmetov and Ming Cao, Parameter-robustness analysis for a biochemical oscillator model describing the social-behaviour transition phase of myxobacteria, January 2018, Proceedings of the Royal Society A 474 (2209).
DOI10.1098/rspa.2017.0499.

[87] Clifford Henry Taubes, Modeling Differential Equations in Biology, Cambridge University Press, Cambridge, UK 2001, sec. ed. 2008.

[88] Terman, D., Chaotic spikes arising from a model of bursting in excitable membranes, Siam J. Appl. Math. 51 (1991) 1418–1450.

[89] Terman, D., The transition from bursting to continuous spiking in excitable membrane models, J. Nonlinear Sci. 2, (1992) 135–182.

[90] Nicolò Tomiati , Alessandro Colombo and Gianantonio Magnani, A nonlinear model of bicycle shimmy, Vehicle System Dynamics, International Journal of Vehicle Mechanics and Mobility. Published online: 26 Apr 2018 https://doi.org/10.1080/00423114.2018.1465574

[91] J.D. Touboul, A.C. Staver and S.A. Levin, On the complex dynamics of savanna landscapes, Proc. Nat. Acad. Sci., Jan 2018. DOI10.1073/pnas.1712356115

[92] Vanhulle, Laura Computationele detectie van homoclinische en heteroclinische connecties voor geïtereerde afbeeldingen, Master thesis, Ghent University, 2017. https://lib.ugent.be/en/catalog/rug01:002376270?i=0&q=Laura+Vanhulle

[93] Vora, Anuj S.; Sinha, Nandan K., Direct Methodology for Constrained System Analysis with Applications to Aircraft Dynamics JOURNAL OF AIRCRAFT, Volume: 54(6), pp. 2378-2385, (2017) https://doi.org/10.2514/1.c034264

[94] Wendi Wang, Dynamics of bacteria-phage interactions with immune response in a chemostat. Journal of Biological Systems 25(4), Dec. 2017, pp. 697-713. DOI: 10.1142/S0218339017400010

[95] Alan Wolf, Jack B. Swift, Harry L. Swinney, John A. Vastano, Determining Lyapunov exponents from a time series, Physica D: Nonlinear Phenomena, Volume 16, Issue 3, July 1985, Pages 285-317

[96] J.K. Wróbel and R.H. Goodman, High-order adaptive method for computing two-dimensional invariant manifolds of three-dimensional maps, Communications in Nonlinear Science and Numerical Simulation 18,7(2013), pp. 1734-1745.

[97] Chih-Hang John Wu, Zhenzhen Shi, David Ben-Arieh, Steven Q Simpson Mathematical Modeling of Innate Immunity Responses of Sepsis: Modeling and Computational Studies: From Data to Knowledge to Healthcare Improvement, August 2016 DOI10.1002/9781118919408.ch8 In book: Healthcare Analytics: From Data to Knowledge to Healthcare Improvement, eds. Hui Yang and Eva K. Lee, Wiley 2016 ISBN: 978-1-118-91939-2

[98] Liuyang Xiong, Lihua Tang, Kefu Liu and Brian R. Mace1, Broadband piezoelectric vibration energy harvesting using a nonlinear energy sink, Journal of Physics D Applied Physics · March 2018 DOI: 10.1088/1361-6463/aab9e3

[99] Gang Zhao, Dagmar Wirth, Ingo Schmitz and Michael Meyer-Hermann: A mathematical model of the impact of insulin secretion dynamics on selective hepatitic insulin resistance. Nature Communications, Volume 8, Article number: 1362 (2017).
doi:10.1038/s41467-017-01627-9

Appendix A

TUTORIAL I: Using the new MATCONT GUI for numerical integration of ODEs

This session was tested on MATCONT7p1 with MATLAB2017b. It illustrates how to input a system of autonomous ordinary differential equations (ODEs)

$$\dot{x} = f(x, \alpha), \quad x \in \mathbb{R}^n, \alpha \in \mathbb{R}^m,$$

into MATCONT and numerically integrate it with simultaneously visualizing orbits in graphic windows.

It is not possible to load into MATCONT a system without any parameters. However, you can introduce a dummy variable to perform time-integration.

We will study the *Rössler chaotic system*:

$$\begin{cases} \dot{x} &= -y - z \\ \dot{y} &= x + Ay \\ \dot{z} &= Bx - Cz + xz, \end{cases}$$

where (x, y, z) are the phase variables, and (A, B, C) are the parameters.

A.1 Getting started

We assume that MATCONT is placed into the directory MATCONT of your system and has been installed properly[1]. Start MATLAB and change the current directory to MATCONT. Start MATCONT by typing

```
matcont
```

[1]With this we refer to the compilation of the C-files in the subdirectory `LimitCycle`; for details see documentation at `https://sourceforge.net/projects/matcont/`

in the MATLAB command line window and press Enter.

You will get several windows related to a default ODE system, like in Figure A.1. The position of the windows might be rearranged if they do not fit the screen.

The main window is called **MatCont** and has several menus. For instance, to end your MatCont session, choose the **Exit** item in the **Select** menu. Hereafter this operation will be indicated with **Select**|**Exit**.

Note. It is sometimes desirable not to load a previously used system (e.g. because the last used system was in some way corrupted). Then by typing

```
matcont clean
```

one gets a fresh start with a blank main MatCont window.

A.2 Input new system

The command **Select**|**System**|**New** opens the **System** window, which contains several fields and buttons. To identify the system, type for example

```
Rossler
```

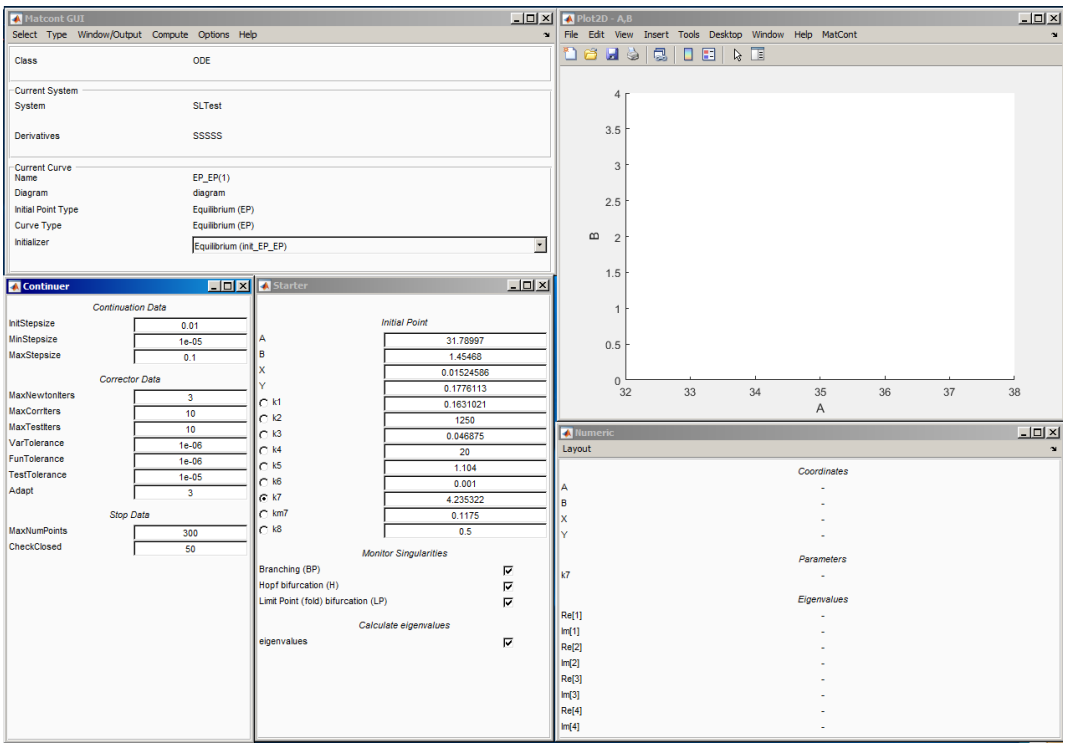


Figure A.1: A typical MatCont startup screen.

in the **Name** field (it must be one word).

Input names of the **Coordinates**: `x,y,z`, the **Parameters**: `A,B,C`, and the name for **Time**: `t` (default).

If shown, select symbolic generation of the 1st order derivatives by pressing the corresponding radio-button [2].

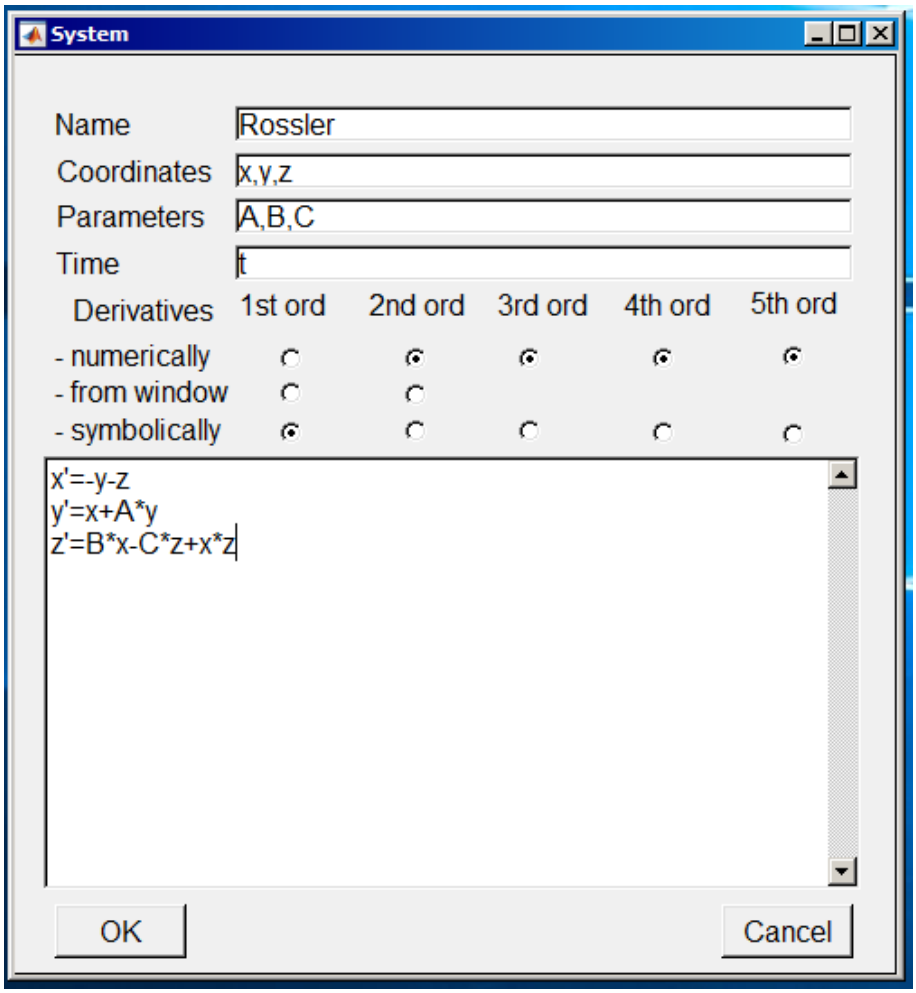


Figure A.2: Specifying a new model.

Finally, in the large input field, type the RHS of Rössler's system as

```
x'=-y-z
y'=x+A*y
```

[2]If the MATLAB Symbolic Toolbox is present, there will be buttons indicated 'symbolically'. The first-order derivatives are used in some of the integration algorithms, the first- and second-order derivatives are used in the continuation, while the third-order derivatives are employed in the normal form computations.

```
z'=B*x-C*z+x*z
```

Avoid typical mistakes:

- Make sure the multiplication is written explicitly with $*$.
- Specify differential equations in the same order as the coordinates.

Now the **System** window should look like in Figure A.2, and you can press **OK** button.

If you made no typing mistakes, the **System** window disappears, and you will see in the main MatCont window that `Rossler` becomes the current **System** of MatCont. If selected, the information field **Derivatives** shows the string `SNNNN` meaning that the symbolic 1st order derivatives of the RHS will be used.

If you want to change or correct an existing system, click **Select**|**Systems**|**Edit/Load/Delete**, select this system in the appearing **Systems** window, and press **Edit** button. The inputed system can be completely erased by selecting **Action**|**Delete** there.

A.3 Selection of solution type

To tell MatCont that we want to integrate the system, i.e. to numerically solve the initial value problem for it, we have to specify the type of the initial point and the type of the curve to compute. To select the initial point type, input **Type**|**Initial Point**|**Point**, which means that the initial point has no special properties. To select the curve type, click **Type**|**Curve**|**Orbit**[3]. The information fields in the MatCont main window will reflect the selections, and two more windows appear. These are the corresponding **Starter** and **Integrator** windows. Move them, if necessary, to make both visible.

A.4 Setting initial data for integration

The **Starter** window is curve-dependent and is used here to specify initial data for the integration. If any field (cf. Figure A.3) seems to be missing, then press **Type**|**Initial point**|**Point** again. Let us input the following initial coordinate and parameter values:

```
x           -5.0
y            5.0
z           10.0

A           0.0
B           0.4
C           4.5
```

[3]Actually, it is default.

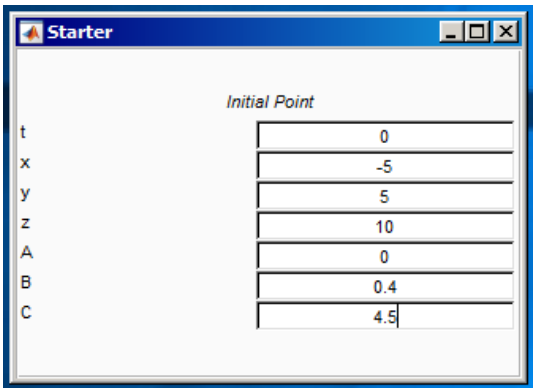


Figure A.3: Starter window for integration.

(see Figure A.3).

We will use the default numerical parameters of the **Integrator**, except of the `Interval`. Input in the **Integrator** window:

```
Interval     100
```

to get Figure A.4. Check that the default method of integration is `ode45`. You can inspect available methods by clicking the **Method** menu button.

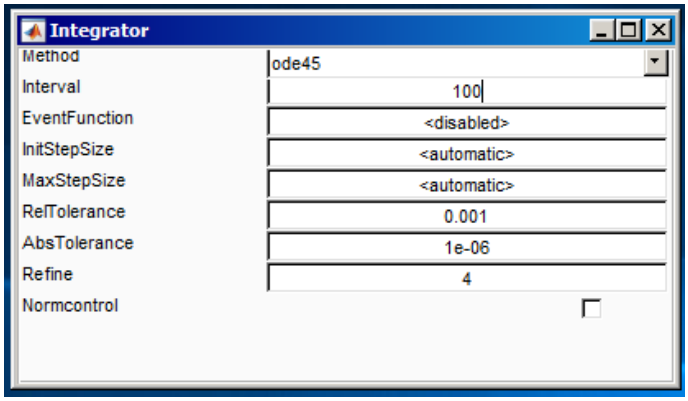


Figure A.4: Integrator window.

A.5 3D visualization

To visualize the orbits, we have to open at least one more window and setup plotting attributes.

A.5.1 3D graphic window

In the main MatCont window, select **Window/Output|Graphic|3D plot**. The first **Plot3D** window appears.

Plot3D windows are used to represent solution curves by their projections to 3-dimensional spaces endowed with right-handed rectangular Cartesian coordinate systems. 3D space is specified by three variables whose values are plotted along axes. The visible part of the space is given by minimum and maximum values for each axis. By default, the visibility limits are from 0 to 1 for all axes.

A.5.2 Variables on axes

First, we need to choose the variables along the axes and the ranges of displayed values along these axes. This can be done by selecting the coordinates `x`, `y`, and `z` and the ranges

$$-9 \leq x \leq 9,\ -9 \leq y \leq 9,\ -2 \leq z \leq 11.$$

in the **Layout** window (see Figure A.5). This window is accessible via the menu **MatCont|Layout** in the **Plot3D** window. Click **OK** to confirm the choice of variables and ranges.

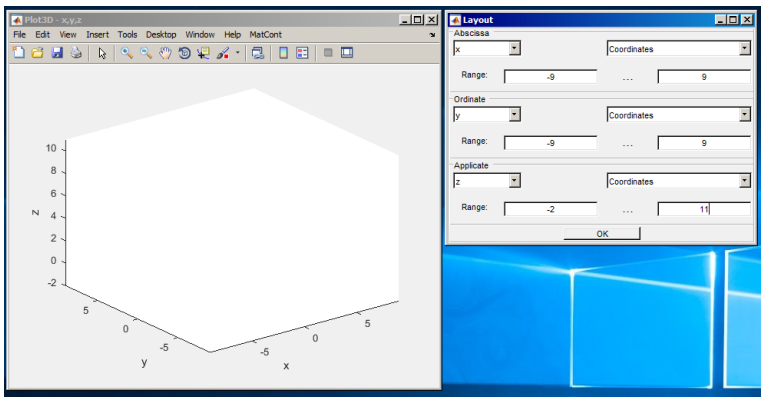


Figure A.5: Selecting variables and ranges along the axes.

The graphics window **Plot3D** is updated automatically with the input and the new names of the axes become visible.

A.6 Integrating orbits

Now we are ready to start numerical integration. We will perform multiple simulations of the Rössler system at different values of a parameter. We will see that the behaviour of the orbits changes qualitatively, suggesting that bifurcations happen in between these parameter values.

A.6.1 Start computation

Input the **Compute|Forward** menu command in the main window to start integration. An **Output** window is opened automatically. The computation can be paused or stopped by clicking the corresponding button in that window. If paused, the computation can be resumed by clicking the corresponding button as shown in Figure A.6.

Also, you can interrupt the computation at any time by pressing the **Esc** key on the keyboard.

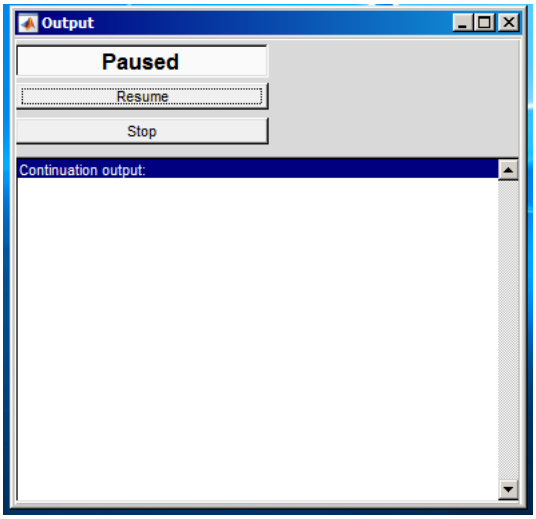


Figure A.6: Output window.

The integration will start and shortly after produces Figure A.7. The orbit tends to a stable equilibrium.

When the computation is finished, the **Output** window looks like Figure A.8. You can now either close the **Output** window or press the **View Result** button to inspect the output in a **Data Browser** window, see Figure A.9. By pressing the **View CurveData** button in the **Data Browser** window one obtains a spreadsheet view of all points computed during the time integration, see Figure A.10.

A.6.2 Other parameter and coordinate values

Clear the **Plot3D** window by selecting **MatCont|Clear** menu option there. Change the parameter `A` value to 0.2 – while keeping all other parameters unchanged – and the initial value of `z` to 1. Start the new integration by **Compute|Forward**. The computed orbit now tends to a stable periodic orbit (*limit cycle*), see Figure A.11. This stability loss by the equilibrium with the generation of a limit cycle is called the (supercritical) *Andronov-Hopf bifurcation*.

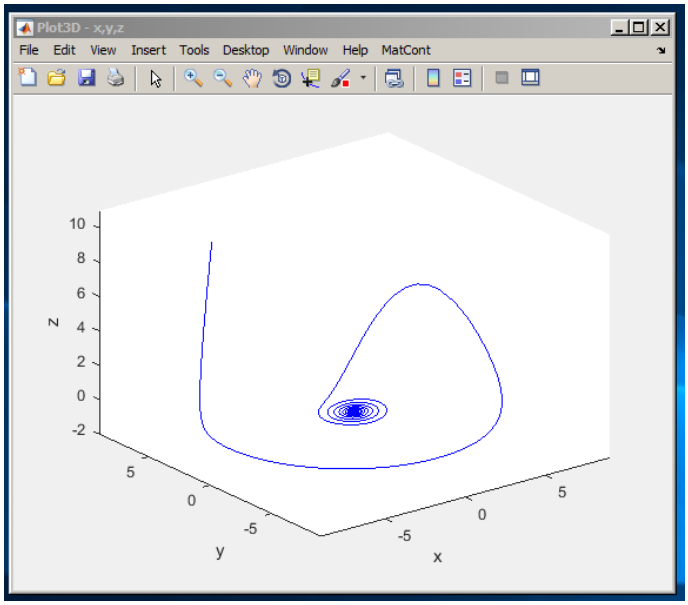


Figure A.7: An orbit in the **Plot3D** window converging to a stable equilibrium.

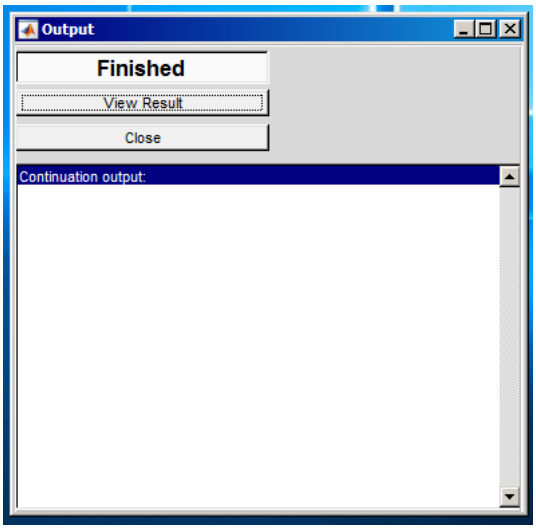


Figure A.8: The **Output** window when the computation is finished.

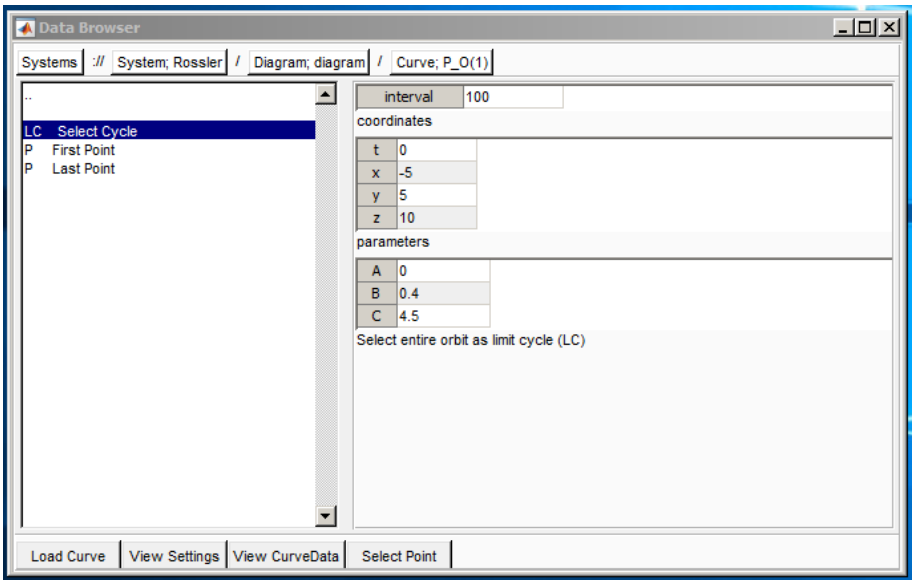


Figure A.9: Inspect the output of the computation in a **Data Browser** window.

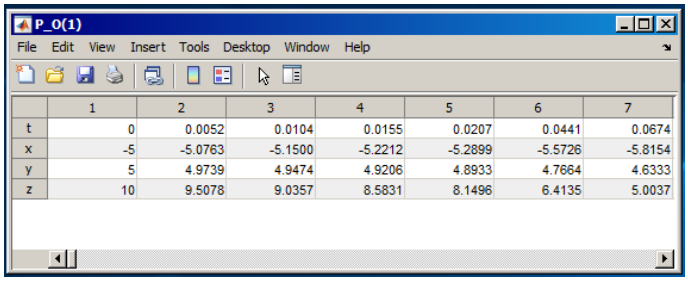


Figure A.10: A spreadsheet view of the points computed by time integration.

Repeat the integration at `A` equal 0.25. You should get Figure A.12 showing another orbit tending to a limit cycle but slowly.

When we now repeat the integration for `A` equal to 0.3, we notice that the orbit tends to a limit cycle making two turns before closure. The appearance of a stable cycle with approximately doubled period, while the primary cycle becomes unstable, is called the (supercritical) *period-doubling bifurcation*. To erase transient behaviour, select **MatCont|Clear** in the **Plot3D** window and then continue the integration via the menu selection **Compute|Extend** in the main window. You should obtain Figure A.13 (left panel after the initial integration and right panel without showing the transient).

Increasing the parameter `A` to 0.315, results in a more complicated cycle making four global turns before closure, see and reproduce Figure A.14. We can conclude that the doubled cycle undergoes the next period-doubling bifurcation from a cascade of such bi-

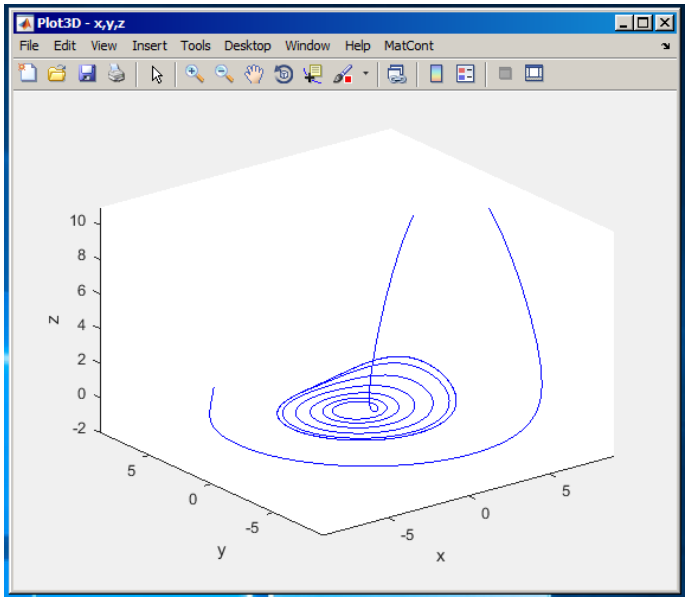


Figure A.11: An orbit in the **Plot3D** window converging to a stable limit cycle at $A = 0.20$.

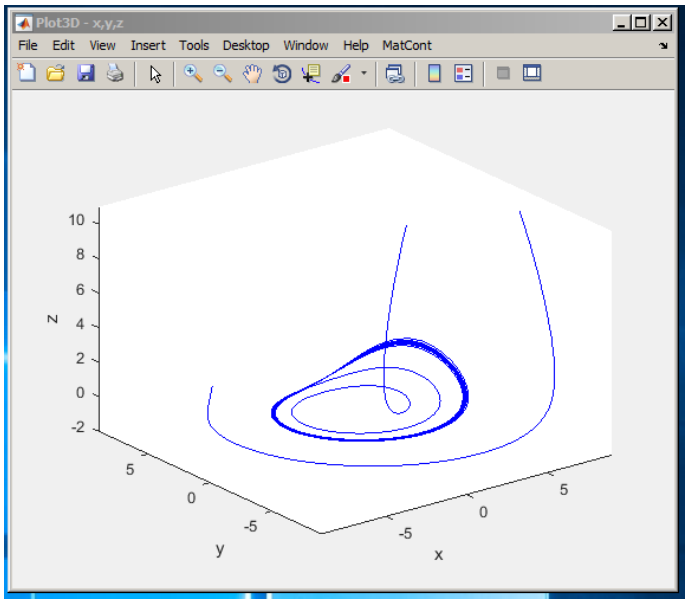


Figure A.12: An orbit in the **Plot3D** window converging to a stable limit cycle at $A = 0.25$.

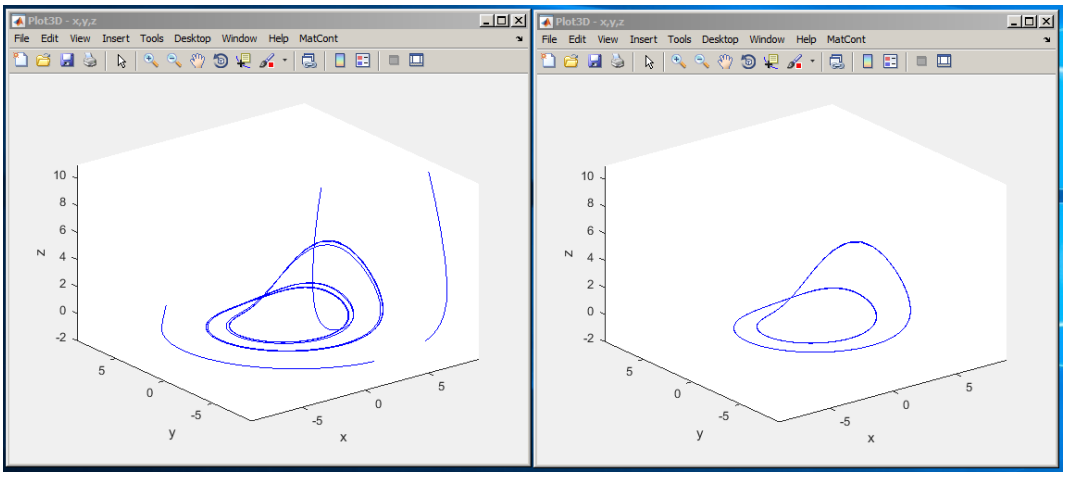


Figure A.13: An orbit in the **Plot3D** window converging to a stable 2-cycle at $A = 0.3$.

furcations.

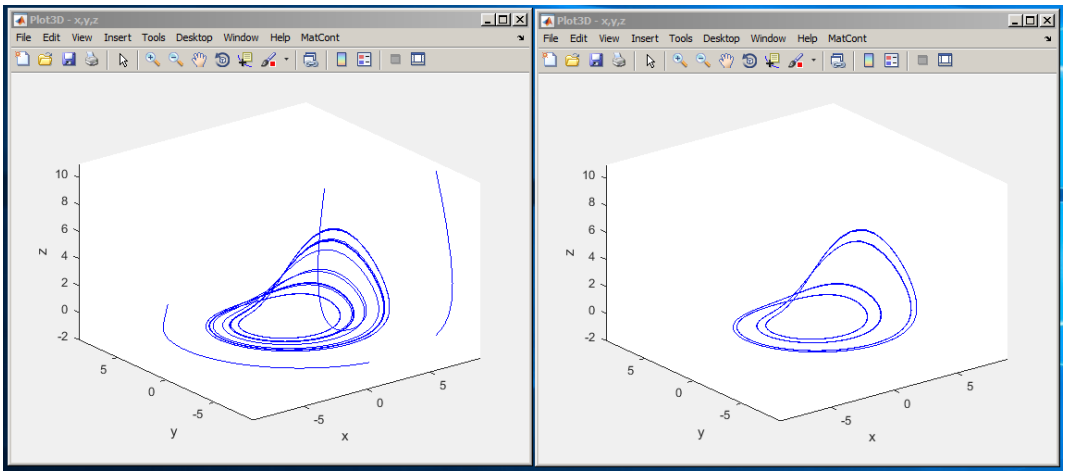


Figure A.14: An orbit in the **Plot3D** window converging to a stable 4-cycle at $A = 0.315$.

Finally, change `A` into 0.36, set the **Interval** in the **Integrator** window to 500, clear the graphic window, and start computation. After some time, you should see a *chaotic attractor* as in Figure A.15.

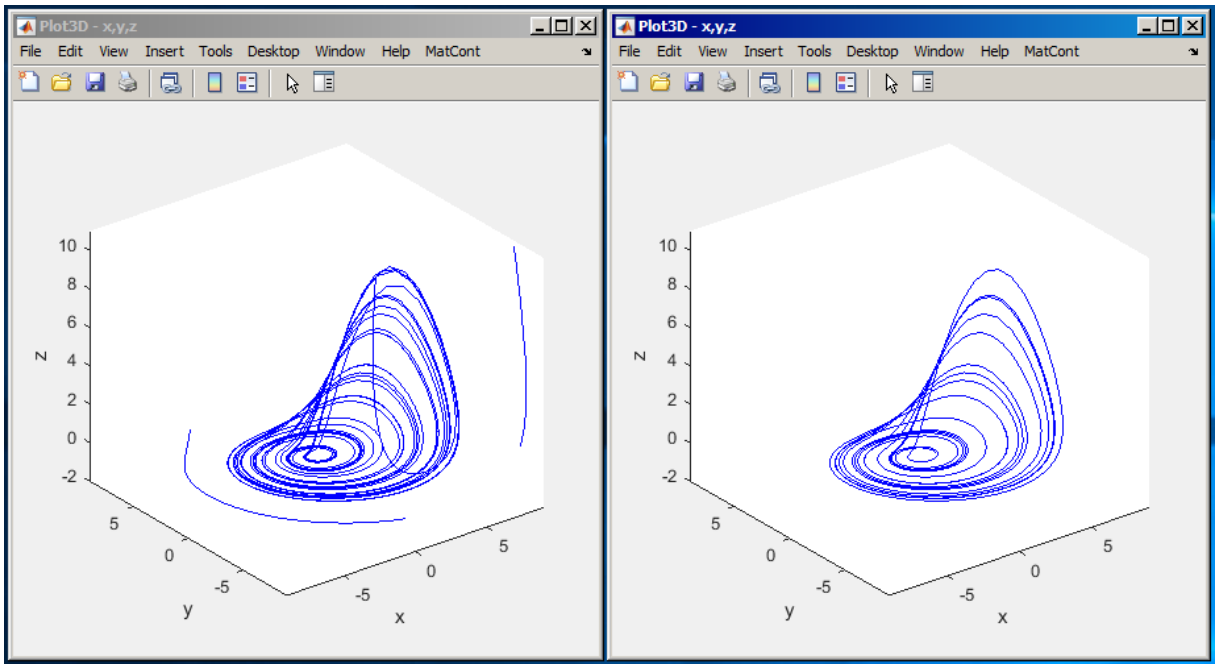


Figure A.15: A chaotic attractor in the **Plot3D** window at $A = 0.36$.

A.7 Plot manipulation

A.7.1 Another viewpoint

You can change the viewpoint in the **Plot3D** window using the standard MATLAB facilities. If you press the **Rotate 3D** button in the Tools section of the menu bar of the **Plot3D** window, then the coordinate system can be rotated with the help of the mouse. A rotated orbit is shown in Figure A.16.

To view the orbit in the original projection, use the standard MATLAB tools.

A.7.2 Redraw

The computed orbit (including the transient) can be redrawn without recomputation using commands from the **Plot3D** window, namely: **Plot|Clear** followed by **Plot|Redraw Curve**.

A.7.3 Export a figure

You can save the produced figure in various formats using the standard MATLAB tools. For example, to produce an Encapsulated PostScript file corresponding to Figure A.16, click **File|Save As..** button in the **Plot3D** window and select **EPS** type. Then specify the file name and location in the appearing dialog box, and press **Save**.

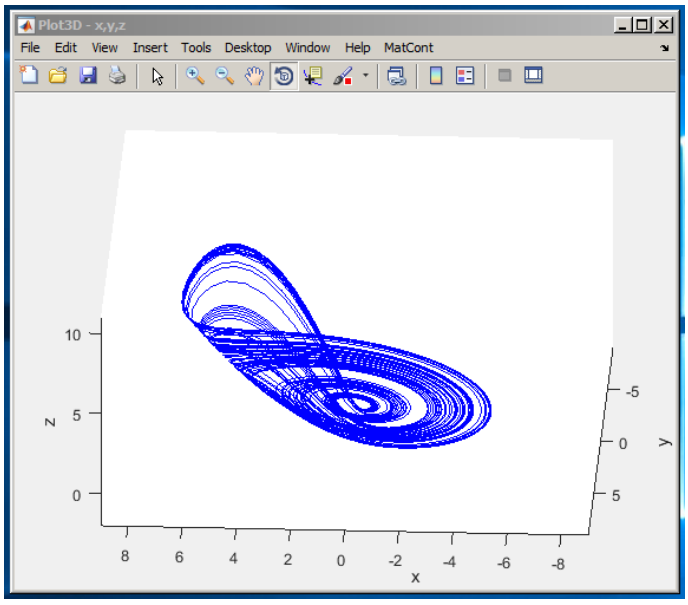


Figure A.16: Another viewpoint on the chaotic attractor at $A = 0.36$.

A.8 2D visualization

Let us plot the time series corresponding to the computed chaotic orbit for $0 \leq t \leq 250$. For this, open another graphic window via the **Window/Output|Graphic|2D plot** in the main MatCont window.

A new window **Plot2D** appears that has a structure similar to the **Plot3D** window.

Using **MatCont|Layout**, set time `t` as the **Abscissa** and coordinate `x` as the **Ordinate** and input new visibility limits along the axes:

```
t          0    250
x         -8    8
```

The layout window is presented at the right of Figure A.17.

Close the **Plot3D** window. In the **Plot2D** window, you can get with **MatCont|Redraw Curve** a time series shown in Figure A.18. You may need to resize the window for a better view.

A.9 Another method of integration

The method of integration can be changed. For example, one can select `ode23s` as **Method** in the **Integrator** window. After selection, the **Integrator** window will be updated.

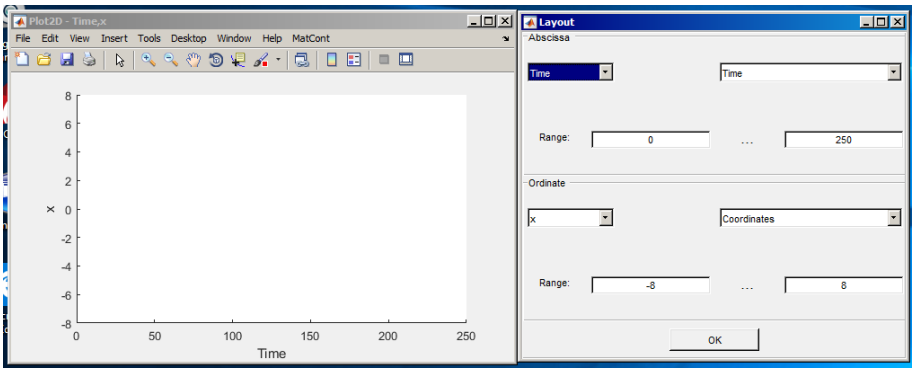


Figure A.17: Coordinates and range limits in a 2D plot.

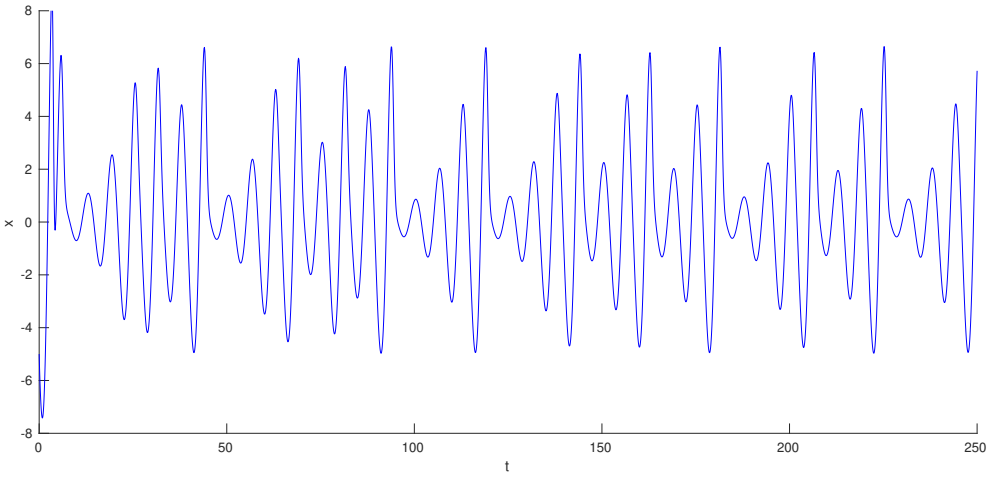


Figure A.18: The first solution component versus time in the **Plot2D** window.

Repeat computations with **Compute|Forward**. Notice in Figure A.19 a change in the time series for big values of t due to a difference in the accuracy of the methods. This shows that one has to be careful with numerical solutions of ODEs.

Note: Normally you will get Figure A.19 in one color. We used a property editor to change the color of the second curve.

A.10 Archive of computed solutions

By default, MatCont keeps only a certain number of computed curves (e.g. orbits) under the standard names corresponding to their **Point type** and **Curve type**. This number is determined by the Archive Filter which can be seen and adapted by the user by pressing

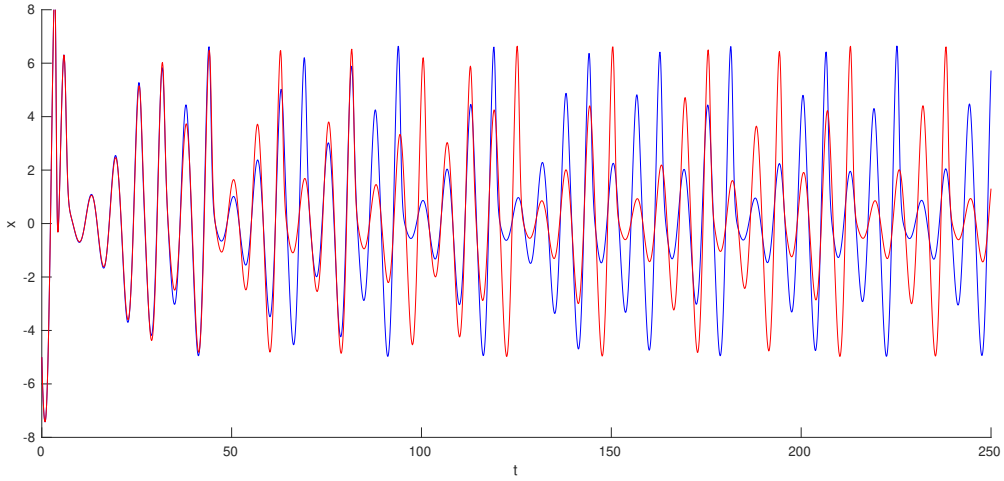


Figure A.19: Two numerical solutions obtained with different integration methods. The second curve was computed with `ode23s`. The color of the second curve has been set to red to accentuate the difference.

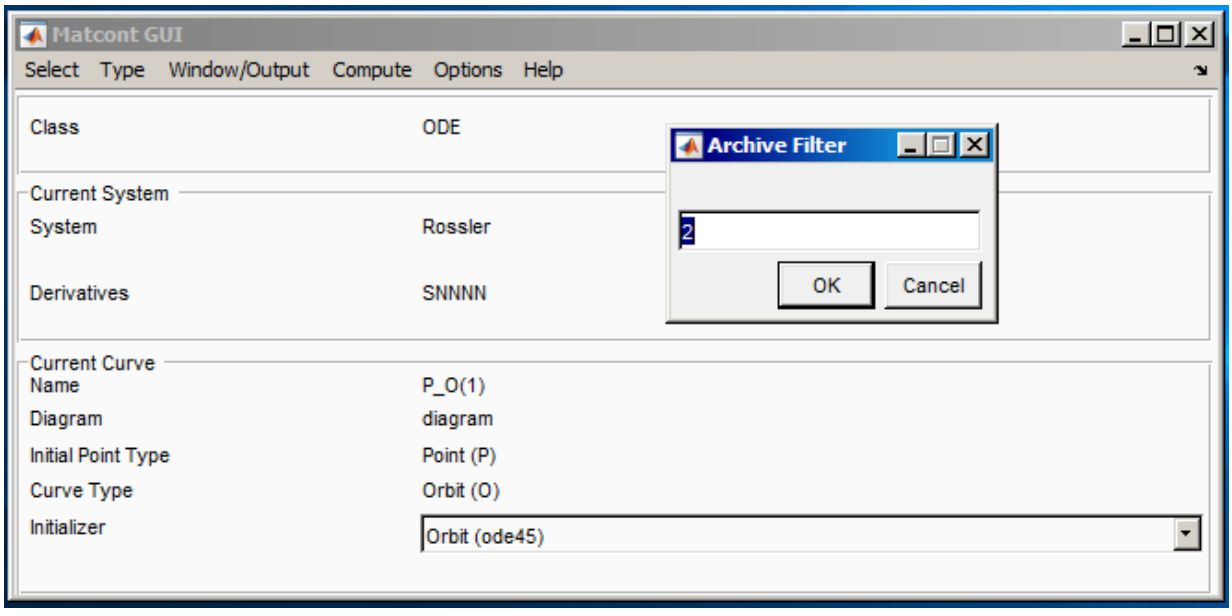


Figure A.20: The Archive Filter is set to 2.

the **Options|Archive Filter** button, see Figure A.20. With this setting, only two curves of each type are preserved.

You can manipulate the curves via **Select|Curve** menu in the main MatCont window. This opens a **Data Browser** window as displayed in Figure A.21.

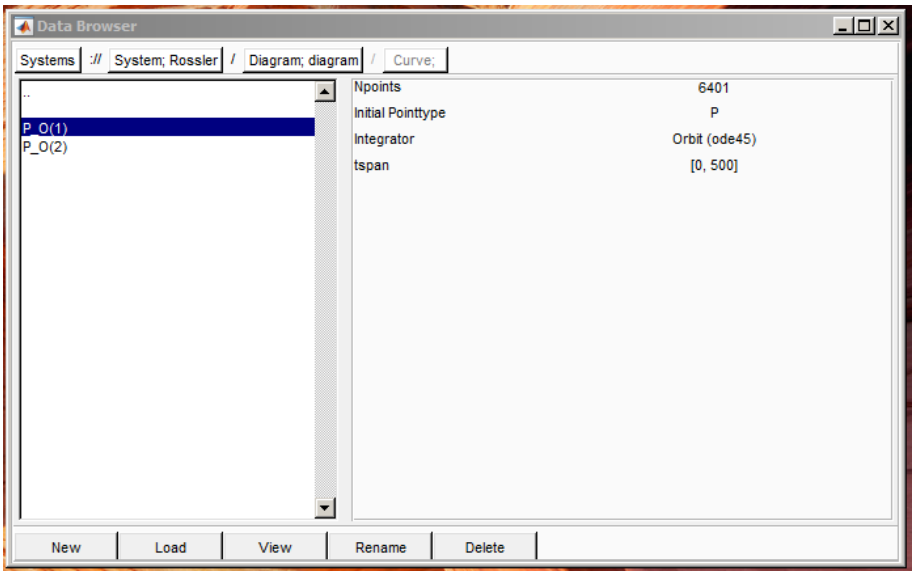


Figure A.21: The **Data Browser** window opened by pressing the **Systems|Curve** button.

It is now possible to load a curve (for example to redraw it separately), view, rename or delete it.

As an exercise, inspect a spreadsheet view of the computed points of the highlighted curve by pressing the **View** button in the **Data Browser** window.

If a curve is renamed, it will be stored permanently. As an exercise rename the two remaining orbits into `orbit1` and `orbit2`, respectively. By pressing the **New** button one returns to an empty curve in the diagram that is open in the **Data Browser** window.

Close the **Data Browser** window.

All currently available curves form a *diagram* that can be redrawn in a graphic window via **MatCont|Redraw Diagram**. If you do this in the **Plot2D** window, you should get Figure A.19 again.

A.11 Additional Problems

A. Consider the *logistic equation*

$$\dot{x} = x\left(1 - \frac{x}{\alpha}\right)$$

with positive α, e.g. $\alpha = 2$. Plot solutions $(t, x(t))$ with different initial conditions $x(0)$.

B. *Van der Pol's equation* is given by

$$\ddot{x} - \alpha(1 - x^2)\dot{x} + x = 0.$$

Rewrite the equation as a system of two first-order ODEs by introducing $y = -\dot{x}$. Plot its orbits for $\alpha = 0$ and $\alpha = 0.5$ in the phase space (x, y).

C. Simulate the *Lorenz chaotic system*

$$\begin{cases} \dot{x} &= \sigma(y - x), \\ \dot{y} &= rx - y - xz, \\ \dot{z} &= -bz + xy, \end{cases}$$

for $\sigma = 10,\ b = \frac{8}{3},\ r = 28$. Maks 3D phase plots as well as plot time-series of z for various initial conditions.

D. Consider the following *Langford system*

$$\begin{cases} \dot{x} &= (\lambda - b)x - cy + xz + dx(1 - z^2), \\ \dot{y} &= cx + (\lambda - b)y + yz + dy(1 - z^2), \\ \dot{z} &= \lambda z - (x^2 + y^2 + z^2), \end{cases}$$

where $b = 3,\ c = \frac{1}{4}$ and $d = \frac{1}{5}$.

1. Integrate this system in MatCont for $\lambda = 1.5, 1.9, 2.01$. Always start from $x_0 = y_0 = 0.1, z_0 = 1$.
2. Briefly describe what you see in each case.
3. Could you support your conclusions by some other numerical and/or analytical arguments ?

Appendix B

TUTORIAL II: Using the new MATCONT GUI for one-parameter bifurcation analysis of equilibria

This session was tested on MATCONT7p1 with MATLAB2017b. It is devoted to the numerical continuation of equilibria in systems of autonomous ODEs depending on one parameter

$$\dot{u} = f(u, \alpha), \quad u \in \mathbb{R}^n, \alpha \in \mathbb{R},$$

and detection of their bifurcations.

B.1 An ecological model with multiple equilibria and limit points

Consider the following system appearing in mathematical ecology:

$$\begin{cases} \dot{x} &= rx(1-x) - \dfrac{xy}{x+a}, \\ \dot{y} &= -cy + \dfrac{xy}{x+a} - \dfrac{dy^2}{y^2+b^2}. \end{cases} \tag{B.1}$$

Fix

$$r = 2, \quad a = 0.6, \quad b = c = 0.25,$$

and consider d as a bifurcation parameter with initial value $d = 0.1$. The aim is to locate equilibria of (B.1) and study their dependence on d.

B.1.1 System specification

Start MATCONT, choose **Select**|**System**|**New**, and input a new ODE system – say `EcoMod` – into MATCONT as shown in Figure B.1.

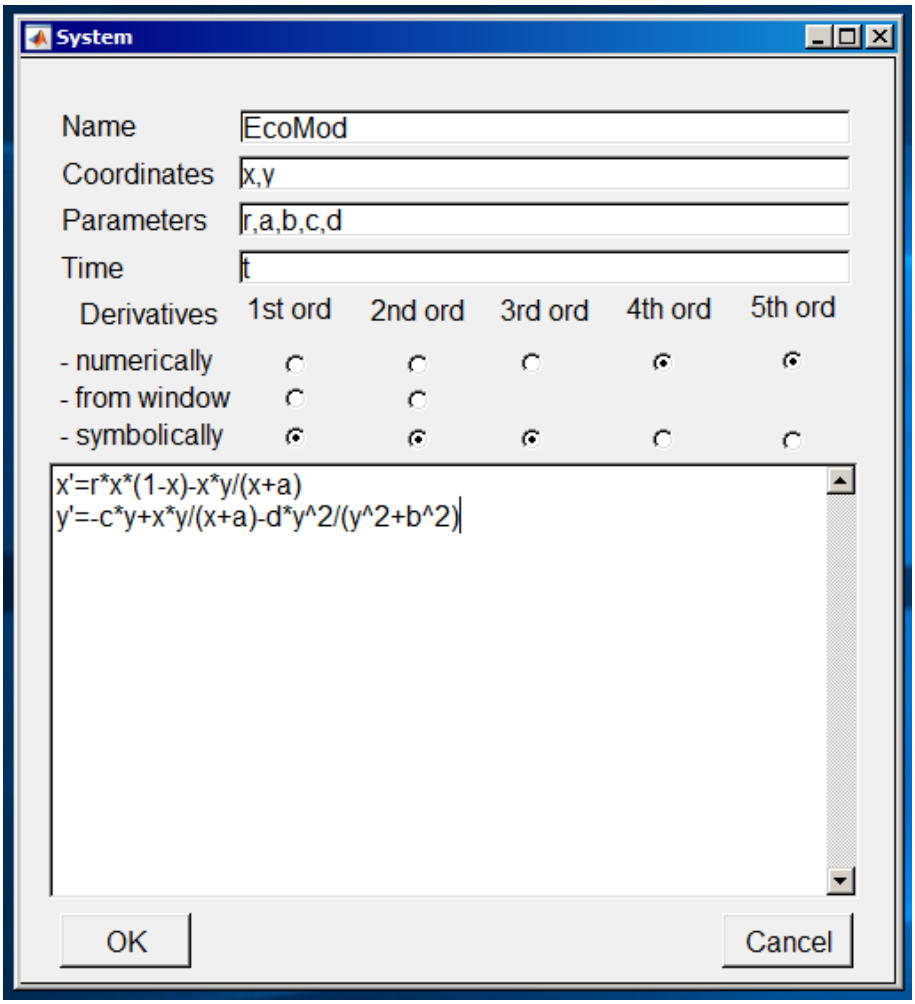


Figure B.1: The ecological model.

B.1.2 Locate an equilibrium by numerical integration

Select **Type|Initial point|Point** and check that the default **Type|Curve|Orbit** is selected.

In the appearing **Integrator** window, increase the integration `Interval` to 200 and set `RelTolerance` to `1e-7` and `AbsTolerance` to `1e-10` (to assure smaller integration steps).

Via the **Starter** window, input the initial point

```
x          1.2
y           1
```

and the initial values of the parameters, namely

```
r            2
```

```
a           0.6
b           0.25
c           0.25
d           0.1
```

Open a **2Dplot** window with **Window/Output|Graphic|2Dplot** and a **Layout** window by clicking **MatCont | Layout** in the **2Dplot** window. Select `x` and `y` as variables along the corresponding axes and the following visibility limits:

```
Abscissa:       0       1.2
Ordinate:       0       2.0
```

as shown in Figure B.2.

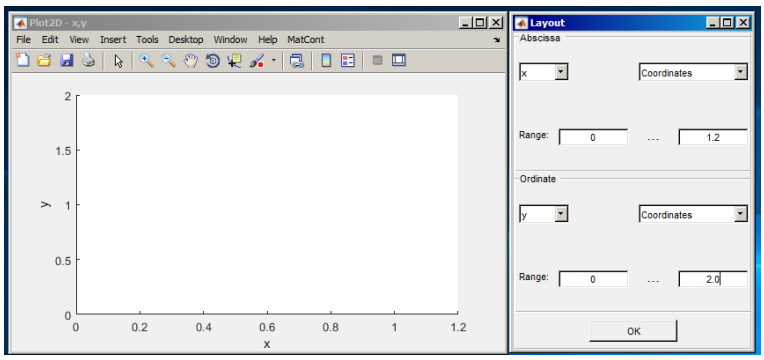


Figure B.2: Opening a **2Dplot** window.

Start **Compute|Forward**. You will get an orbit converging to an equilibrium, see Figure B.3

To see how the coordinates vary along the orbit, select **Window/Output|Numeric** which opens a **Numeric** window. Then input another initial point in the **Starter** window

```
x           1.0
y           0.001
```

and **Compute|Forward**. You get the second orbit from Figure B.3 tending to the same equilibrium, whose (approximate) coordinates can be seen in the **Numeric** window shown in Figure B.4. Close both the **Numeric** window and the **Output** window.

B.1.3 Equilibrium curve

We can continue the equilibrium found by integration with respect to the parameter d. Open the **Data Browser** window by pressing **Select|Initial Point** in the main MatCont window. Then click on **Last Point** and **Select Point**, as shown in Figure B.5.

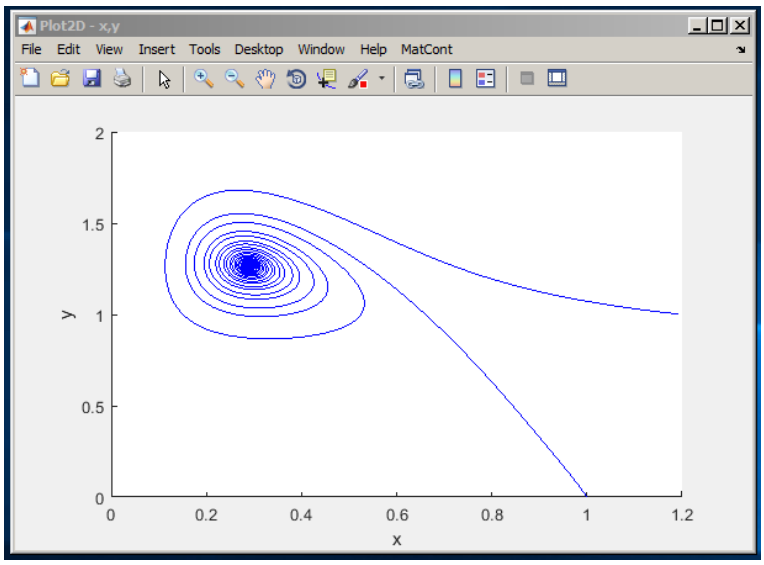


Figure B.3: Phase orbits approaching a stable equilibrium for $d = 0.1$.

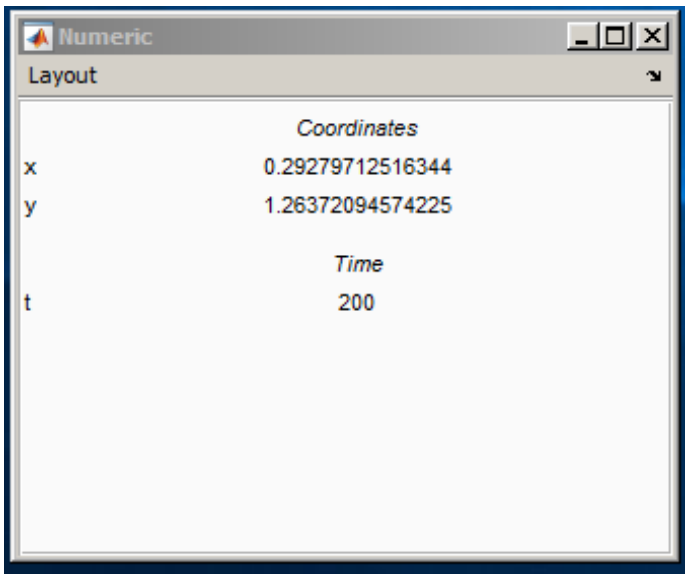


Figure B.4: The coordinates of the stable equilibrium.

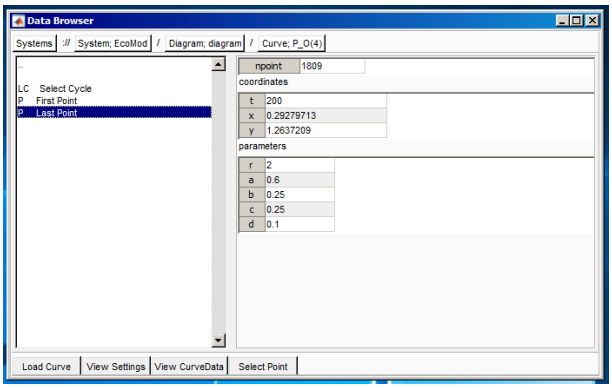


Figure B.5: **Data Browser** window with **Last Point** selected.

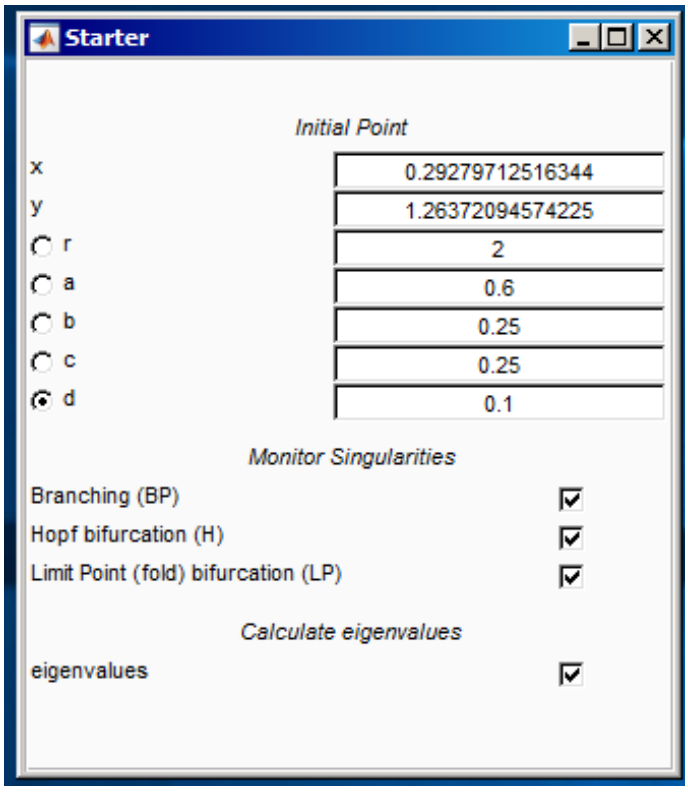


Figure B.6: **Starter** window for the equilibrium continuation: Parameter `d` is activated.

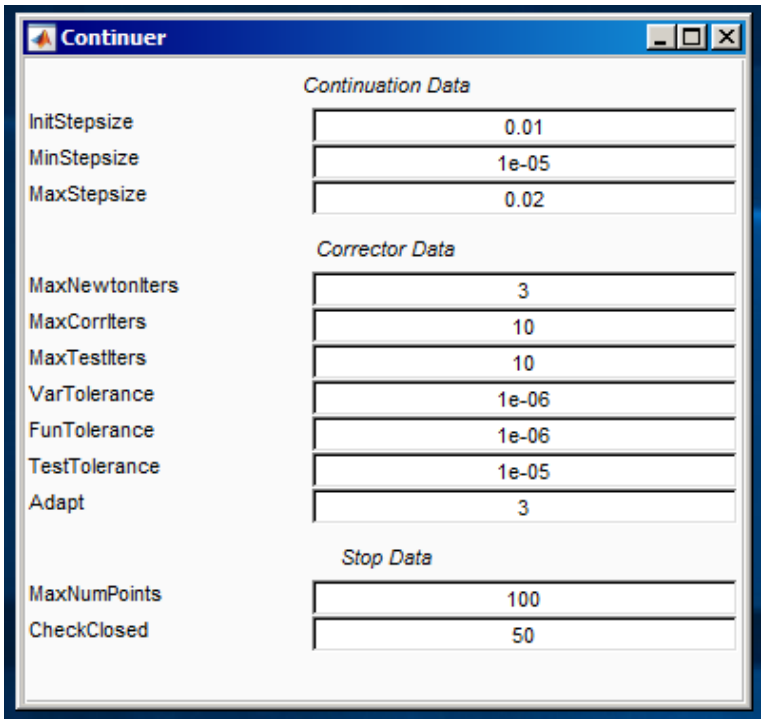


Figure B.7: Continuation parameters in the **Continuer** window.

To tell MatCont to continue an equilibrium curve, we have to specify the initial point type and the curve type. Selecting **Type|Initial Point|Equilibrium** we set the point type `EP` and the same (default) curve type, so that we prepare to compute the `EP_EP` equilibrium curve, as indicated in the main MatCont window. The **Starter** window is now modified and the **Integrator** window is replaced by a **Continuer** window.

The initial values of the parameters and the equilibrium coordinates are visible in the **Starter** window. Press the radio-button next to parameter `d`, indicating that it will be a bifurcation parameter (i.e. *activate* parameter d). The resulting **Starter** window is shown in Figure B.6. There you can also see which singularities will be monitored (all), and whether the equilibrium eigenvalues will be computed (yes).

In the **Continuer** window, several default numerical parameters related to the continuation are listed. Change only

```
MaxStepsize          0.02
MaxNumPoints         100
```

(see Figure B.7).

Use **MatCont | Layout** menu in the **2Dplot** window to select the parameter `d` as abscissa and the coordinate `y` as ordinate with the visibility limits:

```
Abscissa:            0              0.5
```

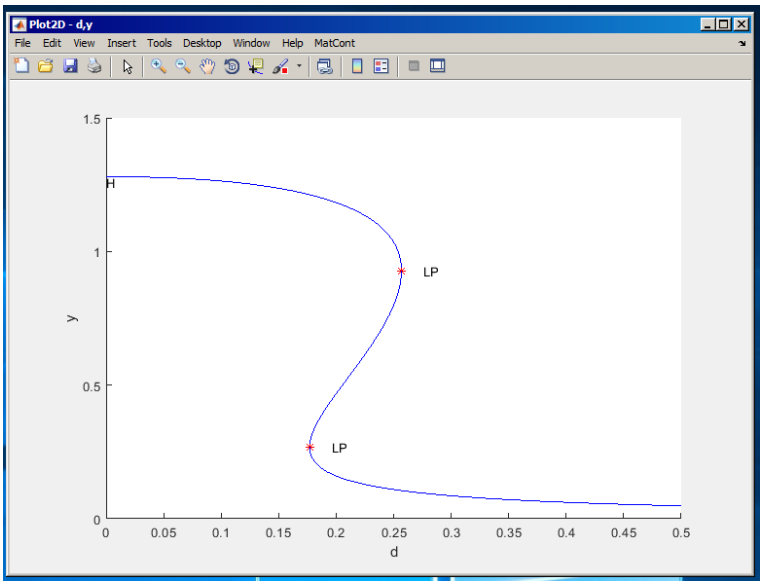


Figure B.8: Equilibrium manifold in the (d, y)-plane: `LP`'s denote limit points, `H` is a Hopf point.

```
Ordinate:          0              1.5
```

and clear the window by pressing **MatCont—Clear** in the **2DPlot** window.

Now click **Compute|Forward** to get part of a sigmoidal curve as in Figure B.8. During the computations an **Output** window is opened and the continuation is stopped each time when a singularity is found, see Figure B.9. Then press **Resume** in the **Output** window to restart the continuation. Now click **Compute|Backward** to complete the sigmoid.

Along the forward branch, MatCont stops at two *limit points* labeled `LP`. You can `Resume` the computation at these points in the **Output** window.

After finishing this continuation press the **View Result** button in the **Output** window. This opens a **Data Browser** window. Press the **View** button at the bottom of the **Data Browser** window. This opens a spreadsheet view of all computational values that will be stored as data of the curve. Figure B.10 shows part of this spreadsheet; the **Curve Data** correspond to the `x,v,s,h,f` output of the continuer as described in the MatCont manual.

Along the backward branch, MatCont stops at a *Hopf point* labeled `H`. Terminate the computation by pressing `Stop` there.

To determine stability of the equilibria and read the bifurcation parameter values, open **Window/Output|Numeric** and select **Layout** in the **Numeric** window.

The **Layout** window offers the choice to visualise during the continuation the coordinates (i.e. state variables), active parameters, testfunctions, eigenvalues, current stepsize, userfunctions (when userfunctions are present), and number of computed points. Activate only the coordinates, parameters and eigenvalues.

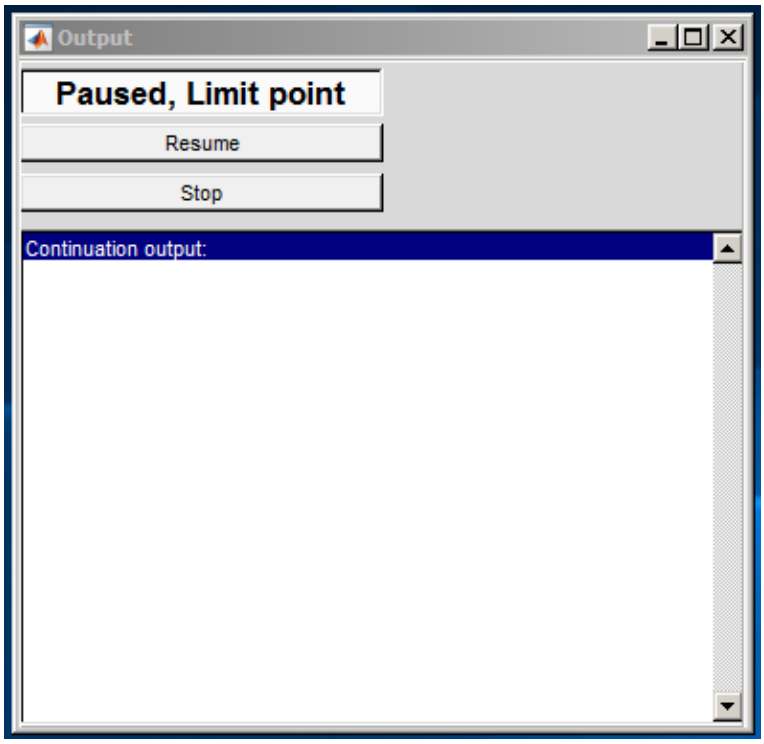


Figure B.9: The **Output** window when paused at an LP point.

This makes the state variables, active parameters and eigenvalues of the equilibrium visible in the **Numeric** window (upon resizing it).

Clear the **2Dplot** window with **MatCont**|**Clear** and repeat **Compute**|**Forward**. In the **Numeric** window you can read the LP parameters:

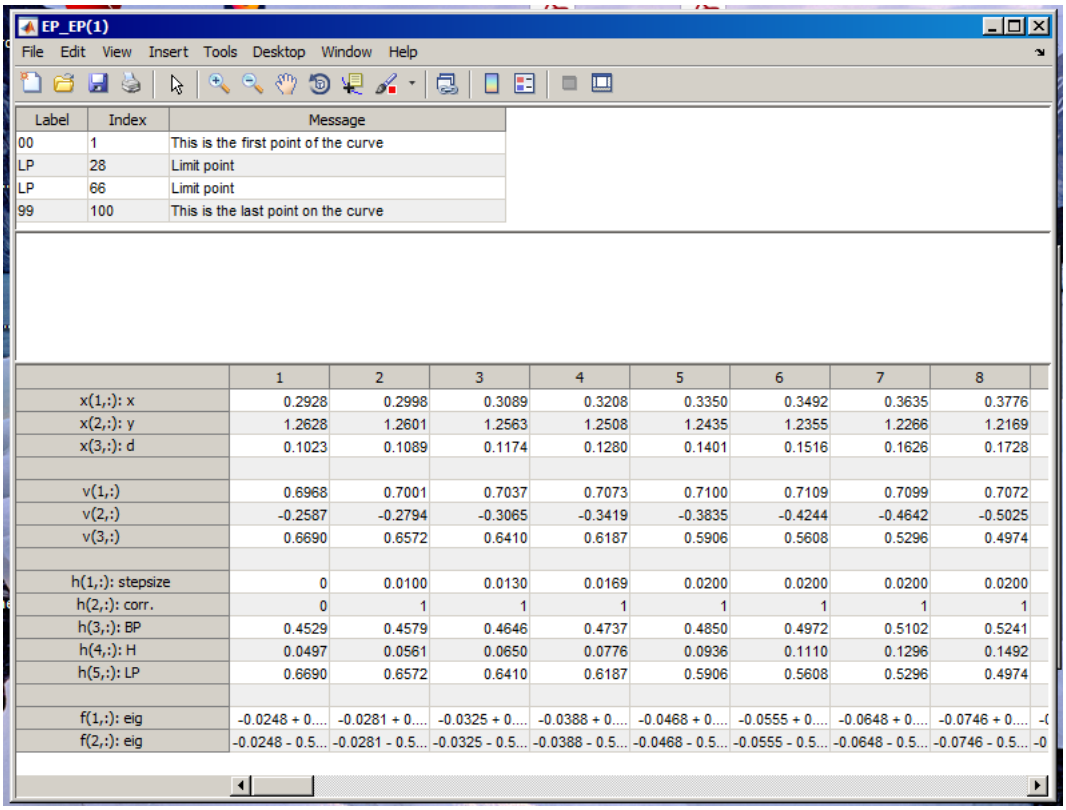
$$d_2 = 0.256805\ldots, \quad d_1 = 0.176927\ldots$$

(see Figures B.11 and B.12). One eigenvalue (close to) zero is also present in each case. The existence of three equilibria for $d \in (d_1, d_2)$ is evident from the figure.

In the MATLAB Command Window, the value of the *fold normal form* coefficient a is shown at each limit point:

```
label = LP, x = ( 0.619532 0.927986 0.256805 )
a=-5.311546e-01
label = LP, x = ( 0.911266 0.268200 0.176927 )
a=5.681504e-01
```

Finally, clear the **2Dplot** window once more and recompute the equilibrium curve forward with **Options** | **Suspend Computation** | **At Each Point** mode selected as shown in Figure B.13. Resume computations after each computed point and monitor the



Label	Index	Message
00	1	This is the first point of the curve
LP	28	Limit point
LP	66	Limit point
99	100	This is the last point on the curve

	1	2	3	4	5	6	7	8
x(1,:): x	0.2928	0.2998	0.3089	0.3208	0.3350	0.3492	0.3635	0.3776
x(2,:): y	1.2628	1.2601	1.2563	1.2508	1.2435	1.2355	1.2266	1.2169
x(3,:): d	0.1023	0.1089	0.1174	0.1280	0.1401	0.1516	0.1626	0.1728
v(1,:)	0.6968	0.7001	0.7037	0.7073	0.7100	0.7109	0.7099	0.7072
v(2,:)	-0.2587	-0.2794	-0.3065	-0.3419	-0.3835	-0.4244	-0.4642	-0.5025
v(3,:)	0.6690	0.6572	0.6410	0.6187	0.5906	0.5608	0.5296	0.4974
h(1,:): stepsize	0	0.0100	0.0130	0.0169	0.0200	0.0200	0.0200	0.0200
h(2,:): corr.	0	1	1	1	1	1	1	1
h(3,:): BP	0.4529	0.4579	0.4646	0.4737	0.4850	0.4972	0.5102	0.5241
h(4,:): H	0.0497	0.0561	0.0650	0.0776	0.0936	0.1110	0.1296	0.1492
h(5,:): LP	0.6690	0.6572	0.6410	0.6187	0.5906	0.5608	0.5296	0.4974
f(1,:): eig	-0.0248 + 0....	-0.0281 + 0....	-0.0325 + 0....	-0.0388 + 0....	-0.0468 + 0....	-0.0555 + 0....	-0.0648 + 0....	-0.0746 + 0....
f(2,:): eig	-0.0248 - 0.5...	-0.0281 - 0.5...	-0.0325 - 0.5...	-0.0388 - 0.5...	-0.0468 - 0.5...	-0.0555 - 0.5...	-0.0648 - 0.5...	-0.0746 - 0.5...

Figure B.10: Spreadsheet view of the **Curve Data**.

eigenvalues in the **Numeric** window. You should see that, indeed, the upper and lower branches correspond to linearly stable equilibria (with all Re $\lambda_i < 0$), while the middle branch correspond to a linear saddle (with $\lambda_1 < 0 < \lambda_2$).

Since $a \neq 0$ within numerical accuracy, the equilibrium curve is approximately a (scaled) parabola near each of the limit points.

Restore the original pause option (`At Special Points`) via **Options** | **Suspend Computation** | **At Special Points** and close the **Numeric** window.

B.1.4 Phase portrait

To verify our conclusions, compute several orbits to get a phase portrait of the system at $d = 0.2$ as in Figure B.14. Start always at $x = 1.2$ but with different values for y.

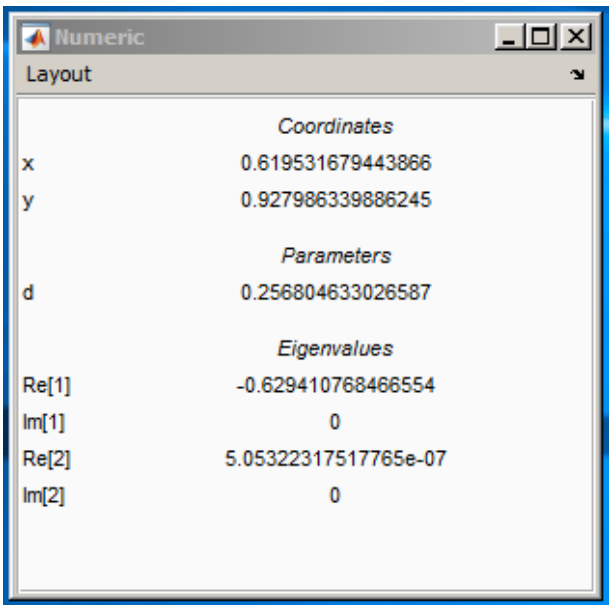


Figure B.11: **Numeric** window at the first limit point.

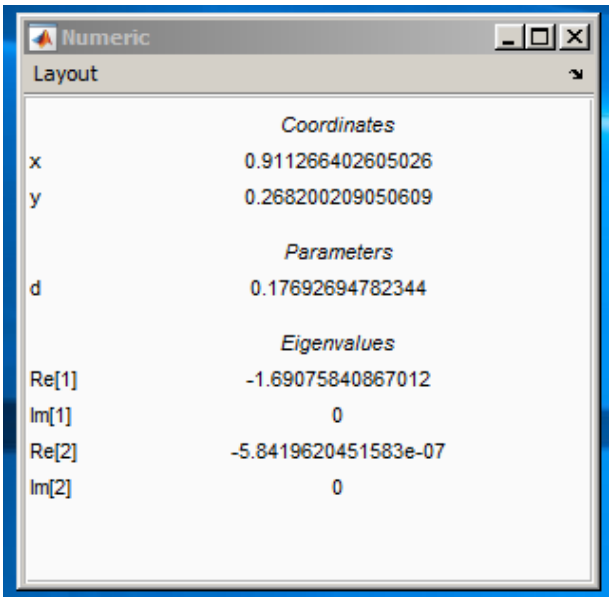


Figure B.12: **Numeric** window at the second limit point.

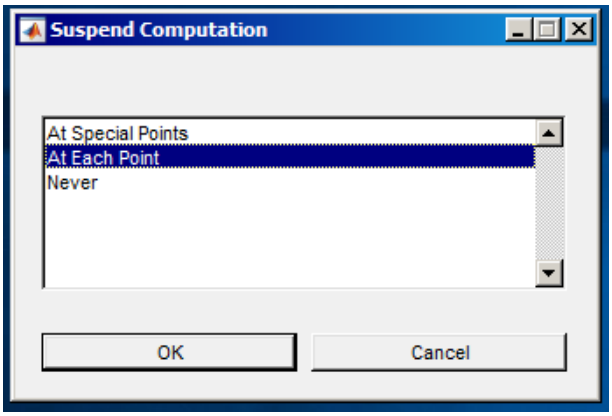


Figure B.13: Pause at each computed point.

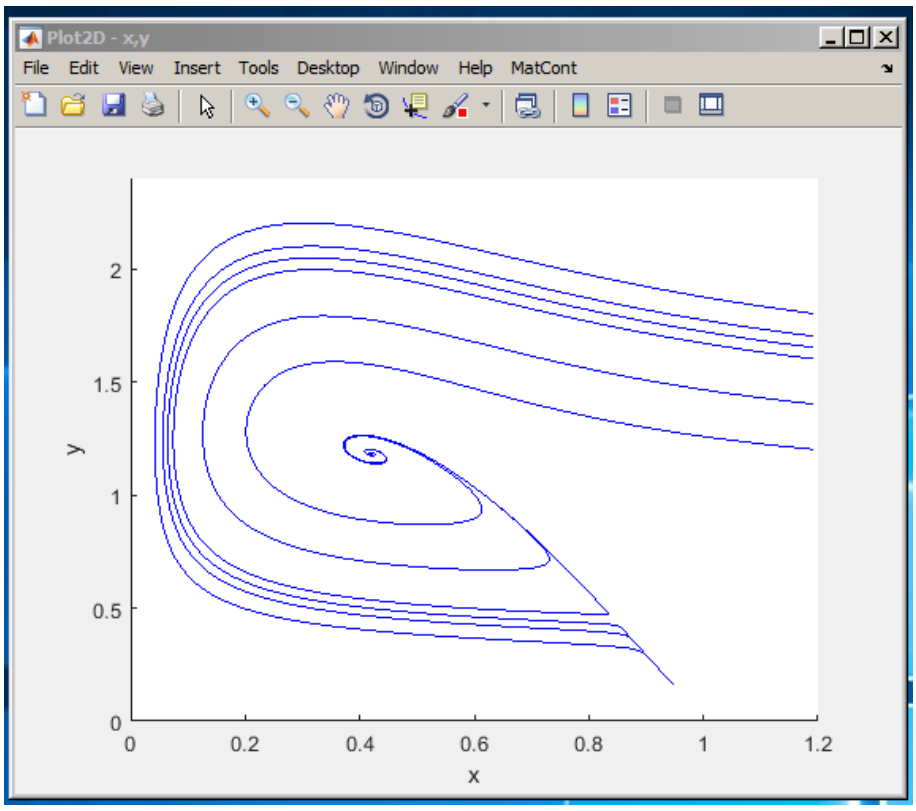


Figure B.14: Two stable equilibria (a focus and a node) and a saddle in the ecological model at $d = 0.2$. The stable manifold of the saddle separates the domain of attraction of the focus from that of the node.

B.2 Limit and branching points in a discretization of Bratu-Gelfand PDE

Consider the following evolution problem for $u = u(x,t)$ with $x \in [0,1], t \geq 0$:

$$u_t = u_{xx} + \alpha e^u, \quad u(0,t) = u(1,t) = 0 \text{ for all } t, \tag{B.2}$$

where α is a real parameter. It is called the *Bratu-Gelfand* problem. In this section we study the behaviour of a stationary solution of (B.2) as a function of α using a finite-difference approximation over an equidistant mesh. One such time-independent solution is obvious: $u \equiv 0$ for $\alpha = 0$.

B.2.1 Discretization

Characterize a solution to (B.2) at time t by its values $u_k(t) = u(x_k, t)$ at the uniformly distributed mesh points

$$x_k = kh, \quad h = \frac{1}{N+1}, \; k = 0, 1, \ldots, N, N+1,$$

for some $N > 1$. Then approximate the spatial derivative in (B.2) by finite differences at the inner points

$$\dot{u}_k(t) = \frac{u_{k-1}(t) - 2u_k(t) + u_{k+1}(t)}{h^2} + \alpha \exp(u_k(t)) = 0, \quad k = 1, 2, \ldots, N, \tag{B.3}$$

and add the boundary conditions in the form

$$\left\{ \begin{array}{rcl} u_0(t) & = & 0, \\ u_{N+1}(t) & = & 0. \end{array} \right. \tag{B.4}$$

After elimination of $u_0(t)$ and $u_{N+1}(t)$ and introduction of a rescaled time τ such that $t = h^2\tau$, the equations (B.3) and (B.4) take the form of the ODE-system

$$\dot{U} = F(U, \lambda), \tag{B.5}$$

where $U = (u_1, u_2, \ldots, u_N)$ and $F : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N$ is given by (B.3) (with $u_0 = u_n = 0$) and

$$\lambda := h^2\alpha = \frac{\alpha}{(N+1)^2} \; .$$

An equilibrium of this system approximates a stationary solution to (B.2) with $O(h^2)$-accuracy.

In this section, we will study a crude spatial discretization of (B.2) with only two internal points (i.e. $N = 2$ and $h = \frac{1}{3}$), so that (B.5) becomes

$$\left\{ \begin{array}{rcl} \dot{u}_1 & = & -2u_1 + u_2 + \lambda e^{u_1}, \\ \dot{u}_2 & = & u_1 - 2u_2 + \lambda e^{u_2}, \end{array} \right. \tag{B.6}$$

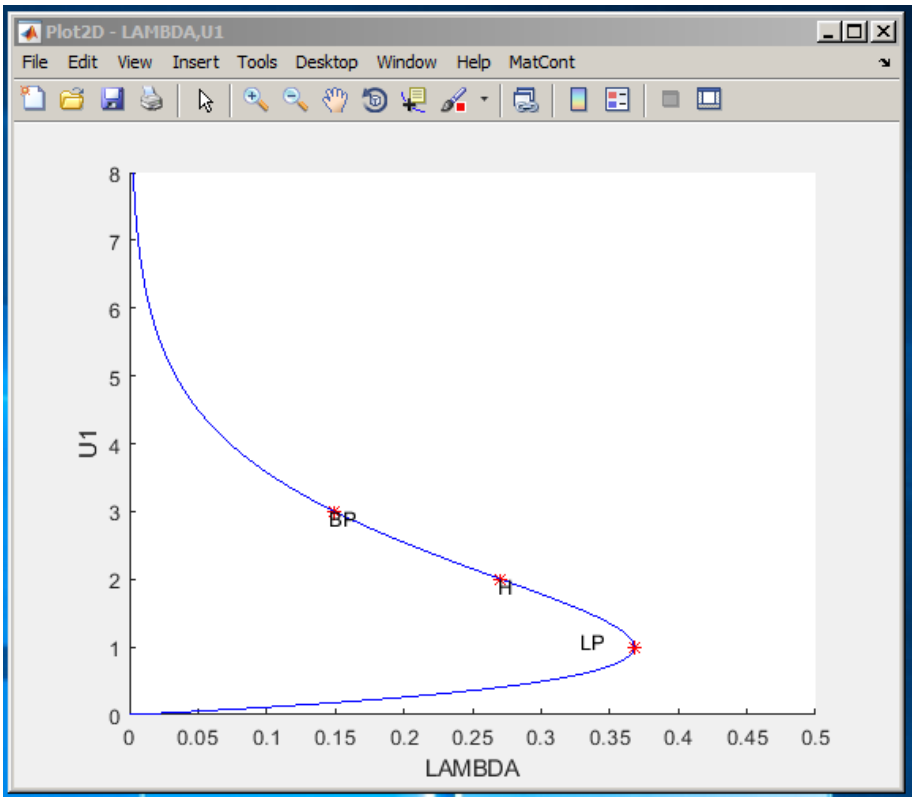


Figure B.15: The curve of symmetric equilibria of (B.6) with limit point `LP`, neutral saddle `H`, and branching point `BP`.

where $\lambda = h^2\alpha = \frac{1}{9}\alpha$. The system (B.6) has $\mathbb{Z}_2$-*symmetry*: It is invariant under the involution $(u_1, u_2) \mapsto (u_2, u_1)$.

We will compute the equilibrium manifold of (B.6) in the (u_1, u_2, λ)-space starting from point $(0, 0, 0)$ corresponding to the trivial stationary solution of (B.2), namely $u(x, t) \equiv 0$.

B.2.2 System specification

Input a new ODE system into MatCont, namely

```
U1'=-2*U1+U2+LAMBDA*exp(U1)
U2'=U1-2*U2+LAMBDA*exp(U2)
```

and choose to generate the derivatives of order 1 and 2 symbolically.

B.2.3 Symmetric equilibrium branch

Select **Type|Initial Point|Equilibrium**. Two windows will appear: **Starter** and **Continuer** windows corresponding to the equilibrium continuation (curve type `EP_EP`). In the **Starter** window, activate the parameter `LAMBDA`. Since (B.6) has an equilibrium $u_1 = u_2 = 0$ for $\lambda = 0$, no further changes to the **Starter** parameters are required.

Open a **2Dplot** window and select the parameter `LAMBDA` and the coordinate `U1` as abscissa and ordinate, respectively, with the visibility limits

```
Abscissa:          0          0.5
Ordinate:          0          8.0
```

To monitor the stability of the equilibrium, open a **Numeric** window and activate the eigenvalues to be visible via the **Numeric|Layout** window.

Click **Compute|Forward**. The first bifurcation point, namely a *limit point* `LP`, will be located at $\lambda = 0.367879....$ In the MATLAB Command Window, you can read the following message:

```
label = LP, x = ( 1.000001 1.000001 0.367879 )
a=3.535537e-01
```

that gives the critical equilibrium coordinates and the critical parameter value, as well as the *quadratic normal form coefficient* $a = 0.35355\ldots$ at `LP`. Thus, the limit point is nondegenerate and the equilibrium manifold near `LP` looks like a parabola.

Further computation detects a *neutral saddle* labeled `H` at $\lambda = 0.270671...$, where the equilibrium has eigenvalues $\mu_{1,2} = \pm 1$, so that their sum is zero. In the MATLAB Command Window, you can read the following message:

```
label = H , x = ( 2.000000 2.000000 0.270671 )
Neutral saddle
```

This is not a bifurcation point for the equilibrium, since it is a hyperbolic saddle.

Resume computations to get the second bifurcation point at $\lambda = 0.14936122...$, namely a *branching point* labelled `BP`, with the following message in the MATLAB Command Window

```
label = BP, x = ( 3.000000 3.000000 0.149361 )
```

Extend the equilibrium curve further until you leave the window. You should get Figure B.15.

To get more information on the `BP` point, single-click at the label and the message

```
*** EP_EP(1): Branch point (BP)
        npoint:  53
```

will appear in the MATLAB command window. It tells us that the BP point is a Branch point that was detected along the EP_EP(1) curve as its 53−th point.

To get more information on the H point, single-click at the label and the message

```
*** EP_EP(1): Neutral Saddle Equilibrium (H)
         npoint:        37
         1st Lyapunov:  Neutral saddle
```

will appear in the MATLAB command window. It tells us the H point is a Neutral Saddle that was detected along the EP_EP(1) curve as its 37−th point. Since it is not a Hopf point, it does not have a 1st Lyapunov coefficient.

To get more information on the LP point, single-click at the label and the message

```
*** EP_EP(1): Limit point (LP)
         npoint:  22
         a:       3.5355366e-01
```

will appear in the MATLAB command window. It tells us the LP point is a Limit point that was detected along the EP_EP(1) curve as its 22−th point. Is normal form coefficient a is nonzero, so it is nondegenerate.

Notice that along the whole computed branch we have $u_1 = u_2$, i.e. it is composed of *symmetric equilibria*. Check by looking at the eigenvalues in the **Numeric** window that only the lower part of the branch is stable, while it is a saddle between LP and BP, and a repeller above it.

Rename the computed curve via **Select**| **Curve**|**Rename** to symmetric.

B.2.4 Symmetry breaking

Let us compute the second branch passing through the branching point BP. Double-click with the computer mouse at the BP point in the **2Dplot** window presented in Figure B.15. This opens a small **Select BP as a new initial point** window, see Figure B.16. Click 'Yes' and the (u_1, u_2, λ)-values corresponding to BP will be loaded and by default the initializer BP_EP to the new equilibrium curve is loaded as well.

In the appearing **Starter** window, untick monitoring all singularities as in Figure B.17.

Leave the **Continuer** window as it is and click **Compute**|**Forward** and **Compute**|**Backward**. You will get Figure B.18. Thus, the branching point BP corresponds, actually, to a *pitchfork bifurcation*: Two more branches of equilibria bifurcate vertically from this point. Moreover, each branch is composed of equilibria which are not symmetric, $u_1 \neq u_2$. However, one branch is mapped into the other by the involution. This phenomenon is called the *symmetry breaking*: We got non-symmetric equilibrium solutions in a symmetric system.

Warning: We have shown numerically that a *discretization* of the Bratu-Gelfand problem (B.2) has multiple stationary solutions and limit and branching points. It does not

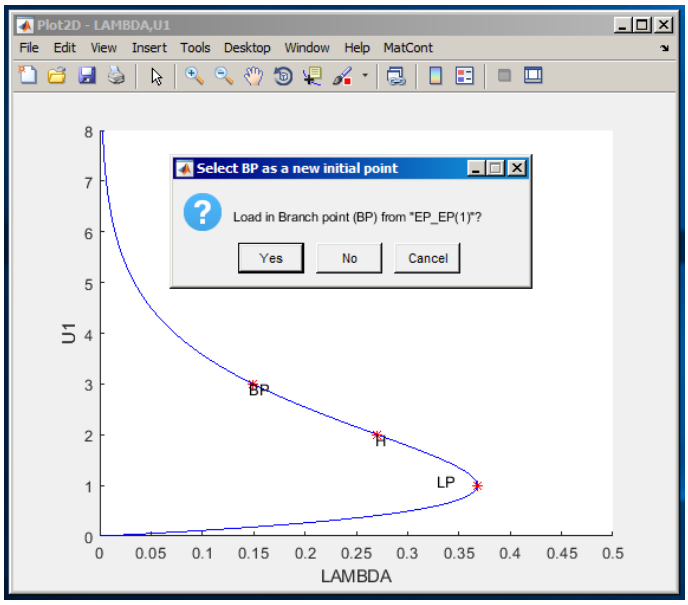


Figure B.16: `BP` selection.

imply immediately that the original PDE problem has the same properties. To verify this, one has to repeat the continuation with smaller h (i.e. bigger N) and eventually make some error estimates. It should be noted that the appearance of the pitchfork is due to the equidistant mesh with a small number of points. If more accurate discretizations of (B.2) are used, the pitchfork and associated branches disappear. The limit point remains and corresponds to a limit point on the branch of the stationary solutions of (B.2).

B.2.5 Stopping at a zero of a userfunction

Sometimes we want to stop computation at a prescribed parameter value. This can be done in MatCont by specifying a *userfunction*. For example, let us compare two non-symmetric equilibria at $\lambda = 0.1$.

Userfunctions can be introduced in two different ways. One of them is by pressing **Select | System | Manage Userfunctions**. The other way is to right-click in the **Current System** field of the MatCont main window and then selecting **Manage Userfunctions**. Use the second method. This opens a **Userfunctions** window. Define a userfunction `Stop` with the associated label `S` by typing `res=LAMBDA-0.1` in the edit field of the **Userfunctions** window (see Figure B.19). Next press **Add** and then **OK** buttons. The **Starter** window will change and show a user function control. Click to activate it. If one wants to monitor the userfunction, the `userfunctions` must also be activated in the layout of the **Numeric** window.

Recomputing the non-symmetric branches with **Compute|Forward** and **Compute|Backward** will stop at two points labeled `S`, where the user-defined function

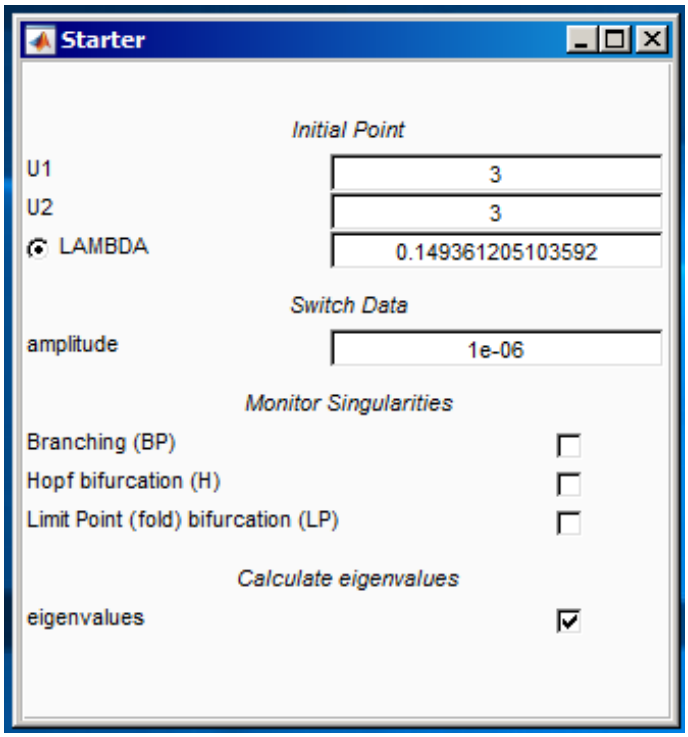


Figure B.17: **Starter** window to switch branches at `BP`.

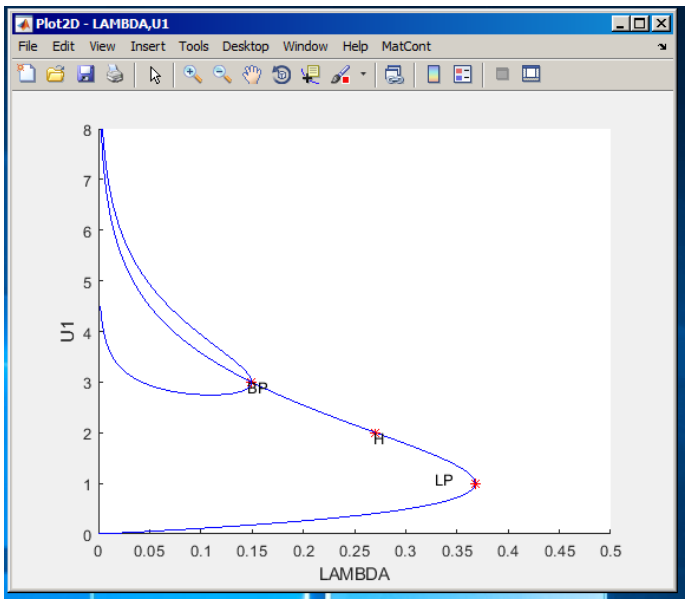


Figure B.18: Two equilibrium branches passing through a branch point `BP`.

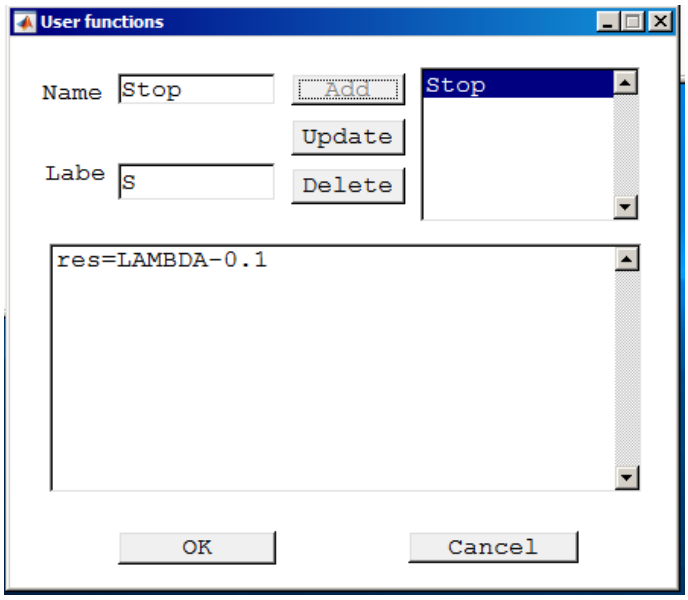


Figure B.19: Userfunction input.

`stop` vanishes, see Figure B.20. You can see the corresponding equilibrium coordinates and eigenvalues in the **Numeric** window.

B.3 Additional Problems

A. Compute the equilibrium manifold of the scalar ODE

$$\dot{y} = x^2 + y^2 - 1, \quad y \in \mathbb{R}, \tag{B.7}$$

where $x \in \mathbb{R}$ is a parameter.

B. Consider the following chemical model

$$\begin{cases} \dot{x} &= 2q_1z^2 - 2q_5x^2 - q_3xy, \\ \dot{y} &= q_2z - q_6y - q_3xy, \\ \dot{s} &= q_4z - Kq_4s, \end{cases}$$

where $z = 1 - x - y - s$ and

$$q_1 = 2.5, \quad q_2 = 1.55, \quad q_3 = 10, \quad q_4 = 0.0675, \quad q_5 = 1.0, \quad q_6 = 0.1,$$

and

$$K = 2.0.$$

Given an equilibrium $(x, y, s) = (0.0032\ldots, 0.8838\ldots, 0.0376\ldots)$, find two more positive equilibria of the system.

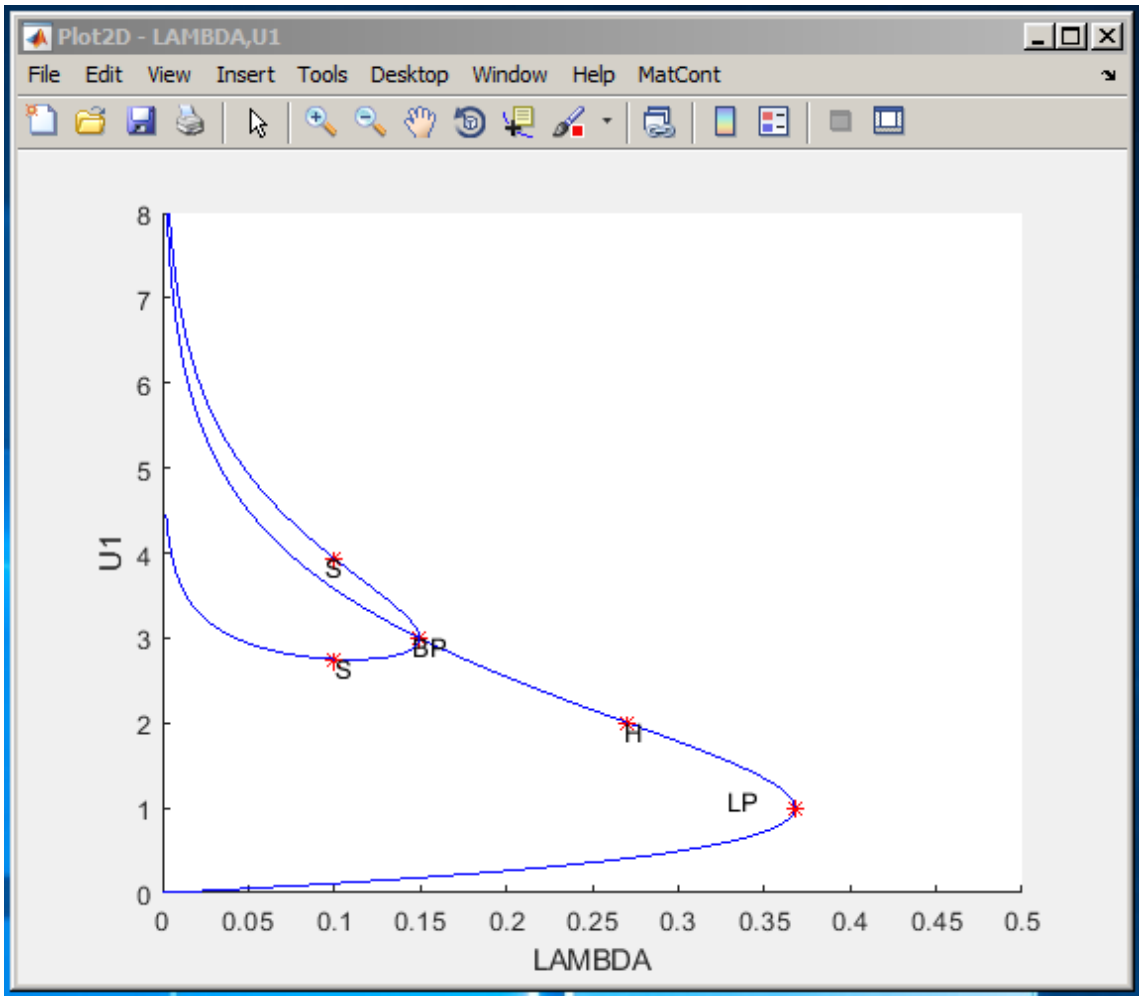


Figure B.20: Zeros of a userfunction are labeled 'S'.

Hint: Compute a curve in the (x, y, s)-space defined by the first two equations:

$$\begin{cases} 2q_1z^2 - 2q_5x^2 - q_3xy &= 0, \\ q_2z - q_6y - q_3xy &= 0, \end{cases}$$

and detect zeros of the user-defined function $F = q_4z - Kq_4s$ along this curve[1].

C. Find a real eigenvalue and the corresponding eigenvector of the matrix

$$B = \begin{pmatrix} -5 & 2 & -3 \\ -5 & 0 & -4 \\ 8 & -2 & 6 \end{pmatrix}$$

[1]As described in Section 2.5, click **Select** | **Userfunction** in the MatCont window and type the expression for F after `res=` in the edit field of the appearing **Userfunctions** window. Fill in the `Name` and `Label` fields, and press **Add** and **OK**.

by continuation of the eigenvector $v_1 = (0, 0, 1)^T$ of the matrix

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -2 \end{pmatrix},$$

corresponding to its eigenvalues $\lambda_1 = -2$.

1. Consider a one-parameter family of matrices

$$C(a) = aB + (1-a)A, \quad C(0) = A, \; C(1) = B,$$

and the following continuation problem in the (v, λ, a)-space:

$$\begin{cases} C(a)v - \lambda v &= 0, \\ v^T v - 1 &= 0. \end{cases}$$

Explain, how it can be used to solve the formulated problem.

2. Setup an auxiliary 4D-system of ODEs

$$\begin{cases} \dot{v} &= C(a)v - \lambda v, \\ \dot{\lambda} &= v^T v - 1 \end{cases}$$

and continue its equilibrium manifold in the (v, λ, a)-space. Use $(v_1, \lambda_1, 0)$ as the initial point. Stop when $a = 1$ using a user-defined function.

3. Continue other two eigenvectors of A and produce a plot tracing all three real eigenvalues in the (a, λ)-plane. What happens in the limit points of these curves?

4. Can you setup a continuation problem for (real and imaginary) parts of complex eigenvalues and eigenvectors of $C(a)$?

D. Study equilibria of the following *complex* ODE:

$$\dot{z} = \alpha + z^2, \quad z \in \mathbb{C}^1, \tag{B.8}$$

where the parameter α and the time are real. *Hint*: Write $z = x + iy$, where (x, y) are real. Then (B.8) takes the form

$$\begin{cases} \dot{x} &= \alpha + x^2 - y^2, \\ \dot{y} &= 2xy. \end{cases} \tag{B.9}$$

At $\alpha = -1$ this system has equilibrium $(x_0, y_0) = (1, 0)$, which can be continued.

Plot also the phase portraits in the (x, y)-plane of (B.9) for $\alpha = -0.25$, $\alpha = 0$, and $\alpha = 0.25$

E. Study discretizations of the Bratu-Gelfand problem in more detail:

1. For $N = 2$, find *analytically* the parameter value $\lambda_{\mathtt{BP}}$ corresponding to the branching point in system (B.6) and compare it with the numerical result in Section 2.
2. Analyse numerically the effect of increasing the number of equidistant mesh points to $N = 3$, $N = 4$, and and $N = 10$ on the α-values corresponding to the limit and branching points in the finite-difference approximation (B.3)-(B.4) of the Bratu-Gelfand problem (B.2).

   Hint: Do not forget that $\alpha = (N + 1)^2\lambda$.

Appendix C

TUTORIAL III: Using the new MATCONT GUI for one-parameter bifurcation analysis of limit cycles

This session was tested on MATCONT7p1 with MATLAB2017b. It is devoted to the numerical initialization and continuation of limit cycles in systems of autonomous ODEs depending on one parameter

$$\dot{x} = f(x, \alpha), \quad x \in \mathbb{R}^n, \alpha \in \mathbb{R},$$

and detection of their bifurcations. We will also switch to the continuation of the limit cycle from the Hopf bifurcation and to the continuation of the doubled cycle at the period-doubling bifurcation.

C.1 Initialization from a converging orbit

We consider again the *Rössler system* that was introduced in TUTORIAL I:

$$\begin{cases} \dot{x} &= -y - z \\ \dot{y} &= x + Ay \\ \dot{z} &= Bx - Cz + xz, \end{cases}$$

where (x, y, z) are the phase variables, and (A, B, C) are the parameters.

Introduce this system in MATCONT, generating the derivatives of order 1, 2 and 3 symbolically. We start with computing by time integration an orbit, using the **Starter** and **Integrator** windows presented in Figure C.1.

The computed orbit is presented in the 3D-plot in Figure C.2 in (x, y, z) - space within the region

$$-9 \leq x \leq 9, \ -9 \leq y \leq 9, \ -2 \leq z \leq 11.$$

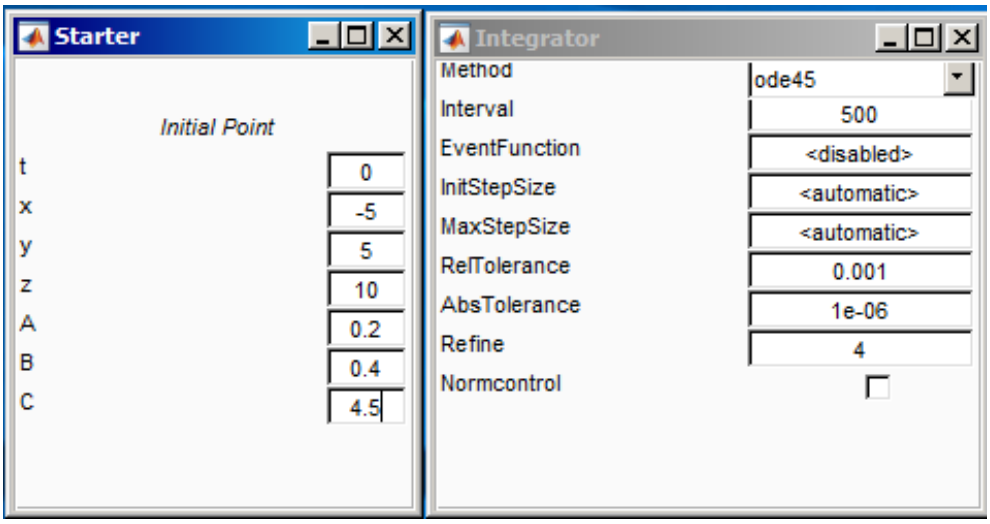


Figure C.1: The **Starter** and **Integrator** windows for the initial time integration.

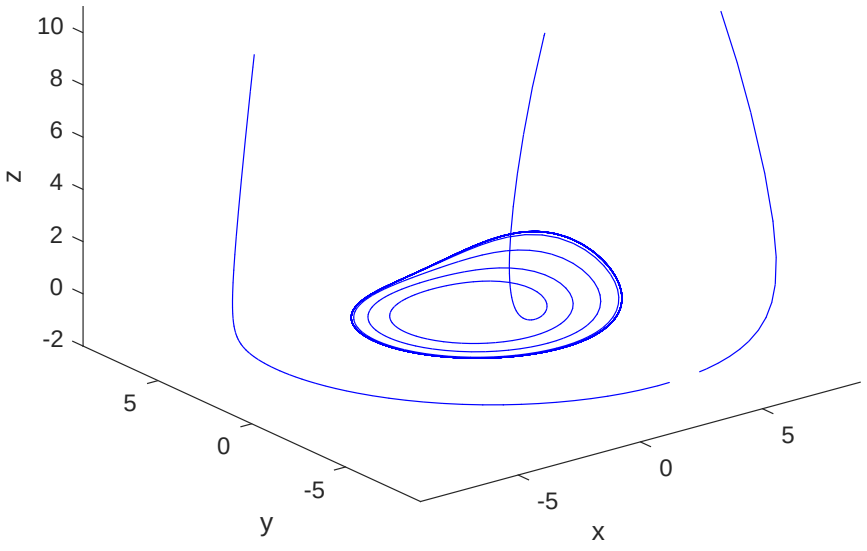


Figure C.2: 3D-plot of the computed orbit.

It is visually clear that the orbit has converged to a (presumably stable) limit cycle. We select the last point of the orbit using **Select** | **Initial Point** | **Last Point** | **Select Point** and clear the 3D-plot.

It is readily seen that time integration from the selected point over a time interval 5 does not close the orbit, see the **Starter** and **Integrator** windows in Figure C.3, and the orbit displayed in Figure C.4.

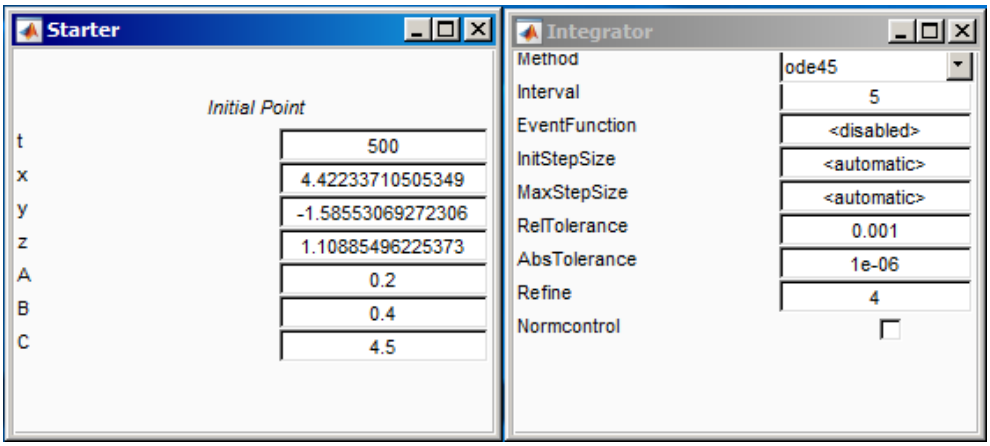


Figure C.3: **Starter** and **Integrator** windows for time integration from a selected point on the orbit.

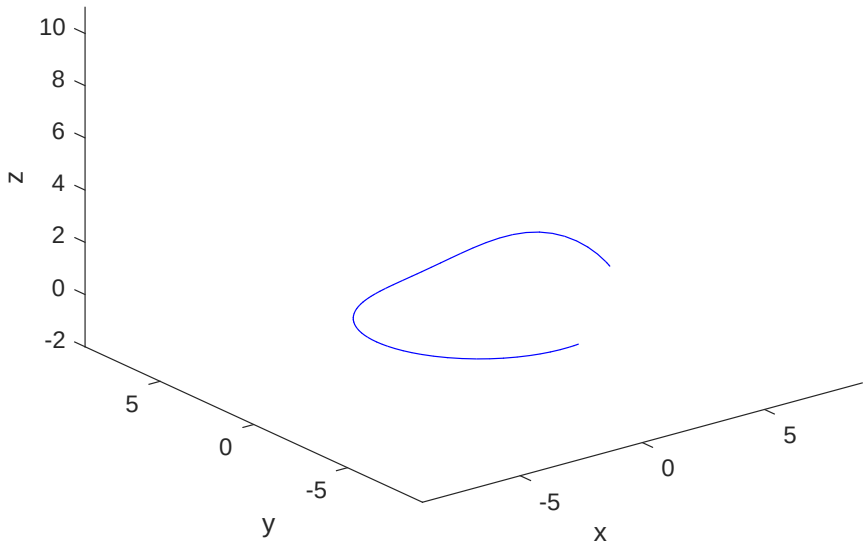


Figure C.4: Computed orbit over an interval 5.

Now perform a time integration over an interval 8 and check that it closes the limit cycle. This implies that 8 is larger than the period of the limit cycle but smaller than

twice that period, which is the condition for starting the continuation of limit cycles from an orbit that is computed along the limit cycle. That is why time integrations over an interval 5 or 500 are of no use.

Now press **Output | View Result** and double-click at **Select Cycle.** (we note that a single-click at **Select Cycle** followed by clicking **Select Point** also works). This opens a small **Choose Tolerance ...** input window with two input fields. The top one allows to change the tolerance for detecting a limit cycle along the orbit. The bottom one allows to change the number of mesh intervals that will be used to discretize the limit cycle. Make sure that it is set to 40; the window should now look like Figure C.5.

Now press **OK**. The main MatCont window is then prepared for the continuation of a limit cycle, see Figure C.6(left). In the corresponding **Starter** window activate the parameter `A` and the `Period`. Also, click all singularity monitoring buttons to 'yes', cf. Figure C.6(right). We note that the initial value of the Period is estimated by the initializer and equal to 6.27..., a value between 5 and 8.

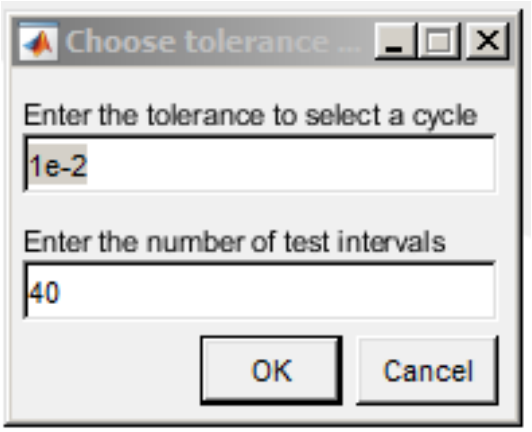


Figure C.5: Input window opened by pressing the **Select Cycle** button.

In the corresponding **Continuer** window only default values are to be used. Open a **Numeric** window and click on the 'multipliers' field in its **Layout** window to activate the multipliers of the periodic orbits to be displayed in the **Numeric** window during continuation.

Press **Compute|Forward**. The continuation of limit cycles now starts. Observe that there is period-doubling bifurcation for $A = 0.27191773$. The 3D-plot window and the **Numeric** window are displayed in Figure C.7. Observe that at the PD-point there is a multiplier equal to -1.

In the MATLAB command line the output

```
Period Doubling (period = 6.148967e+00, parameter = 2.719177e-01)
Normal form coefficient = -4.283700e-03
```

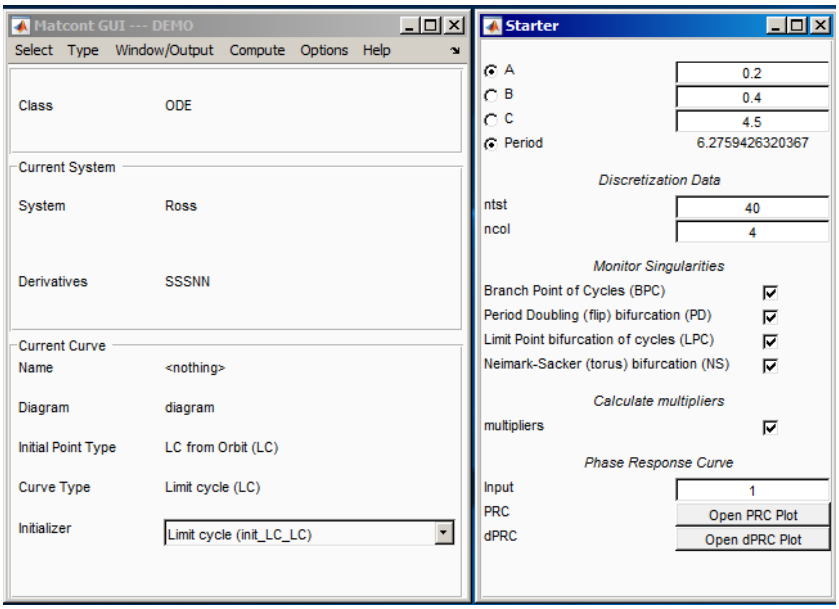


Figure C.6: The main MatCont window when prepared to start the continuation of limit cycles.

is generated, including the normal form coefficient of the PD bifurcation.

Stop the computation and close the session.

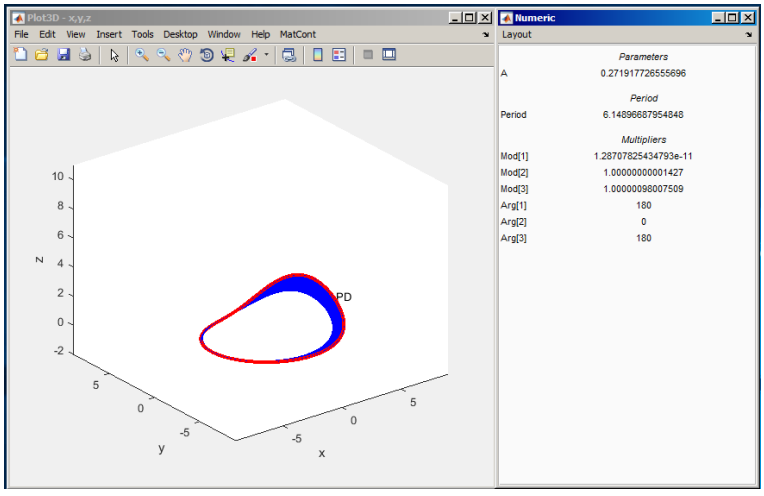


Figure C.7: The **3D-plot** and the **Numeric** windows at detection of the period-doubling bifurcation.

C.2 Fold and Neimark-Sacker bifurcations of cycles in a chemical model

Consider the following chemical model by *Steinmetz and Larter*:

$$\begin{cases} \dot{A} &= -k_1ABX - k_3ABY + k_7 - k_{-7}A, \\ \dot{B} &= -k_1ABX - k_3ABY + k_8, \\ \dot{X} &= k_1ABX - 2k_2X^2 + 2k_3ABY - k_4X + k_6, \\ \dot{Y} &= -k_3ABY + 2k_2X^2 - k_5Y. \end{cases} \tag{C.1}$$

We will fix all parameters but k_7 and study periodic solutions (limit cycles) of (C.1) when this parameter varies.

C.2.1 System specification

Specify a new ODE system – say `StLar` – in MatCont

```
A'=-k1*A*B*X-k3*A*B*Y+k7-km7*A
B'=-k1*A*B*X-k3*A*B*Y+k8
X'=k1*A*B*X-2*k2*X^2+2*k3*A*B*Y-k4*X+k6
Y'=-k3*A*B*Y+2*k2*X^2-k5*Y
```

where (`A,B,X,Y`) are the coordinates and (`k1,k2,k3,k4,k5,k6,k7,km7,k8`) are the parameters. Use (default) `t` for time and generate symbolical derivatives of order 1, 2 and 3.

C.2.2 Fold bifurcation of limit cycles

Continuation of an equilibrium

We will continue an equilibrium of (C.1) and detect its Hopf bifurcation, from which the continuation of a limit cycle can be started.

Input **Type**|**Initial point**|**Equilibrium** and **Type**|**Curve**|**Equilibrium** in the main MatCont window.

Input the following numerical data in the appeared **Starter** window:

```
A               31.78997
B                1.45468
X                0.01524586
Y                0.1776113
k1               0.1631021
k2            1250
k3               0.046875
k4              20
```

```
k5              1.104
k6              0.001
k7              4.235322
km7             0.1175
k8              0.5
```

These values correspond to an unstable equilibrium (A, B, X, Y) in the system. Activate the parameter `k7`.

Open **Window/Output** | **Numeric** and change its appearance via the **Numeric** | **Layout** command. Namely, select `Eigenvalues` to be shown in the window.

Use **Window/Output** | **Graphic** | **2D plot** to open the corresponding window and select the coordinates (`A,B`) as abscissa and ordinate, respectively, with the visibility limits:

```
Abscissa:          32          38
Ordinate:           0           4
```

respectively.

Start **Compute**|**Forward**. The equilibrium curve will be continued and you get a *Hopf bifurcation point* labeled by `H`. The message in the MATLAB Command Window

```
label = H , x = ( 34.808899 1.328517 0.015246 0.177611 4.590046 )
First Lyapunov coefficient = 1.527549e-02
```

at $k_7 = 4.590046\ldots$ indicates a *subcritical* Hopf bifurcation. Indeed, there are two eigenvalues of the equilibrium with Re $\lambda_{1,2} \approx 0$ at this parameter value visible in the **Numeric** window. The critical frequency Im $\lambda_1 \neq 0$, while the first Lyapunov coefficient is positive. Thus, there should exist an unstable limit cycle, bifurcating from the equilibrium. Resume computations and terminate them when the curve leaves the graphic window.

For later use, rename the computed curve via **Select** | **Curve** | **Rename** into `Equilibrium(+)`.

Cycle continuation

Click **Select** | **Initial point** | **Hopf** | **Select Point** to select the `H: Hopf` point in the `Equilibrium(+)` curve as initial point. The **Starter** and **Continuer** windows for the continuation of the limit cycle from the Hopf point will appear.

Tick boxes to monitor all `Monitor Singularities` fields of the **Starter** window. Increase the `MaxStepsize` to 1.0 in the **Continuer** window to allow larger steps along the curve and set `MaxNumPoints` to 50. The **Starter** and **Continuer** windows should look like in Figure C.8.

Change the layout of the **Numeric** window via **Numeric**|**Layout** by selecting all `Multipliers` to be shown. The absolute values (*modulae*) and arguments in *angular grads* will be displayed.

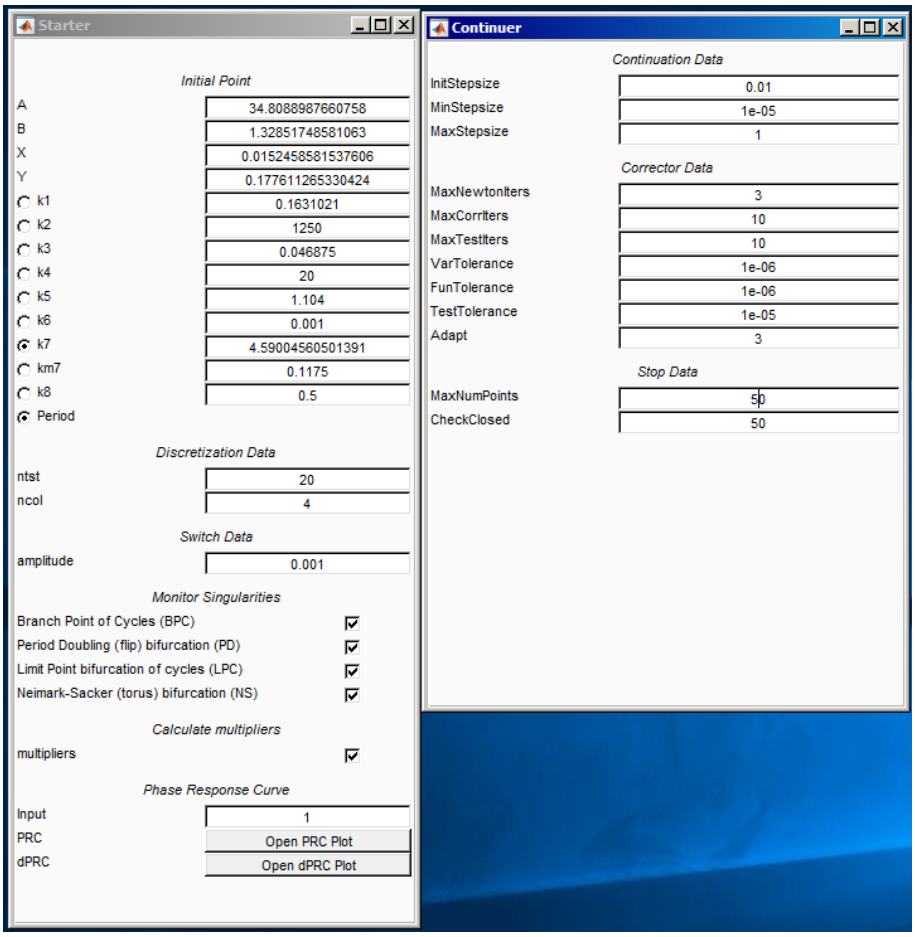


Figure C.8: The **Starter** and **Continuer** windows for the cycle continuation.

Compute|Forward will produce a family of cycles with a *cycle limit point* labeled by `LPC` at $k_7 = 4.74838\ldots$. The **Numeric** window corresponding to `LPC` is shown in Figure C.9.

In the MATLAB Command Window, the following message appears:

```
Limit point cycle (period = 1.036108e+01, parameter = 4.748384e+00)
Normal form coefficient = -2.714838e-01
```

The critical cycle has (approximately) a double multiplier $\mu = 1$ and the normal form coefficient is nonzero. Thus, the limit cycle manifold has a fold here. Resume the computations. The continuation algorithm will automatically follow the second (stable) cycle branch after the `LPC` point. Verify this by looking at the nontrivial multipliers in the **Numeric** window ($|\mu| < 1$ for all such multipliers). The computations produce Figure C.10 (after **MatCont|Redraw diagram** in the **2Dplot** window.)

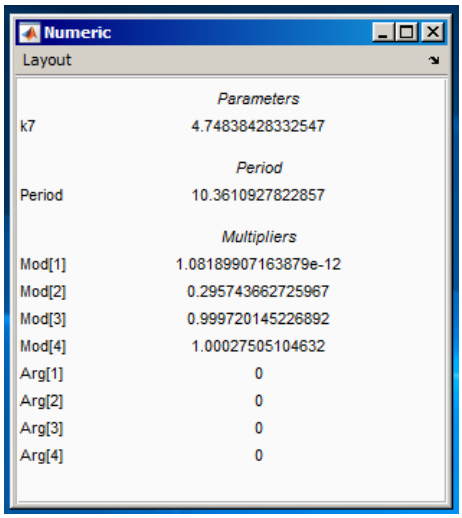


Figure C.9: The **Numeric** window at the LPC-point.

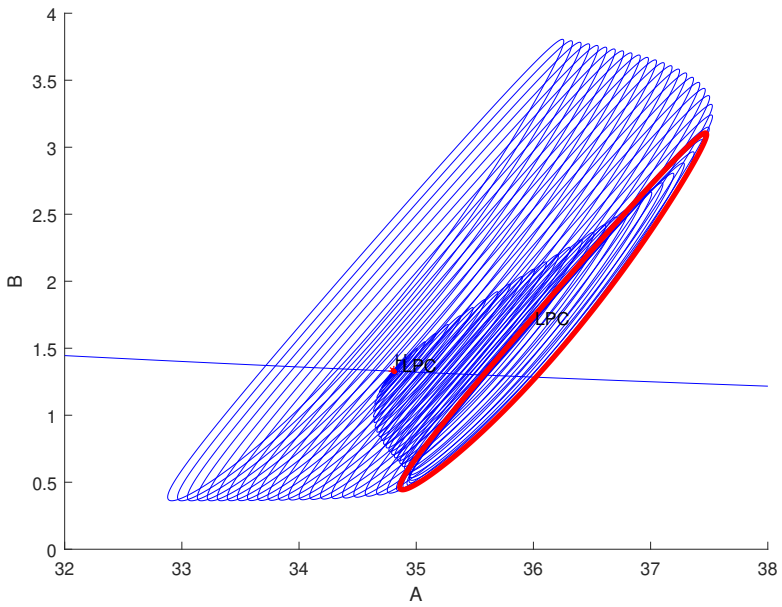


Figure C.10: The family of limit cycles bifurcating from the Hopf point H: LPC is a fold bifurcation of the cycle.

To visualize `LPC` in another way, open another **2Dplot** window and select parameter `k7` and `Period` of the cycle as abscissa and ordinate, respectively. Use the visibility limits:

```
Abscissa:          4.5          4.8
Ordinate:          7            12
```

You will get a curve with a limit point, clearly indicating the presence of two limit cycles with different periods for $k_7 < 4.74838$ near `LPC` (see Figure C.11). Close both **2Dplot**

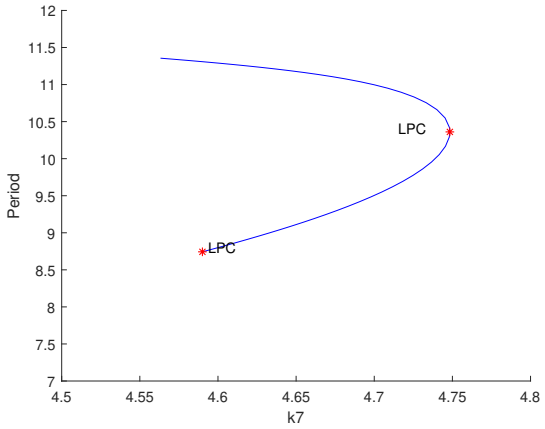


Figure C.11: Period of the cycle versus k_7.

windows.

C.2.3 Neimark-Sacker bifurcation

Backward continuation of the equilibrium

Load the first computed equilibrium curve `Equilibrium(+)` in the **Data Browser** window appearing after the **Select**|**Curve** command. Select **Type**|**Curve**|**Equilibrium** for continuation.

Open a new **2Dplot** window but select now `B` and `X` as abscissa and ordinate, respectively, with the visibility limits:

```
Abscissa:          25           27
Ordinate:          0             0.03
```

Compute|**Backward** and **Extend** once until you get another *Hopf bifurcation* at $k_7 = 0.712475\ldots$ with the message

```
label = H , x = ( 1.808301 25.573303 0.015246 0.177611 0.712475 )
First Lyapunov coefficient = -2.371880e-02
```

in the MATLAB Command Window. The first Lyapunov coefficient is negative now. This means that a *stable* limit cycle bifurcates from the equilibrium, when it loses stability. Resume computations and terminate them when the equilibrium curve leaves the graphic window. Rename the computed curve into `Equilibrium(-)`.

Cycle continuation

Select the Hopf point in the curve `Equilibrium(-)` as initial. MatCont will prepare to continue a limit cycle curve from the Hopf point (curve type `H_LC`). Select `yes` in all `Monitor Singularities` fields of the **Starter** window. Activate k_7 and the Period in the **Starter** window. Set

```
MaxStepsize              1
MaxNumPoints            25
```

in the **Continuer** window. Activate the **Multipliers** to be shown in the **Numeric** window.

Click **Compute**|**Forward** to start the continuation of the limit cycle. At $k_7 = 0.716434\ldots$ the message `Neimark-Sacker` indicates a *torus* bifurcation. In the MATLAB Command Window, the following message appears:

```
Neimark-Sacker (period = 1.091213e+01, parameter = 7.164336e-01)
Normal form coefficient = -4.912065e-08.
```

Indeed, there are two complex multipliers with (approximately) $|\mu| = 1$ and one trivial at $\mu = 1$. This can be seen in the **Numeric** window that stays open, see Figure C.12. The normal form coefficient is small but nonzero, indicating that a stable two-dimensional *invariant torus* bifurcates from the limit cycle.

Resume computations further to see that after the `NS`-point the cycle becomes unstable (with two multipliers satisfying $|\mu| > 1$. You should get a family of limit cycles bifurcating from the Hopf point as in Figure C.13.

Close the **Numeric** window.

Dynamics on a stable torus

Click **Select**|**Initial point...** and select the `NS: Neimark-Sacker` point in the computed limit cycle curve. The critical parameter values will be read in from the archive. Immediately after that select **Type**|**Initial point**|**Point** and **Type**|**Curve**|**Orbit**. A point on the critical cycle will thus be selected as the initial for orbit integration.

Increase slightly the value of the parameter k_7 and perturb the initial point by altering the value of B, namely set:

```
k7              0.7165
B               26.2
```

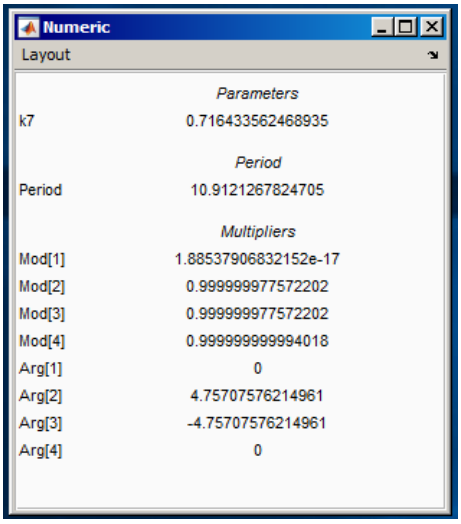


Figure C.12: The **Numeric** window at the NS-point.

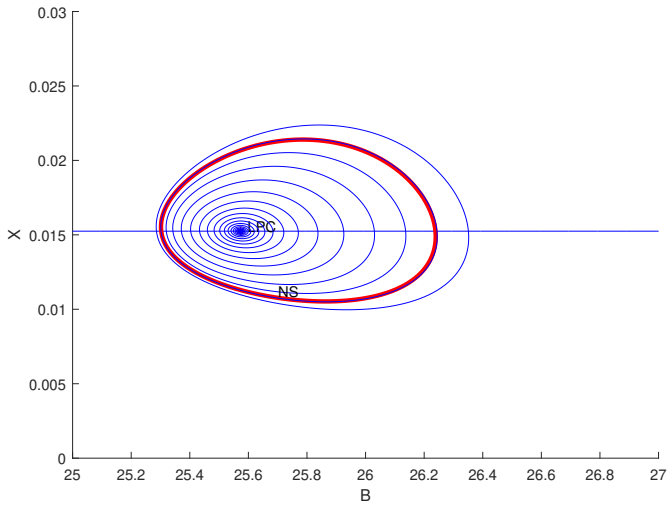


Figure C.13: Family of limit cycles bifurcating from the second Hopf point.

Change **Method** of integration to `ode23s` and alter

```
Interval             1000
Rel.Toletance           1e-6
Abs.Tolerance           1e-9
```

in the **Integrator** window.

To speed up visualization, click **Options**|**Output Interval** and set `Plot after` 1000 points, see Figure C.14

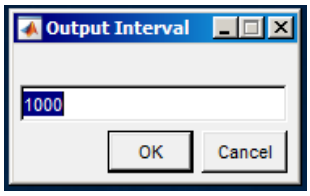


Figure C.14: **Output Interval** window.

Clear the graphic window and **Compute**|**Forward** followed by **Compute**|**Extend**. The integration will take some time. After a transient, the orbit will exhibit modulated oscillations with two frequencies near the limit cycle. This is a motion on a stable two-dimensional torus born via the Neimark-Sacker bifurcation, see Figure C.15. Open another

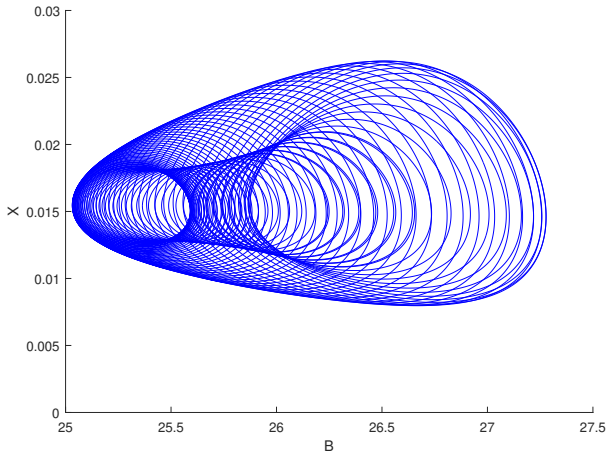


Figure C.15: Dynamics on a stable 2-torus.

2Dplot window with time `t` and coordinate `B` as abscissa and ordinate, with the visibility limits:

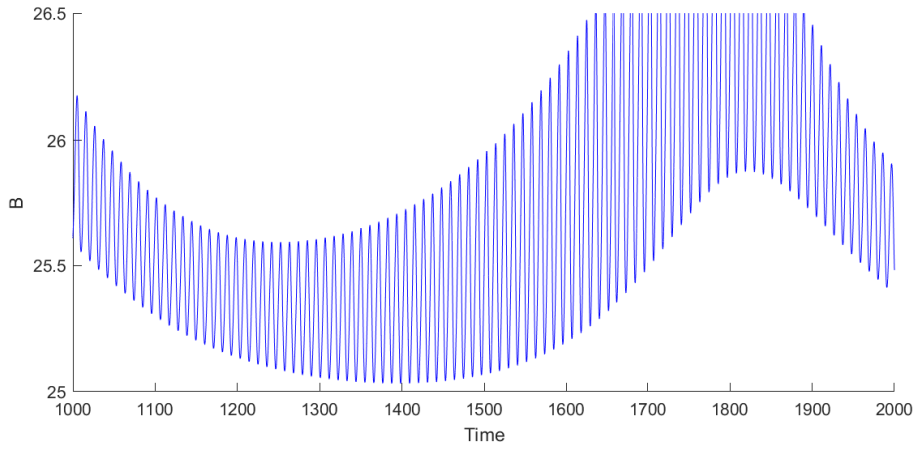


Figure C.16: Modulated chemical oscillations.

```
Abscissa:          1000          2000
Ordinate:            25.0          26.5
```

You will see the high-frequency oscillations with low-frequency modulation as in Figure C.16.

Restore the original `Plot after` 1 point setting using **Options**|**Output** menu of the main MatCont window.

C.3 Period-doubling bifurcation in an adaptive control model

Consider the following *adaptive control system of Lur'e type*

$$\begin{cases} \dot{x} &= y, \\ \dot{y} &= z, \\ \dot{z} &= -\alpha z - \beta y - x + x^2. \end{cases} \tag{C.2}$$

We will fix $\alpha = 1$ and continue limit cycles of (C.2) using β as a bifurcation parameter.

C.3.1 System specification

As usual, define a new system in MatCont with the coordinates `x,y,z`, parameters `alpha,beta`, and select derivatives of order 1, 2, and 3 to be generated symbolically.

C.3.2 Equilibrium continuation

Select **Type**|**Initial point**|**Equilibrium** and **Type**|**Curve**|**Equilibrium**. In the **Starter** window, set `alpha` equal to 1 and activate `beta`.

Open a **3Dplot** window and select `beta,x` and `y` as variables along the coordinate axes with the visibility limits:

```
Abscissa:          0.4      1.2
Ordinate:         -0.5      1.2
Applicate:        -0.6      0.7
```

respectively.

Compute|**Forward** results in an equilibrium curve with a Hopf bifurcation (labeled `H`) detected with the following message in the MATLAB Command Window:

```
label = H , x = ( 0.000000 0.000000 0.000000 1.000000 )
First Lyapunov coefficient = -3.000000e-01
```

The Hopf bifurcation occurs at $\beta = 1$ and is supercritical, thus generating a stable limit cycle. Resume the continuation and eventually stop it, when the curve leaves the plotting region.

C.3.3 Cycle continuation starting from the H-point

Click **Output** | **View Result** and double-click at the `H: Hopf` point. The **Starter** and **Continuer** windows appear, corresponding to the limit cycle curve `H_LC`. Make sure that `beta` and `Period` are activated.

In the **Starter** window, increase the number of mesh points `ntst` to 40 and select `yes` in all `Monitor Singularities` fields. In the **Continuer** window, set `MaxStepsize` to 0.7 and `MaxNumPoints` to 80.

With **Window/Output** | **Numeric**, open a **Numeric** window and make the cycle multipliers visible by activating them in the **Numeric** | **Layout** window.

Compute|**Forward** will produce a branch of limit cycles of increasing amplitude with two *period-doubling bifurcations* (labeled `PD`). The **Numeric** windows at both `PD` bifurcations are shown in Figures C.17 and C.18.

Resume the computations at both `PD` points. The MATLAB Command Window will contain the messages

```
Period Doubling (period = 8.386068e+00, parameter = 6.273245e-01)
Normal form coefficient = -3.405330e-03
Period Doubling (period = 9.864971e+00, parameter = 5.417461e-01)
Normal form coefficient = -7.655596e-04
```

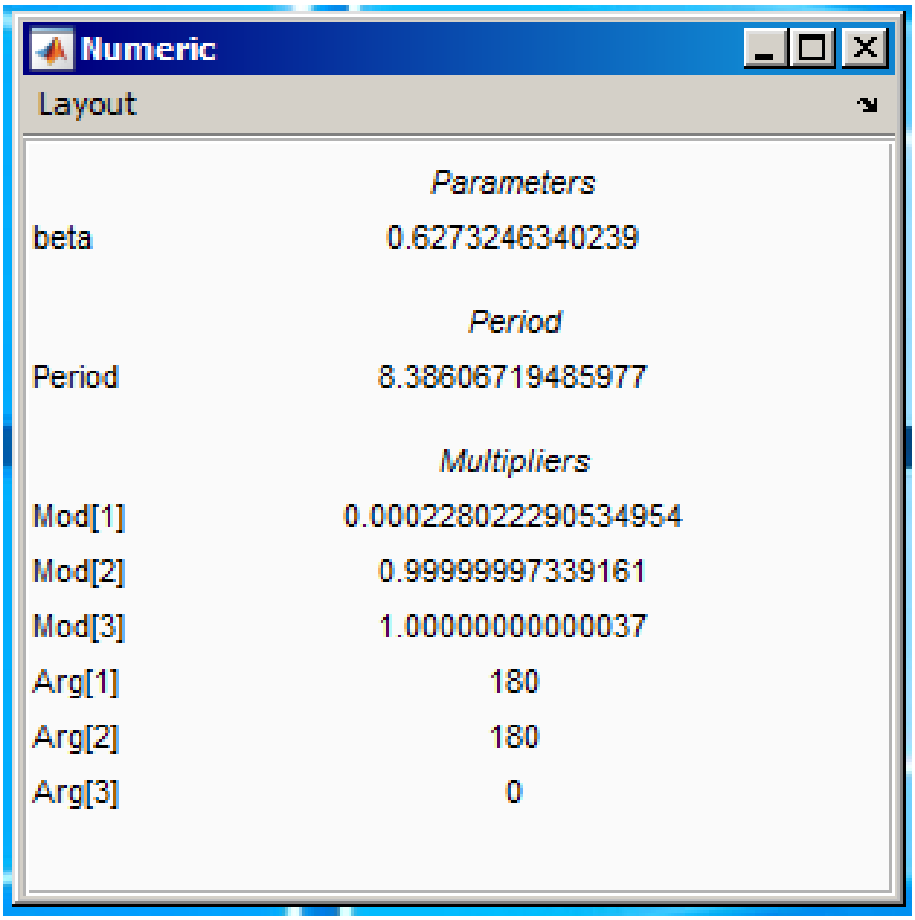


Figure C.17: The **Numeric** window at the first period-doubling bifurcation. The bifurcation parameter value is $\beta = 0.6273246\ldots$.

indicating that in both cases the normal form coefficients are negative, so that stable double-period cycles are involved.

You get a cycle manifold shown in Figure C.19. The period of the cycle rapidly increases when β approaches $\beta_{\rm hom} = 0.515489\ldots$. To see this, open a **2Dplot** window (see Figure C.20) with the axes `beta` and `Period` and the plotting region

```
Abscissa:      0.51       0.53
Ordinate:      0.0       35.0
```

Actually, the limit cycle approaches a *homoclinic orbit* to the second equilibrium $(x, y, z) = (1, 0, 0)$ of (C.2), which is a global bifurcation[1]. Below $\beta_{\rm hom}$ no cycle exists. Near the homoclinic bifurcation, the computation of the cycle and its multipliers becomes inaccurate.

[1]At $\beta = \beta_{\rm hom}$, this equilibrium is a saddle-focus with the *saddle quantity* $\sigma < 0$. Thus, according to Shilnikov's Theorem, from this homoclinic orbit only one limit cycle bifurcates - the one we computed.

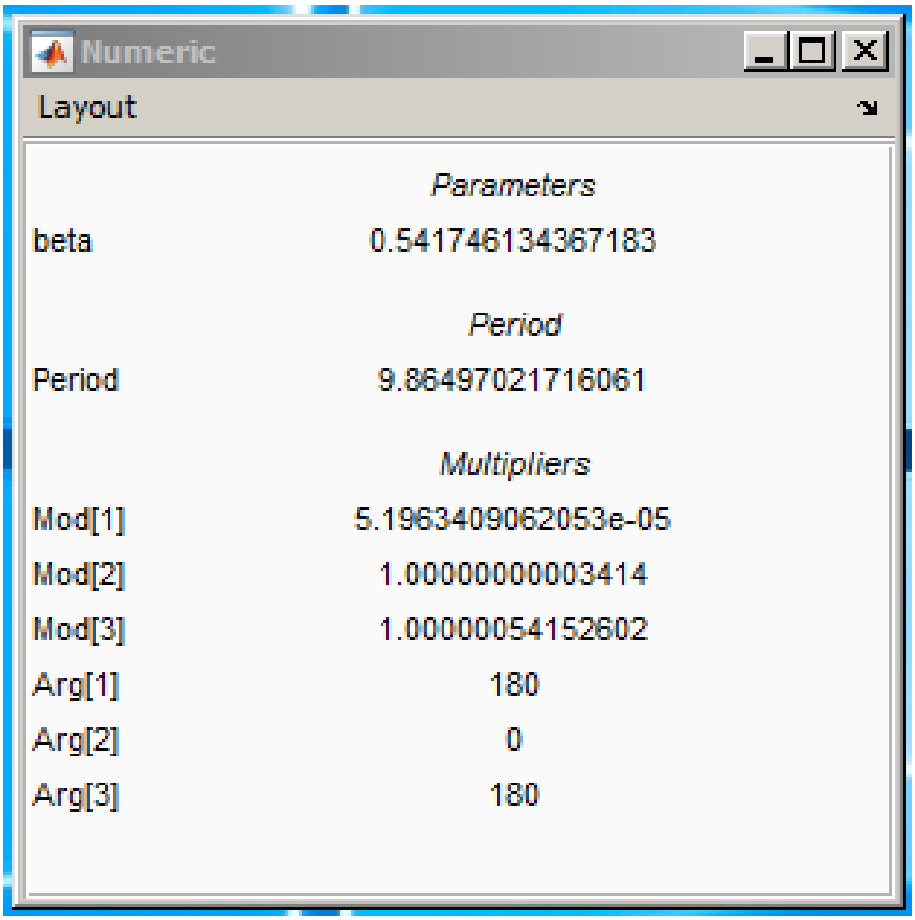


Figure C.18: The **Numeric** window at the second period-doubling bifurcation. The bifurcation parameter value is $\beta = 0.5417461\ldots$.

The loss of accuracy can be seen in the **Numeric** window, where no multiplier close to 1 is present at the last computed point.

C.3.4 Continuation of the double cycle from the PD-point

Using **Select**|**Initial point**, take the first `PD: Period Doubling` point in the computed above curve as initial.

Adjust the **3Dplot** window by setting the visibility limits for `beta` as follows:

```
Abscissa:          0.5          0.63
```

In the **Starter** window, set the number of mesh points `ntst` to 80, increase the `amplitude` to 0.005, and select `yes` only to `Branching` in the `Monitor Singularities` section.

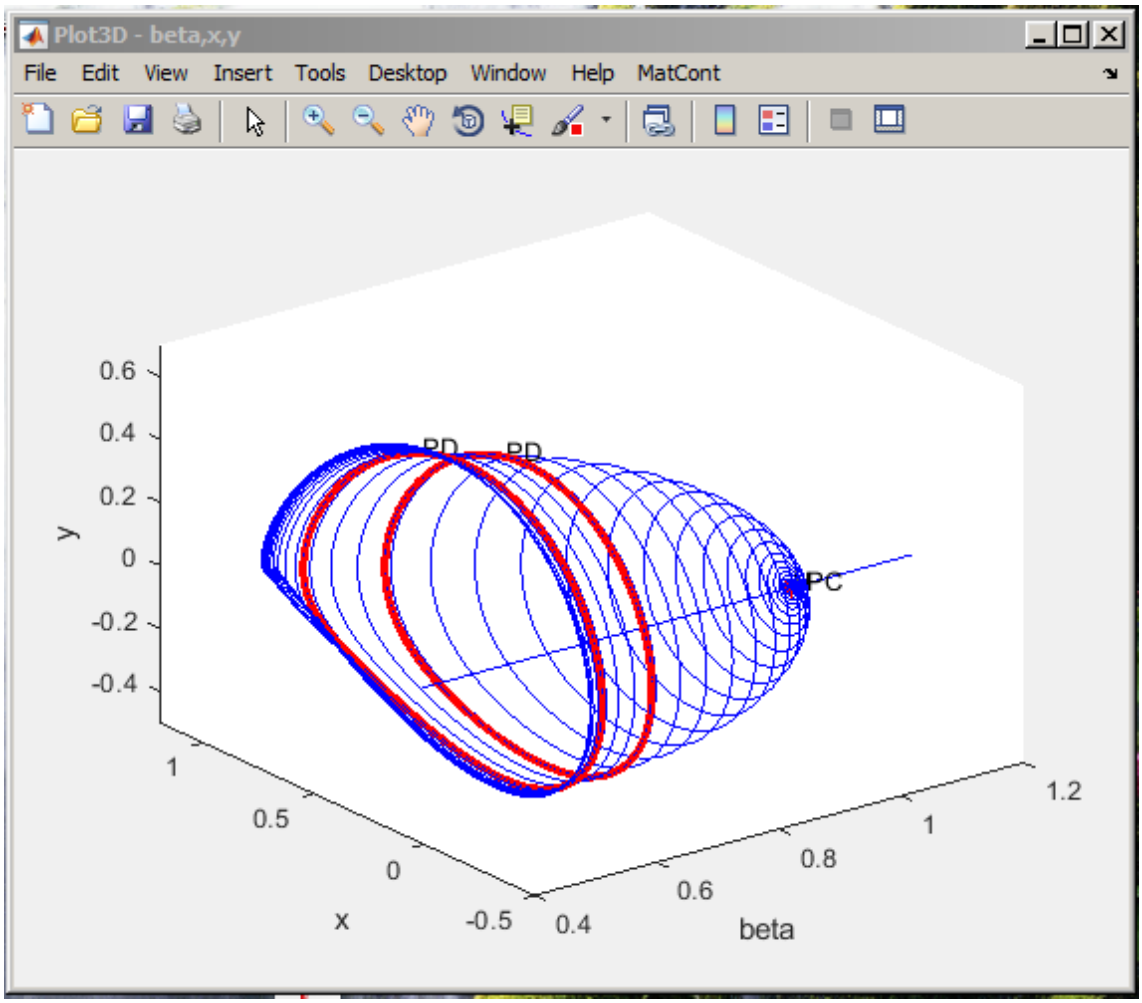


Figure C.19: The cycle branch in system (C.2) rooted at the Hopf point. Two critical cycles undergoing the period-doubling bifurcations are labeled by PD.

In the **Continuer** window, increase MaxTestIters to 30, and make sure that MaxStepsize remains equal to 0.7.

Click **Compute|Forward** and continue the doubled cycle bifurcating from the first PD-point at $\beta = \beta_{\mathrm{PD},1}$, see Figure C.21. This cycle remains stable until at (approximately) $\beta = \beta_{\mathrm{PD},2}$ a *branching point cycle* (BPC) is detected. This is not surprising, since the PD-bifurcation for the original limit cycles corresponds to a branching point for the doubled cycles.

Stop computations at BPC and exit MatCont.

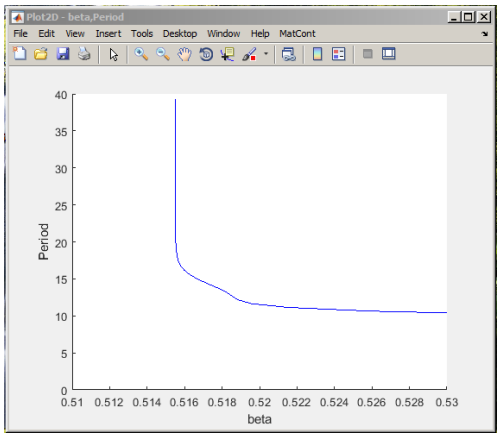


Figure C.20: The cycle period near the homoclinic bifurcation.

C.4 Additional Problems

A. Study numerically limit cycles in the system

$$\begin{cases} \dot{x} &= \mu x - y - xz, \\ \dot{y} &= x + \mu y, \\ \dot{z} &= -z + y^2 + x^2 z, \end{cases}$$

that is another model of a feedback control system.

B. Consider the following *predator-prey* model

$$\begin{cases} \dot{x} &= x - \dfrac{xy}{1+\alpha x}, \\ \dot{y} &= -y + \dfrac{xy}{1+\alpha x} - \delta y^2. \end{cases}$$

1. Prove that the following polynomial system

$$\begin{cases} \dot{x} &= x(1+\alpha x) - xy, \\ \dot{y} &= -(y+\delta y^2)(1+\alpha x) + xy, \end{cases} \tag{C.3}$$

has the same orbits in the positive quadrant as the original model. Derive an explicit condition on (α, δ) for system (C.3) to have a positive equilibrium with one zero eigenvalue. Derive a similar condition for (C.3) to have a positive equilibrium with a pair of purely imaginary eigenvalues.

2. Introduce new variables

$$\begin{cases} \xi &= \ln x, \\ \eta &= \ln y, \end{cases}$$

in which the original system becomes:

$$\begin{cases} \dot{\xi} &= 1 - \dfrac{\exp(\eta)}{1+\alpha\exp(\xi)}, \\ \dot{\eta} &= -1 + \dfrac{\exp(\xi)}{1+\alpha\exp(\xi)} - \delta\exp(\eta). \end{cases} \tag{C.4}$$

Fix $\alpha = 0.3$ and study numerically limit cycles of (C.4), when the parameter δ varies. *Hint*: Begin with finding an equilibrium at $\delta = 0.35$ by integration.

3. Relate your analytical and numerical results.

C. Consider the following *predator-double prey* system:

$$\begin{cases} \dot{x} &= x(2.4 - x - 6y - 4z), \\ \dot{y} &= y(\beta - x - y - 10z), \\ \dot{z} &= -z(1 - 0.25x - 4y + z), \end{cases}$$

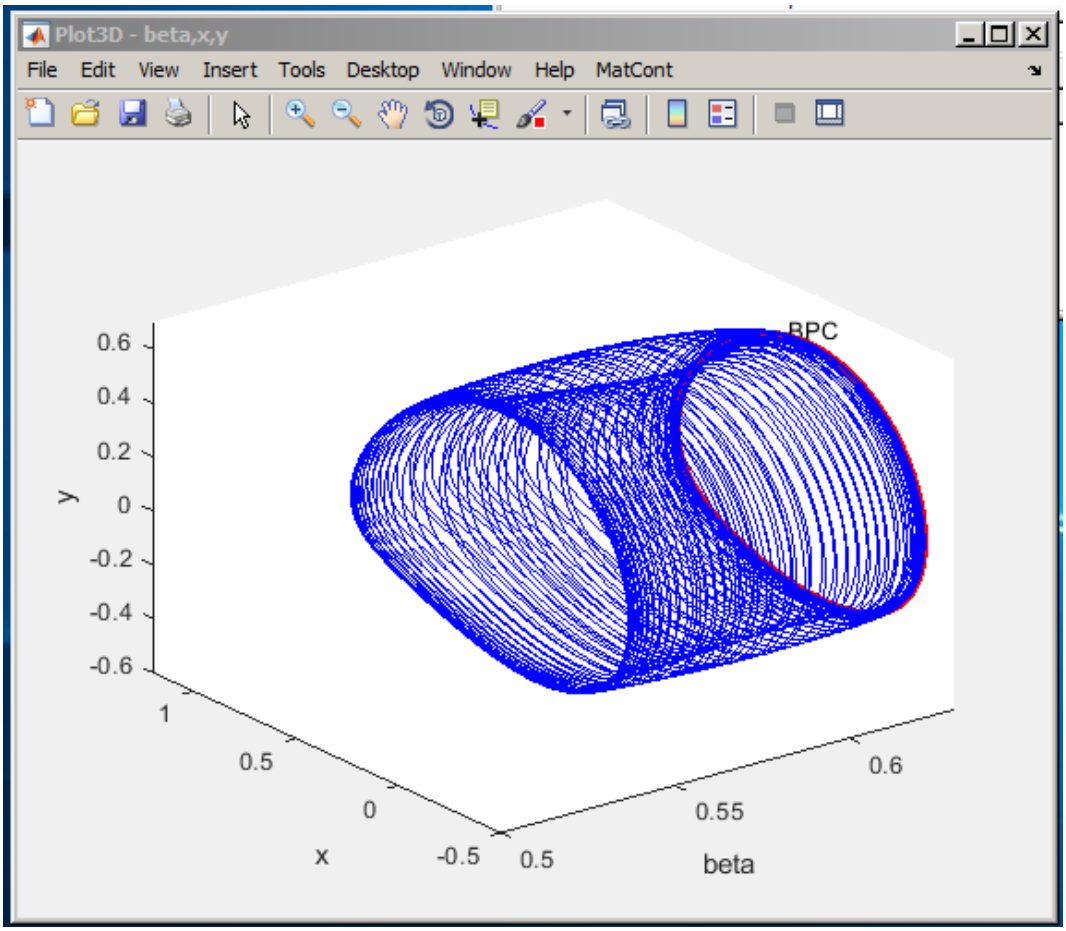


Figure C.21: The branch of stable doubled cycles connects two PD points.

modelling dynamics of two prey populations affected by a predator.

1. Starting from $\beta = 1.77$, find its *positive* equilibrium by orbit integration.
2. Continue the equilibrium with respect to β until it exhibits a Hopf bifurcation.
3. Continue a limit cycle from the Hopf point and monitor the dependence of its period upon β. *Hint*: Set `ntst` to a high value.
4. Plot the cycle at different period values and try to understand its asymptotic shape. *Hint*: Compute and analyze the equilibria located on the coordinate planes.

D. Consider the famous *Lorenz system*

$$\begin{cases} \dot{x} &= \sigma(-x+y), \\ \dot{y} &= rx - y - xz, \\ \dot{z} &= -bz + xy, \end{cases}$$

with the standard parameter values $\sigma = 10, b = \frac{8}{3}$ and $r > 0$. Use MatCont to analyse its simplest limit cycles numerically.

1. Compute a critical parameter value $r_{\mathtt{BP}}$, at which the trivial equilibrium $(0,0,0)$ undergoes a *symmetric* pitchfork bifurcation.
2. Compute a critical parameter value $r_{\mathtt{H}}$, corresponding to a Hopf bifurcation of *nontrivial* equilibria.
3. Continue the limit cycle born at the Hopf bifurcation with respect to r and find out what happens to it. Plot the cycle period as function of r, and monitor stability.
4. Demonstrate by numerical integration that for $r = 400$ the system has a stable symmetric limit cycle and determine its period approximately. Continue this cycle with respect to r and find the critical parameter value $r_{\mathtt{BPC}}$ corresponding to a *branch point of cycles* (*symmetry breaking*). *Hint*: Use the procedure to start `LC` continuation from a simulation as done in Section 1, for the *Rössler* system.

Derive explicit formulas for $r_{\mathtt{BP}}$ and $r_{\mathtt{H}}$ as functions of (σ, b) and verify the numerical values obtained with MatCont.

Appendix D

TUTORIAL IV: Using the new MATCONT GUI for two-parameter bifurcation analysis of equilibria and limit cycles

This session was tested on MATCONT7p1 with MATLAB2017b. It is devoted to the numerical continuation of codim 1 bifurcations of equilibria and limit cycles in systems of autonomous ODEs depending on two parameters

$$\dot{x} = f(x, \alpha), \quad x \in \mathbb{R}^n, \alpha \in \mathbb{R}^2,$$

and detection of their codim 2 bifurcations. We will also switch at some codim 2 equilibrium bifurcations to the continuation of codim 1 bifurcation curves rooted there.

D.1 Bifurcations of equilibria in the Bykov–Yablonskii–Kim model

We will use MATCONT to continue equilibria and their bifurcations in the following chemical model that describes CO-oxidation on platinum:

$$\begin{cases} \dot{x} &= 2k_1 z^2 - 2k_{-1} x^2 - k_3 xy \\ \dot{y} &= k_2 z - k_{-2} y - k_3 xy \\ \dot{s} &= k_4 z - k_{-4} s, \end{cases}$$

where $z = 1 - x - y - s$. The ratio

$$K = \frac{k_{-4}}{k_4}$$

and the parameter k_2 will be used as two bifurcation parameters.

D.1.1 Specify the model in MatCont

Specify a new ODE system in MatCont with coordinates (`x,y,s`) and time (`t`):

```
z=1-x-y-s
x'=2*Q1*z^2-2*Q5*x^2-Q3*x*y
y'=Q2*z-Q6*y-Q3*x*y
s'=Q4*z-K*Q4*s
```

The reaction rates $k_1, k_2, k_3, k_4, k_{-1}, k_{-2}$ are denoted by `Q1, Q2, Q3, Q4, Q5, Q6`, respectively, while `K` stands for K. Generate the derivatives of order 1, 2, 3, 4 and 5 symbolically. Note that z is an *auxiliary variable* which is introduced to simplify the formal form of the equations and to improve the readability. The choice of the name is irrelevant but it should not conflict with the internal names in the MATLAB symbolic toolbox if that toolbox is used.

D.1.2 Equilibrium continuation

Select both **Type|Initial point|Equilibrium** and **Type|Curve|Equilibrium**. In the **Starter** window input the following numerical values, corresponding to a stable equilibrium in the system:

```
x               0.001137
y               0.891483
s               0.062345
Q1              2.5
Q2              2.204678
Q3              10
Q4              0.0675
Q5              1
Q6              0.1
K               0.4
```

Activate the parameter `Q2`.

In the **Continuer** window, decrease the `MaxStepsize` along the curve to

```
MaxStepsize          0.025
```

Open a **2Dplot** window with (`Q2,x`) on the axes and input the following visibility limits:

```
Abscissa:          0.5              2.0
Ordinate:          0.0              0.16
```

Also open a **Numeric** window and make `Eigenvalues` visible using **Numeric|Layout**.

Start the continuation of the equilibrium curve with the **Compute|Forward** command. Monitor the eigenvalues in the **Numeric** window. There will be four bifurcation points

detected: two *Hopf points* (labeled by `H`) and two *fold points* (labeled by `LP`). Resume computations at each bifurcation point, and stop them when the curve leaves the window. You should get the central S-shaped curve presented in Figure D.1. In a narrow interval of `Q2`-values (between two `LP`-points) the system has three equilibria.

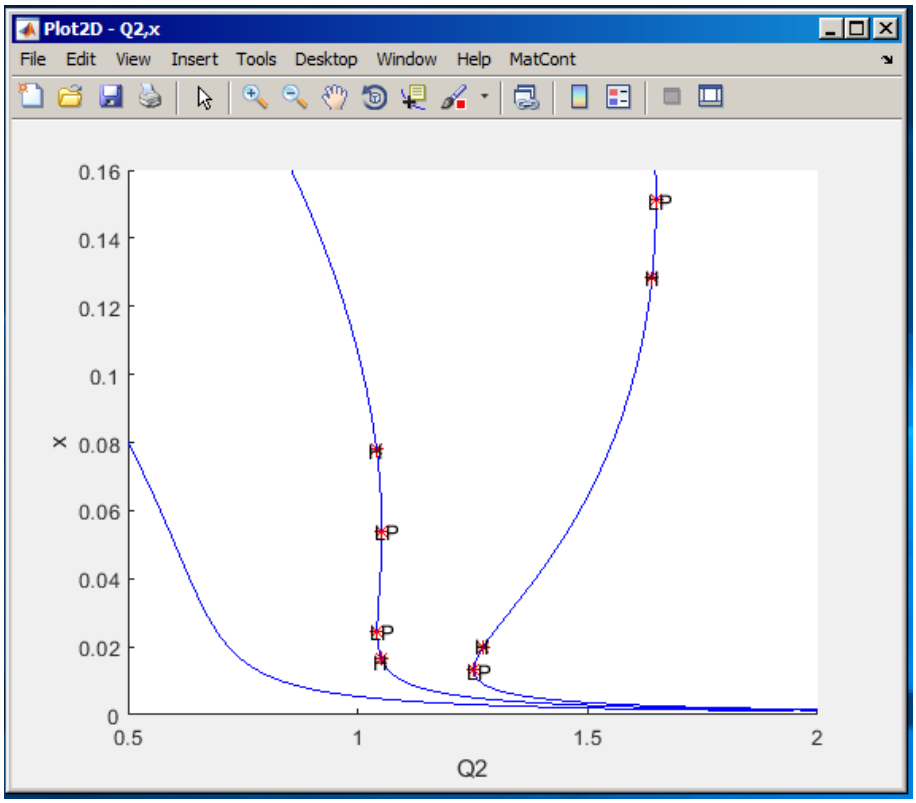


Figure D.1: Equilibrium curves with $K = 0.15, 0.4$, and 2.0

The normal form coefficients for the Hopf and limit point bifurcations can be read in the MATLAB Command Window:

```
label = H , x = ( 0.016357 0.523973 0.328336 1.051558 )
First Lyapunov coefficient = 1.070259e+01
label = LP, x = ( 0.024717 0.450257 0.375018 1.042049 )
a=1.166509e-01
label = LP, x = ( 0.054030 0.302241 0.459807 1.052200 )
a=-1.346534e-01
label = H , x = ( 0.077929 0.233063 0.492149 1.040991 )
First Lyapunov coefficient = 4.332247e+00
```

Note that both first Lyapunov coefficients are positive (implying that unstable limit cycles bifurcate there), while the LP-coefficients are both nonzero (their sign is irrelevant and depends on the choice of a direction).

Rename the computed equilibrium curve via **Select**|**Curve** and **Select**|**Curve**|**Rename** into

```
Equilibrium1(+)
```

Change the value of the parameter K in the **Starter** window:

```
K          0.15
```

and repeat the computations with **Compute**|**Forward**. You will get a monotone curve without bifurcation points (the left curve in Figure D.1). There is only one equilibrium for all parameter values Q2.

Finally, change the value of K to

```
K          2.0
```

and click on **Compute**|**Forward** again. Another equilibrium curve will come out with the order of the singularities reversed (the right curve in Figure D.1). There is now a big interval of the parameter values, in which there are three equilibria in the system.

In the MATLAB Command Window, the following messages appeared:

```
label = LP, x = ( 0.013310 0.771372 0.071773 1.252630 )
a=1.705787e+00
label = H , x = ( 0.019639 0.726319 0.084681 1.271109 )
Neutral saddle
label = H , x = ( 0.128699 0.384106 0.162399 1.640254 )
Neutral saddle
label = LP, x = ( 0.151168 0.344167 0.168222 1.648672 )
a=-1.787355e+00
```

indicating that the LP-bifurcations are still nondegenerate ($a \neq 0$), while – instead of two Hopf points – two *neutral saddles* have appeared, which are not bifurcation points. Note that they are still marked by H.

Store the last two computed curves permanently by renaming them into

```
Equilibrium2(+)
Equilibrium3(+)
```

via **Select** |**Curve** | **Rename**. Close the **Data Browser** window.

D.1.3 Continuation of fold and Hopf bifurcations

This section explains how to continue fold and Hopf bifurcations with respect to two control parameters. While doing so, you will see how MatCont detects some codim 2 bifurcations and reports their normal form coefficients.

Fold continuation

Select the second `LP: Limit point` in the first computed curve `Equilibrium1(+)` via **Select** | **Curve** | **Equilibrium1(+)** | **LP: Limit point** | **Select point**. The new **Starter** window corresponding to the continuation of a limit point curve `LP_LP` is to appear, in which you should keep parameter `Q2` active and activate parameter `K`, see Figure D.2.

Compute the limit point curve in both directions to get Figure D.3. Notice that both

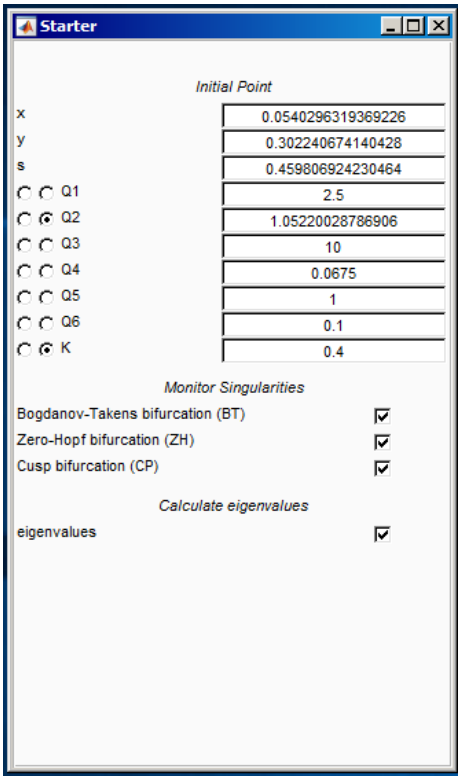


Figure D.2: **Starter** window for the continuation of the limit point curve.

parameters (`Q2,K`) are varied along the curve. Each point on the curve is a limit point for the equilibrium curve at the corresponding value of `K` (or
Q2).

Three points were detected, corresponding to codim 2 bifurcations: two *Bogdanov-Takens* (`BT`) and one *cusp* (`CP`). At each `BT` point the system has an equilibrium with a double zero eigenvalue, while at the `CP` point there is an equilibrium with a simple zero eigenvalue but the normal form coefficient of the fold is zero. The normal form coefficients (a, b, c) of these bifurcations are reported in the MATLAB Command Window and they are all nonzero:

```
label = BT , x = ( 0.115909 0.315467 0.288437 1.417628 0.971397 )
(a,b)=(-8.378442e-02, -2.136280e+00)
```

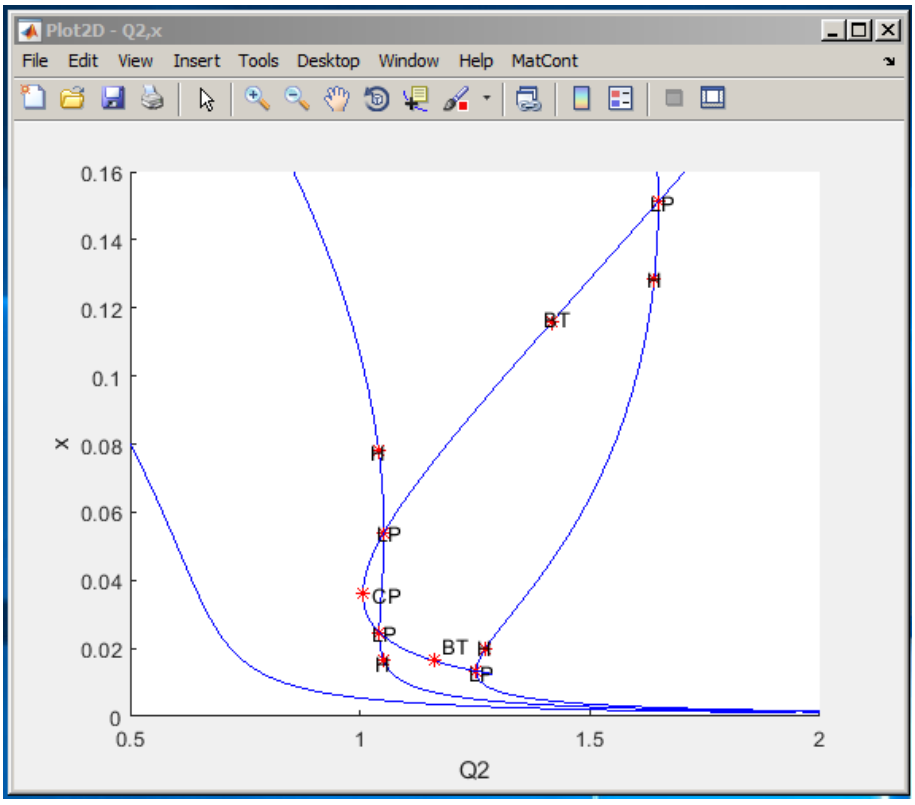


Figure D.3: Limit point curve in Bykov's model: `BT` - Bogdanov-Takens points; `CP` - cusp.

```
label = CP , x = ( 0.035940 0.352008 0.451368 1.006408 0.355991 )
c=3.627844e-01
label = BT , x = ( 0.016337 0.638410 0.200456 1.161199 0.722339 )
(a,b)=(-4.822563e-02, -1.937632e+00)
```

Rename the obtained limit point branches as following:

```
Fold(+)
Fold(-)
```

Hopf continuation

The Bogdanov-Takens points are common points for the limit point curves and curves corresponding to equilibria with eigenvalues $\lambda_1 + \lambda_2 = 0, \lambda_3 \neq 0$. Actually, at each `BT` point, the Hopf bifurcation curve (with $\lambda_{1,2} = \pm i\omega_0, \omega_0 > 0$) turns into the neutral saddle curve (with real $\lambda_1 = -\lambda_2$). Thus, we can start a Hopf curve from a Bogdanov-Takens point.

Select the `BT: Bogdanov-Takens point` in the curve `Fold(+)` as initial. Then choose **Type |Curve|Hopf** to prepare for the continuation of the Hopf curve. The new **Starter** window for the continuation of a Hopf curve will appear automatically, see Figure D.4.

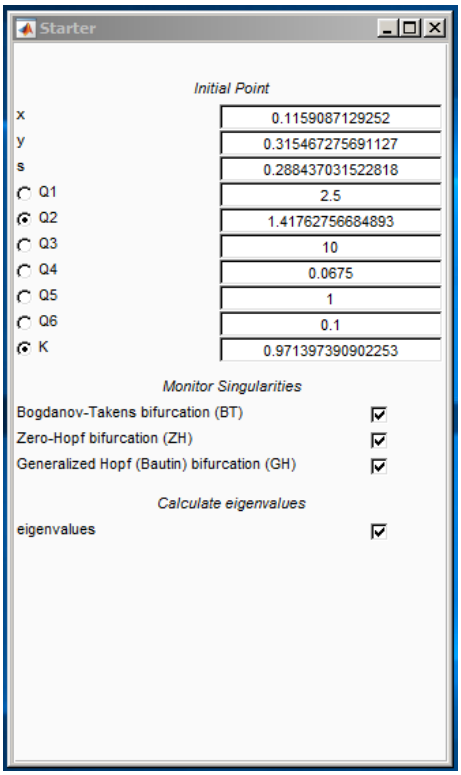


Figure D.4: **Starter** window for the continuation of the Hopf curve from the `BT`-point.

Check that the type of the curve to be computed is changed in the MatCont window. Activate the parameters $Q2, K$.

Compute|Forward (resuming the computation at each special point and using **Compute |Extend**) produces a *closed* Hopf curve as in Figure D.5. Actually, only the part of it to the left of two `BT` points corresponds to a Hopf bifurcation; the other part represents a neutral saddle. As it has been mentioned, this transition happens at the Bogdanov-Takens points. There are two more codim 2 bifurcation points at the Hopf point: Two *Bautin* or *generalized Hopf* (`GH`) points, where the first Lyapunov coefficient l_1 vanishes.

The Command Window of MATLAB shows a message about the starting `BT` point and the following messages related to the `GH` points:

```
label = GH, x = ( 0.018022 0.368238 0.497968 0.891319 0.232487 0.003324 )
l2=-7.768996e+02
label = GH, x = ( 0.064311 0.211095 0.554870 0.924255 0.305879 0.003512 )
l2=-2.401233e+02
```

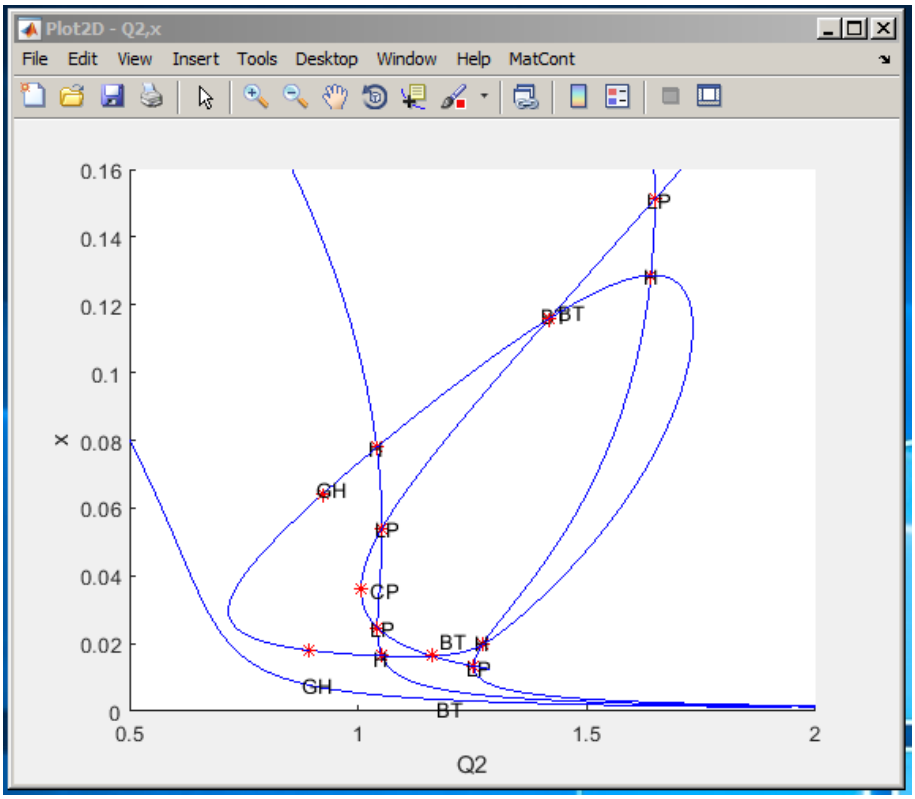


Figure D.5: The parameter plot with the added Hopf curve in Bykov's model: `GH` - generalized Hopf points

indicating that both `GH` points are nondegenerate, since the second Lyapunov coefficients `l2` are nonzero (in fact, negative).

Two-parameter bifurcation diagram

Rename the obtained Hopf curve into

`Hopf(+)`

and delete all computed equilibrium curves.

Open a new **2Dplot** window to plot the bifurcation diagram in the `(Q2,K)`-plane with the visibility limits

```
Q2        0.7      1.5
K         0.1      1.1
```

Upon redrawing the diagram in the `(Q2,K)`-window, you will get Figure D.6.

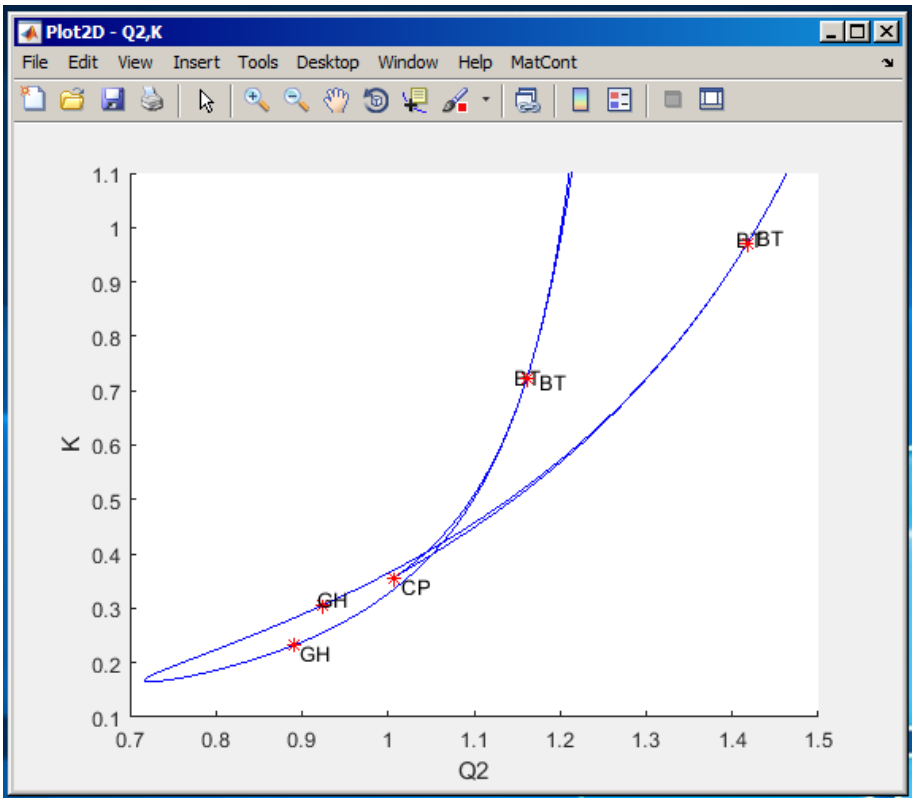


Figure D.6: Fold and Hopf curves in Bykov's model: `BT`- Bogdanov-Takens, `CP` -cusp, `GH` - generalized Hopf points

The edge between two fold curves is a parameter region where the system has three equilibria. Crossing a Hopf curve results in the appearance of either stable or unstable periodic orbit. There are other bifurcations in the model.

D.2 Fold and torus bifurcations of cycles in the Steinmetz–Larter model

Consider the following chemical model:

$$\begin{cases} \dot{A} &= -k_1ABX - k_3ABY + k_7 - k_{-7}A, \\ \dot{B} &= -k_1ABX - k_3ABY + k_8, \\ \dot{X} &= k_1ABX - 2k_2X^2 + 2k_3ABY - k_4X + k_6, \\ \dot{Y} &= -k_3ABY + 2k_2X^2 - k_5Y. \end{cases}$$

We will study bifurcations of limit cycles of this model when parameters (k_7, k_8) vary. Note: If you have worked out TUTORIAL III, the system was already introduced in the

second section. In this case, recompile the system with symbolic derivatives up to order 5 and delete all previously computed curves.

D.2.1 System specification

Specify a new ODE system – say `STLAR` – in MatCont

```
A'=-k1*A*B*X-k3*A*B*Y+k7-km7*A
B'=-k1*A*B*X-k3*A*B*Y+k8
X'=k1*A*B*X-2*k2*X^2+2*k3*A*B*Y-k4*X+k6
Y'=-k3*A*B*Y+2*k2*X^2-k5*Y
```

where (`A,B,X,Y`) are the coordinates and (`k1,k2,k3,k4,k5,k6,k7,km7,k8`) are the parameters. Use (default) `t` for time and generate symbolically partial derivatives of order 1, 2, 3, 4 and 5.

D.2.2 Preliminary one-parameter analysis

Continuation of an equilibrium

To begin with, we continue an equilibrium of the model and detect its Hopf bifurcation.

Input **Type**|**Initial point**|**Equilibrium** in the main MatCont window.

Input the following numerical data in the appearing **Starter** window:

```
A                31.78997
B                 1.45468
X                 0.01524586
Y                 0.1776113
k1                0.1631021
k2             1250
k3                0.046875
k4               20
k5                1.104
k6                0.001
k7                4.235322
km7               0.1175
k8                0.5
```

These values correspond to an equilibrium (A, B, X, Y) in the system. Activate the parameter `k7`.

Open **Window/Output** | **Numeric** and change its appearance via the **Numeric** | **Layout** command. Namely, select `Eigenvalues` and `stepsize` to be shown in the window.

Start **Compute**|**Forward**. The equilibrium curve will be continued and you get a Hopf bifurcation. The message in the MATLAB Command Window

```
label = H , x = ( 34.808899 1.328517 0.015246 0.177611 4.590046 )
First Lyapunov coefficient = 1.527549e-02
```

at $k_7 = 4.590046\ldots$ indicates a *subcritical* Hopf bifurcation. Indeed, there are two eigenvalues of the equilibrium with Re $\lambda_{1,2} \approx 0$ at this parameter value visible in the **Numeric** window. The critical frequency Im $\lambda_1 \neq 0$, while the first Lyapunov coefficient is positive. Thus, there should exist an unstable limit cycle, bifurcating from the equilibrium.

Note also that in the Hopf point, as in all other bifurcation points, the `stepsize` is indicated as 0 in the **Numeric** window. The reason for this is that bifurcation points are not located by continuation but by a variant of the bisection method.

Stop computations and rename the computed curve via **Select** | **Curve** | **Rename** into

```
Equilibrium(+)
```

Compute|**Backward** with **Compute**|**Extend** once until you get a *Hopf bifurcation* at $k_7 = 0.712475\ldots$ with the message

```
label = H , x = ( 1.808301 25.573303 0.015246 0.177611 0.712475 )
First Lyapunov coefficient = -2.371880e-02
```

in the MATLAB Command Window. The first Lyapunov coefficient is negative now. This means that a *stable* limit cycle bifurcates from the equilibrium, when it looses stability. Stop computations and rename the computed curve into `Equilibrium(-)`.

Cycle continuation

Select the `H: Hopf` point in the curve `Equilibrium(-)` as initial. MatCont will prepare to continue a limit cycle curve from the Hopf point (curve type `H_LC`). Choose the `yes`-option to monitor all singularities in the **Starter** window and set `amplitude` to 0.001. Choose k_7 and **Period** as the free parameters. Set

```
MaxStepsize              1
MaxNumPoints            25
```

in the **Continuer** window. Change the appearance of the **Numeric** window via the **Window**|**Layout** command by selecting `Multipliers` to be shown.

Click **Compute**|**Forward** to start the continuation of the limit cycle. At $k_7 = 0.716434\ldots$ the message `Neimark-Sacker` indicates a *torus* bifurcation. In the MATLAB Command Window, the following message appears:

```
Neimark-Sacker (period = 1.091213e+01, parameter = 7.164336e-01)
Normal form coefficient = -4.912065e-08
```

Indeed, there are two complex multipliers with (approximately) $|\mu| = 1$ and one trivial multiplier (approximately) equal to 1. This can be seen in the **Numeric** window. The normal form coefficient is small but negative, indicating that a stable two-dimensional *invariant torus* bifurcates from the limit cycle.

Rename the computed curve into `cycle`.

D.2.3 Two-parameter analysis

Continuation of the Hopf bifurcation curve

Select the `H: Hopf point` in the computed `Equilibrium(+)` curve starting with the **Select | Curve** command. Then navigate in the **Data Browser** to the `Equilibrium(+)` curve, double-click it and then select the Hopf point.

Select **Type|Curve|Hopf**. Activate two parameters, namely `k7` and `k8`, in the **Starter** window.

Open a new **2Dplot** window with the parameters `k7` and k8 as abscissa and ordinate, respectively, and the visibility limits:

```
Abscissa        0           7
Ordinate        0.3         1
```

Continue the Hopf bifurcation curve with **Compute|Forward** followed by **Compute|Extend**. At

$$(k_7, k_8) = (6.336084\ldots, 0.413039\ldots)$$

a *generalized Hopf* bifurcation will be found, where the first Lyapunov coefficient vanishes (label `GH`). Resume the continuation and terminate it when the Hopf curve leaves the window. In the MATLAB Command Window, the following message has appeared:

```
label = GH, x = ( 50.40856 0.97900 0.01342 0.13184 6.33604 0.41303 0.59436 )
l2=3.325910e-03
```

from which it follows that this codim 2 bifurcation is nondegenerate (the second Lyapunov coefficient `l2` is nonzero). Rename the computed curve to `Hopf(+)`.

The backward continuation of the Hopf bifurcation curve reveals one more generalized Hopf point at

$$(k_7, k_8) = (0.999480\ldots, 0.645896\ldots).$$

Resume and extend until the curve leaves the window. You should get Figure D.7.

The MATLAB Command Window contains the message:

```
label = GH, x = ( 3.00921 14.18442 0.01797 0.26026 0.99947 0.64589 0.29229 )
l2=-4.044531e-03
```

indicating that this `GH` is also nondegenerate. Rename this curve to `Hopf(-)`.

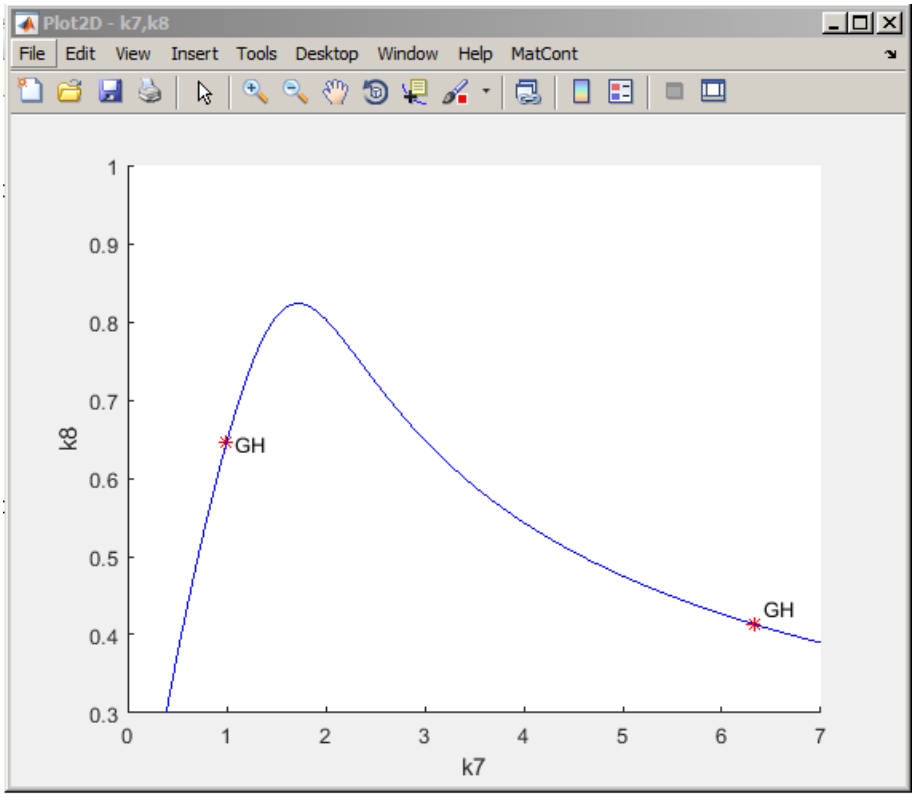


Figure D.7: The Hopf bifurcation curve with two GH's - generalized Hopf points.

Continuation of the LPC-bifurcation starting at the GH-point

Using **Select | Curve** and the **Data Browser** select `GH: Generalized Hopf point` in the `Hopf(+)` curve as initial point. Since it is known that an `LPC` curve originates there, select **Type|Curve|Limit point of cycles** to continue. The **Starter** and **Continuer** windows for the continuation of the fold bifurcation of cycles from the generalized Hopf point will appear.

In the **Starter** window, set `yes` to monitor for all singularities except the `Cusp point` of cycles and increase the `amplitude` of the predicted `LPC` to 0.01. To increase speed set `ntst` to 20 and check that `ncol` is 4. In the **Continuer** window, set `MaxStepsize` equal to 2.0 and `MaxNumPoints` equal to 300. The **Starter** and **Continuer** windows should look like in Figure D.8.

Compute|Forward the `LPC`-bifurcation curve (with **Compute|Extend**) and observe that it actually connects the two found `GH` points in the Hopf curve, see Figure D.9. The continuation (which takes some time) stops with the message `Current step size too small` near the left `GH` point.

The computed curve will also contain two extra codim 2 points labeled by `R1`. These are `strong resonance` 1:1 points, where the cycle has a triple multiplier 1 (counting the

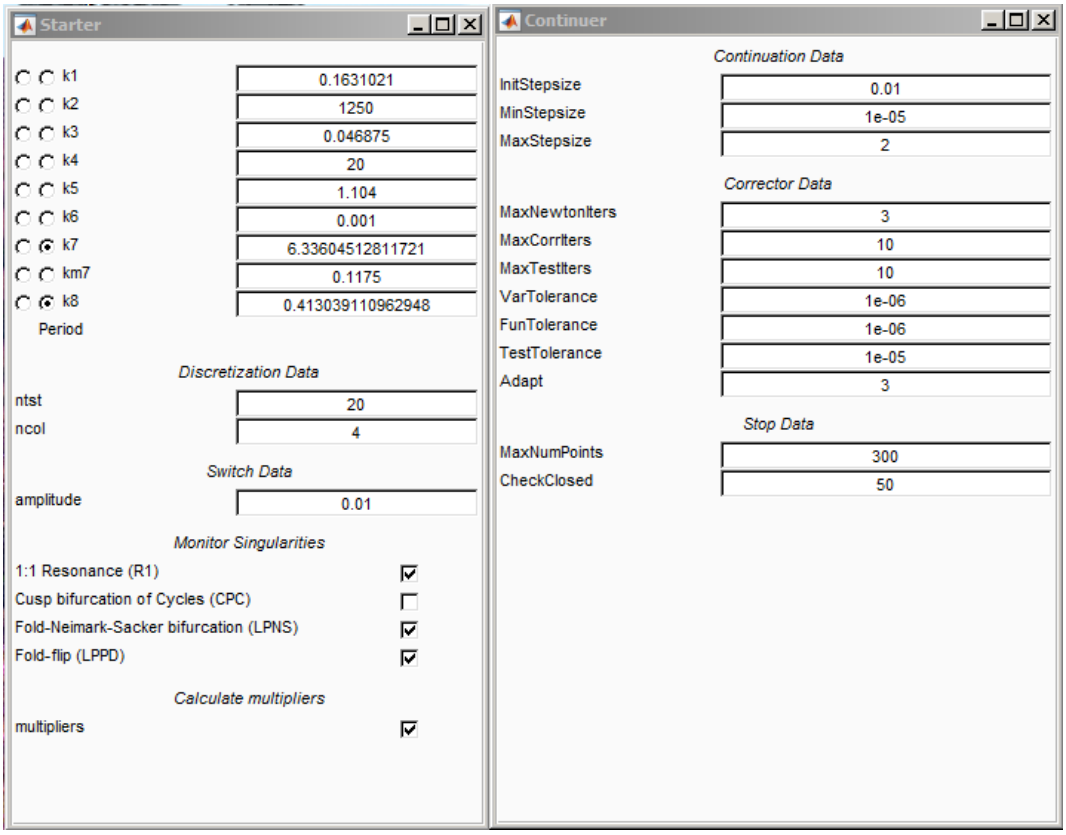


Figure D.8: The **Starter** and **Continuer** windows for the LPC-continuation from the GH-point.

trivial multiplier). In the **Numeric** window, you can read the corresponding values of (k_7, k_8). The MATLAB Command Window gives the following messages:

```
Resonance 1:1 (period = 1.40019e+01, parameters = 1.85767e+00, 9.30422e-01)
ab=1.432638e-01
Resonance 1:1 (period = 1.23945e+01, parameters = 1.17955e+00, 7.23957e-01)
ab=-2.686129e-03
```

where the critical cycle period and parameter values, as well as the product of the normal form coefficients for `R1`, are reported.

Rename the computed `GH_LPC` curve to `cyclefold`.

Continuation of the `NS`-curve in two parameters

Take the `NS: Neimark-Sacker` point in the limit cycle curve `cycle` as initial point. By default, the Curve Type is `Neimark-Sacker`. Activate k_7, k_8 as free parameters.

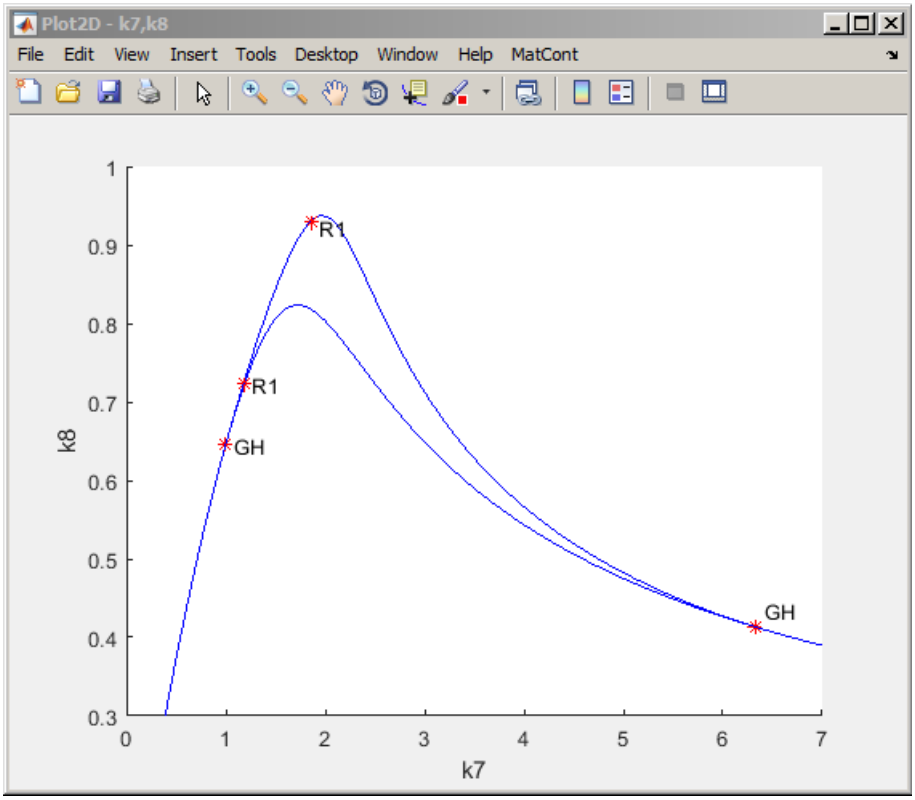


Figure D.9: Hopf and limit point of cycles curves connecting the GH-points.

To speed up the continuation, untick the boxes for monitoring any singularity. Compute the multipliers, though. Set `MaxNumPoints` to 300.

Change the plotting region of the **2Dplot** window to

`Abscissa`	`1`	`2.2`
`Ordinate`	`0.6`	`1`

and redraw the diagram.

Click **Compute|Forward** and wait until the computed `NS`-curve enters the window (this takes some time). Extend the computation several times to obtain (after some editing) Figure D.10.

The computed `NS`-curve passes through the `R1` points of strong resonance 1:1, where it tangentially meets the `LPC`-curve and turns from the Neimark-Sacker bifurcation curve into the non-bifurcation neutral saddle cycle curve (both are characterised by the presence of two multipliers with $\mu_1\mu_2 = 1$).

Warning: There are many other bifurcations in the model and Figure D.10 shows only a few of them.

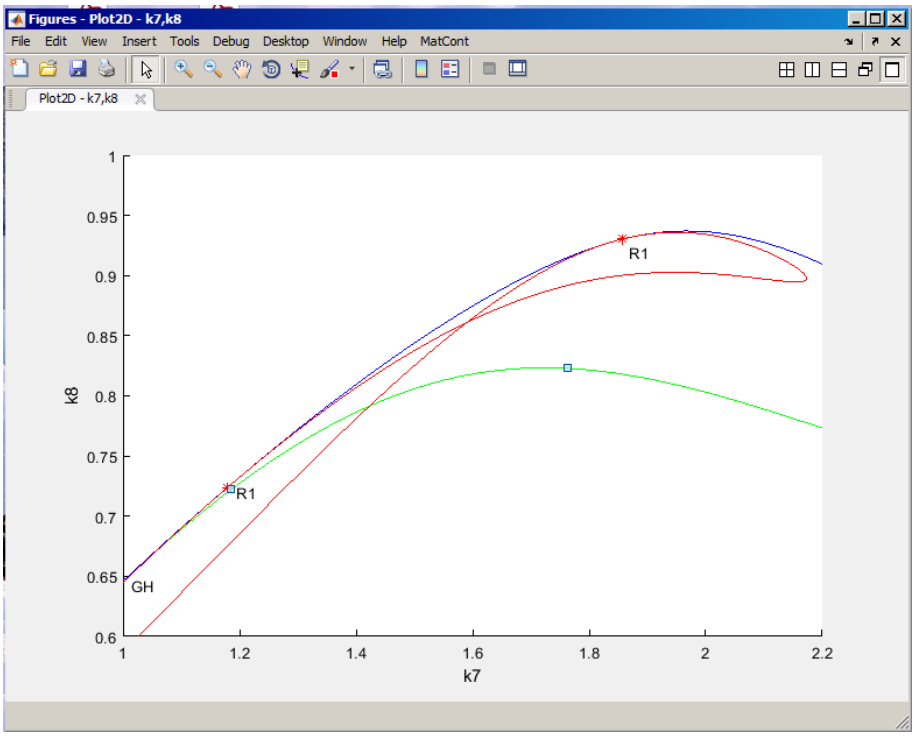


Figure D.10: Hopf (green), limit point of cycles (blue) and Neimark-Sacker (red) curves: The NS-curve between two R1-points corresponds to neutral saddle cycles.

D.3 Additional Problems

A. Consider the following system by Lorenz [1984]:

$$\begin{cases} \dot{x} &= -y^2 - z^2 - ax + aF, \\ \dot{y} &= xy - bxz - y + G, \\ \dot{z} &= bxy + xz - z, \end{cases} \tag{D.1}$$

where (a, b, F, G) are parameters.

1. Using MatCont, compute fold and Hopf bifurcation curves for equilibria of (D.1) with $a = \frac{1}{4}, b = 4$, in the parameter domain

$$\{(G, F) : 0 \leq G \leq 3, 0 \leq F \leq 3\}.$$

Find numerical parameter values $(G_{\mathtt{ZH}}, F_{\mathtt{ZH}})$ at which (D.1) exhibits a *fold-Hopf bifurcation,* i.e. has an equilibrium with eigenvalues $\lambda_1 = 0, \lambda_{2,3} = \pm i\omega_0, \omega_0 > 0$. Find numerical parameter values $(G_{\mathtt{CP}}, F_{\mathtt{CP}})$ at which (D.1) exhibits a *cusp bifurcation,* i.e. has an equilibrium with a triple zero.

2. Derive analytical expressions for $(G_{\texttt{ZH}}, F_{\texttt{ZH}})$ and $(G_{\texttt{CP}}, F_{\texttt{CP}})$ as functions of (a, b) and verify the numerical results above.

   Hints: At the fold-Hopf both the trace and the determinant of the Jacobian matrix of (D.1) vanish. To find a triple equilibrium, reduce the equilibrium system for (D.1) to one cubic equation for the x-coordinate of the equilibrium and look for its triple root.

B. Consider the *adaptive control system of Lur'e type*

$$\begin{cases} \dot{x} &= y, \\ \dot{y} &= z, \\ \dot{z} &= -\alpha z - \beta y - x + x^2. \end{cases} \tag{D.2}$$

1. Continue the equilibrium $(x, y, z) = (0, 0, 0)$ of (D.2) with respect to parameter α starting at $\alpha = 2, \beta = 1$ and detect its Hopf bifurcation.
2. Compute the Hopf bifurcation curve in the (α, β)-plane and plot it using the visibility limits

`alpha`	`0`	`1.5`
`beta`	`0`	`2`

3. Find the period-doubling (`PD`) of the cycle bifurcating from the Hopf point and continue it in two parameters.
4. Find the period-doubling of the *doubled* cycle and continue it in the (α, β)-plane.

C. Consider the following prey-predator system from mathematical ecology by Bazykin and Khibnik [1981]:

$$\begin{cases} \dot{x} &= \dfrac{x^2(1-x)}{n+x} - xy, \\ \dot{y} &= -y(m-x), \end{cases} \tag{D.3}$$

where $n > 0$, $0 < m < 1$.

1. Compare its bifurcation diagrams with respect to parameter m for $n_1 = \frac{1}{4}$ and $n_2 = \frac{1}{16}$ using MatCont.
2. Compute the two-parameter bifurcation diagram in the (m, n)-plane and sketch all qualitatively different phase portraits.

   Hint: The diagram should include a Hopf bifurcation curve (`H`) and a cycle fold curve (`LPC`) that connects a *generalized Hopf point* (`GH`) on `H` with the point $(m, n) = (0, 0)$.
3. Derive an analytic expression for the Hopf bifurcation curve in (D.3). *Hint*: Consider the orbitally-equivalent polynomial system

$$\begin{cases} \dot{x} &= x^2(1-x) - xy(n+x), \\ \dot{y} &= -y(m-x)(n+x). \end{cases} \tag{D.4}$$

4. Verify the numerically found GH-point by proving that the first Lyapunov coefficient l_1 vanish at $(m_{\mathtt{GH}}, n_{\mathtt{GH}}) = \left(\frac{1}{4}, \frac{1}{8}\right)$. *Hints*:
   (a) At Hopf parameter values, translate the origin in (D.4) to the equilibrium and scale the variables to obtain a system in the form

$$\begin{cases} \dot{\xi} &= -\omega\eta + P(\xi,\eta), \\ \dot{\eta} &= \omega\xi + Q(\xi,\eta), \end{cases}$$

   where P, Q contain terms of order two and higher in (ξ, η).
   (b) Introduce complex variables $z = \xi + i\eta$ and $\bar{z} = \xi - i\eta$, and derive the equation

$$\dot{z} = i\omega z + \sum_{2\le j+k\le 3} \frac{1}{j!k!} g_{jk} z^j \bar{z}^k + O(|z|^4).$$

   (c) Compute

$$l_1 = \frac{1}{2\omega^2} \operatorname{Re}\left(i g_{20} g_{11} + \omega g_{21}\right).$$

Appendix E

Listing of `Projectie.m`

```
function obj = Projectie(Uman, Sman, itnumber)
tic
    intersecties=[];
    % De punten worden geordend
    [~,I]=sort(Uman(1,:));
    A=Uman(:,I);
    [~,Y]=sort(Sman(1,:));
    B=Sman(:,Y);

    %struct construeren met als velden, het punt, de lijnstukken die
    %eindigen met dit punt, de lijnstukken die beginnen met dit punt
    %en tot welke variëteit het behoort (stable of unstable)
    s=struct('punt', {}, 'eindpunt', {}, 'beginpunt', {},...
        'stab', {}, 'index', {});
    i=1;
    j=1;
    while i <= length(Uman) || j <= length(Sman)

        if i > length(Uman) || A(1,i) > B(1,j)
            punt=B(:,j);
            s(i+j-1).punt=punt;
            col=Y(1,j);
            s(i+j-1).index=col;
            if (col-1) > 0
                lijnstuk(col, col-1, Sman, i+j-1);
            end
            if (col+1) <= length(Sman)
                lijnstuk(col, col+1, Sman, i+j-1);
            end
            s(i+j-1).stab='stable';
```

```
            j=j+1;
        elseif j > length(Sman) || A(1,i) <= B(1,j)
            punt=A(:,i);
            s(i+j-1).punt=punt;
            col=I(1,i);
            if (col-1) > 0
                lijnstuk(col, col-1, Uman, i+j-1);
            end
            if (col+1) <= length(Uman)
                lijnstuk(col, col+1, Uman, i+j-1);
            end
            s(i+j-1).stab='unstable';
            i=i+1;
        end
    end


    function lijnstuk(a, b, man, index)
        if man(1,a) >= man(1,b)
            lijn=[man(:,b); man(:,a)];
            if isempty(s(index).eindpunt)
                s(index).eindpunt=lijn;
            else
                s(index).eindpunt=[s(index).eindpunt, lijn];
            end
        else
            lijn=[man(:,a); man(:,b)];
            if isempty(s(index).beginpunt)
                s(index).beginpunt=lijn;
            else
                s(index).beginpunt=[s(index).beginpunt, lijn];
            end
        end
    end


    % koppels van lijnstukken met overlappende x intervallen
    c=struct('lijnstuk', {}, 'index', {});
    k=1;
    m=1;
    x={};
    E=[];
    e=1;
```

```
% for-lus over alle punten
for i=1: length(s)
    [~,n]=size(s(i).beginpunt);
    % for-lus over lijnstukken die punt als begin hebben (max 4
    % lijnstukken)
    for j=1: n
        c(k).lijnstuk=s(i).beginpunt(:,j);
        c(k).index=s(i);
        % for-lus over lijnstukken die overlappen
        % met interval (max n)
        for l=1:k-1
            if ~strcmp(c(l).index.stab, c(k).index.stab)
                x{m}=c(l).lijnstuk;
                x{m+1}=c(k).lijnstuk;
                m=m+2;
                if strcmp(c(l).index.stab, 'stable')
                    E(e)=c(l).index.index;
                else
                    E(e)=c(k).index.index;
                end
                e=e+1;
            end
        end
        k=k+1;
    end
    % for-lus over lijnstukken die punt als eind hebben (max 4
    % lijnstukken)
    [~,n]=size(s(i).eindpunt);
    for j=1:n
        % for-lus over lijnstukken die overlappen
        % met interval (max n)
        l=1;
        while ~(isequal(c(l).lijnstuk, s(i).eindpunt(:,j)))
            l=l+1;
        end
        c(l)=[];
        k=k-1;
    end
end

e=0;
% koppels die ook in y-intervallen overlappen
for i=1:2:length(x)
```

```
    max1=max(x{1,i}(4,1),x{1,i}(2,1));
    max2=max(x{1,i+1}(4,1),x{1,i+1}(2,1));
    min1=min(x{1,i}(4,1),x{1,i}(2,1));
    min2=min(x{1,i+1}(4,1),x{1,i+1}(2,1));
    e=e+1;
    if max2 <= max1 && ~(max2 < min1)
        punt=intersectie(x{1,i}, x{1,i+1});
        if ~isempty(punt)
            dummy=[punt;E(e)];
            intersecties=[dummy, intersecties];
        end
    elseif min2 >= min1 && ~(min2 > max1)
        punt=intersectie(x{1,i}, x{1,i+1});
        if ~isempty(punt)
            dummy=[punt;E(e)];
            intersecties=[dummy, intersecties];
        end
    elseif min2 <= min1 && max2 >= max1
        punt=intersectie(x{1,i}, x{1,i+1});
        if ~isempty(punt)
            dummy=[punt;E(e)];
            intersecties=[dummy, intersecties];
        end
    end


end
% Intersecties worden geordend volgens stabiele variëteit
[~,W]=sort(intersecties(4,:), 'descend');
intersecties=intersecties(:,W);
for i=1:length(intersecties)
    j=1;
    while (i+j) <= length(intersecties) && ...
                intersecties(4,i)==intersecties(4,i+j)
       j=j+1;
    end
    if j > 1
    [~,Z]=sort(intersecties(3,i:i+j-1));
    B=intersecties(:,i:i+j-1);
    intersecties(:, i:i+j-1)=B(:, Z);
    end
end
obj=banen(intersecties(1:2,:), itnumber);
```

```
    toc
end

function [punt] = intersectie(a, b)
    punt = [];
    x1=b(1,1);
    x2=b(3,1);
    y1=b(2,1);
    y2=b(4,1);
    int=[];

    fold=(x2-x1)*(a(2,1)-y2)-(y2-y1)*(a(1,1)-x2);
    fnew=(x2-x1)*(a(4,1)-y2)-(y2-y1)*(a(3,1)-x2);
    if (fnew*fold)<0
        system=[1, 0,                  x2-x1; ...
                0, 1,                      y2-y1; ...
                a(4,1)-a(2,1),    a(1,1)-a(3,1), 0 ];
        RHS=[x2;y2;a(4,1)*a(1,1)-a(3,1)*a(2,1)];
        int=system\RHS;
    end
    if ~isempty(int) && int(3)>0 && int(3)<=1
        punt= int;
    end

end
```

Appendix F

Listing of `Projectie2.m`

```
function obj = Projectie2(Uman, Sman, itnumber)
tic
    intersecties=[];
    % De punten worden geordend
    [~,I]=sort(Uman(1,:));
    A=Uman(:,I);
    [~,Y]=sort(Sman(1,:));
    B=Sman(:,Y);

    %struct construeren met als velden, het punt, de lijnstukken die
    %eindigen met dit punt, de lijnstukken die beginnen met dit punt
    % en tot  welke variëteit het behoort (stable of unstable)
    s=struct('punt', {}, 'eindpunt', {}, 'beginpunt', {},...
                'stab', {}, 'index', {});
    i=1;
    j=1;
    while i <= length(Uman) || j <= length(Sman)

        if i > length(Uman) || A(1,i) > B(1,j)
            punt=B(:,j);
            s(i+j-1).punt=punt;
            col=Y(1,j);
            if (col-1) > 0
                lijnstuk1(col-1, col, Sman, i+j-1);
            end
            if (col+1) <= length(Sman)
                lijnstuk(col, col+1, Sman, i+j-1);
            end
            s(i+j-1).stab='stable';
            s(i+j-1).index=col;
```

```
            j=j+1;
        elseif j > length(Sman) || A(1,i) <= B(1,j)
            punt=A(:,i);
            s(i+j-1).punt=punt;
            col=I(1,i);
            if (col-1) > 0
                lijnstuk1(col-1, col, Uman, i+j-1);
            end
            if (col+1) <= length(Uman)
                lijnstuk(col, col+1, Uman, i+j-1);
            end
            s(i+j-1).stab='unstable';
            i=i+1;
        end

    end


    function lijnstuk(a, b, man, index)
        if man(1,a) >= man(1,b)
            lijn=[man(:,b); man(:,a)];
            if isempty(s(index).eindpunt)
                s(index).eindpunt=lijn;
            else
                s(index).eindpunt=[s(index).eindpunt, lijn];
            end
        else
            lijn=[man(:,a); man(:,b)];
            if isempty(s(index).beginpunt)
                s(index).beginpunt=lijn;
            else
                s(index).beginpunt=[s(index).beginpunt, lijn];
            end
        end
    end

 function lijnstuk1(a, b, man, index)
        if man(1,a) >= man(1,b)
            lijn=[man(:,b); man(:,a)];
            if isempty(s(index).beginpunt)
                s(index).beginpunt=lijn;
            else
                s(index).beginpunt=[s(index).beginpunt, lijn];
```

```
        end
    else
        lijn=[man(:,a); man(:,b)];
        if isempty(s(index).eindpunt)
            s(index).eindpunt=lijn;
        else
            s(index).eindpunt=[s(index).eindpunt, lijn];
        end
    end
end

c=struct('lijnstuk', {}, 'index', {});
z=struct('lijnstuk', {}, 'index', {});
k=1;
w=1;
m=1;
x={};
E=[];
e=1;
% for-lus over alle punten
for i=1: length(s)
    [~,n]=size(s(i).beginpunt);
    for j=1: n
        % De lijnstukken van het punt worden toegevoegd aan c of z
        % naargelang het punt stabiel of onstabiel is
        if strcmp(s(i).stab, 'stable')
            c(k).lijnstuk=s(i).beginpunt(:,j);
            c(k).index=s(i).index;
            % Het lijnstuk wordt met alle lijnstukken van de andere
            % lijst opgeslagen als paar, aangezien ze overlap hebben
            % in de x-richting
            for l=1:w-1
                x{m}=z(l).lijnstuk;
                x{m+1}=c(k).lijnstuk;
                E(e)=c(k).index;
                e=e+1;
                m=m+2;
            end
            k=k+1;
        elseif strcmp(s(i).stab, 'unstable')
            z(w).lijnstuk=s(i).beginpunt(:,j);
            for l=1:k-1
                x{m+1}=c(l).lijnstuk;
```

```
                x{m}=z(w).lijnstuk;
                E(e)=c(l).index;
                e=e+1;
                m=m+2;
            end
            w=w+1;
        end
    end
    % Lijnstukken met punt als eindpunt worden verwijderd uit lijst
    [~,n]=size(s(i).eindpunt);
    for j=1:n
        if strcmp(s(i).stab, 'stable')
            l=1;
            while ~(isequal(c(l).lijnstuk, s(i).eindpunt(:,j)))
                l=l+1;
            end
            c(l)=[];
            k=k-1;
        elseif strcmp(s(i).stab, 'unstable')
            l=1;
            while ~(isequal(z(l).lijnstuk, s(i).eindpunt(:,j)))
                l=l+1;
            end
            z(l)=[];
            w=w-1;
        end
    end
end
% koppels die ook in y-intervallen overlappen
e=0;
for i=1:2:length(x)
    max1=max(x{1,i}(4,1),x{1,i}(2,1));
    max2=max(x{1,i+1}(4,1),x{1,i+1}(2,1));
    min1=min(x{1,i}(4,1),x{1,i}(2,1));
    min2=min(x{1,i+1}(4,1),x{1,i+1}(2,1));
    e=e+1;
    if max2 <= max1 && ~(max2 < min1)
        punt=intersectie(x{1,i}, x{1,i+1});
        if ~isempty(punt)
            dummy=[punt;E(e)];
            intersecties=[dummy, intersecties];
        end
    elseif min2 >= min1 && ~(min2 > max1)
```

```
            punt=intersectie(x{1,i}, x{1,i+1});
            if ~isempty(punt)
                dummy=[punt;E(e)];
                intersecties=[dummy, intersecties];
            end
        elseif min2 <= min1 && max2 >= max1
            punt=intersectie(x{1,i}, x{1,i+1});
            if ~isempty(punt)
                dummy=[punt;E(e)];
                intersecties=[dummy, intersecties];
            end
        end


    end
    % Intersecties worden geordend volgens stabiele variëteit
    [~,W]=sort(intersecties(4,:), 'descend');
    intersecties=intersecties(:,W);
    for i=1:length(intersecties)
        j=1;
        while (i+j) <= length(intersecties) && ...
                    intersecties(4,i)==intersecties(4,i+j)
           j=j+1;
        end
        if j > 1
        [~,Z]=sort(intersecties(3,i:i+j-1));
        B=intersecties(:,i:i+j-1);
        intersecties(:, i:i+j-1)=B(:, Z);
        end
    end
    obj=banen(intersecties(1:2,:), itnumber);
    toc
end

function [punt] = intersectie(a, b)
    punt = [];
    x1=b(1,1);
    x2=b(3,1);
    y1=b(2,1);
    y2=b(4,1);
    int=[];

    fold=(x2-x1)*(a(2,1)-y2)-(y2-y1)*(a(1,1)-x2);
```

```
    fnew=(x2-x1)*(a(4,1)-y2)-(y2-y1)*(a(3,1)-x2);
    if (fnew*fold)<0
        system=[1,0,                  x2-x1; ...
                0,1,                  y2-y1; ...
                a(4,1)-a(2,1),    a(1,1)-a(3,1), 0];
        RHS=[x2;y2;a(4,1)*a(1,1)-a(3,1)*a(2,1)];
        int=system\RHS;
    end
    if ~isempty(int) && int(3)>0 && int(3)<=1
        punt= int;
    end

end
```

Appendix G

Listing of banen.m

```
function [HomCell]=banen(intersections, itnumber)

global man_ds
homCurves=zeros(1+size(intersections,1),size(intersections,2));
homCurves(2:end,:)=intersections;
%%
index=1;
findedind=1;
while ~isempty(findedind)
    homCurves(1,findedind)=index;
    j=findedind;
    f_p=homCurves(2:end,j);
    for k=1:itnumber
    f_p=feval(man_ds.func,0,f_p,man_ds.P0{:});
    end
    for i=j+1:size(homCurves,2)
        if norm(homCurves(2:end,i)-f_p)<1e-3
            homCurves(1,i)=index;
            f_p=homCurves(2:end,i);
            for k=1:itnumber
            f_p=feval(man_ds.func,0,f_p,man_ds.P0{:});
            end
        end
    end
    index=index+1;
    findedind=find(homCurves(1,:)==0,1);
end
assignin('base','homCurves',homCurves);

index=index-1;
```

```
HomCell=cell(index,1);
for i=1:index
    indici=homCurves(1,:)==i;
    HomCell{i}=homCurves(2:end,indici);
end

end
```

Appendix H

List of settings

The settings in de Settings class are configured by the computations. To give an overview of all settings that can be made in the current version of the MatCont GUI, we provide a listing of a settings object that has been configured by all computations and therefore contains all possible settings. The system `adapt2` was selected as the current system and has as coordinates `x`, `y` and `z`. The parameters are `alpha` and `beta`.

```
              system: adapt2
       userfunctions: <ufdata>
                  IP: Equilibrium (EP)

             forward: true

        option_pause: At Special Points
      option_archive: 2
       option_output: 1
option_tsearchorder: true
option_moorepenrose: true
   option_increment: 1e-05
```

A settings contains a 'initial point (IP)'. This point has been set to an Equilibrium. This setting can also store additional data.

E.g. whenever a point is selected from a continuation curve, the relevant data from the `s`, `x` and `v` structures are also stored in that setting. This setting is required for some initializers. The setting `forward` is set whenever a computation is activated.

These are the settings related to the continuation computation.

```
       InitStepsize: 0.01
        MinStepsize: 1e-05
        MaxStepsize: 0.1

     MaxNewtonIters: 3
```

```
MaxCorrIters: 10
MaxTestIters: 10
VarTolerance: 1e-06
FunTolerance: 1e-06
TestTolerance: 1e-05
Adapt: 3

MaxNumPoints: 300
CheckClosed: 50
```

Coordinates are added to the settings with prefix `co_` and parameters are added with prefix `pa_`. To select a parameter as active parameter, you address the setting with suffix `_select`. The settings with suffix `_branch` are used to select branch parameters. The settings `coord` and `parameters` can be used to set all coordinates or all the parameters at once respectively. The userfunctions are added with prefix `uf_` combined with the label.

```
time: 0
co_x: 0
co_y: 0
co_z: 0
coord: [ 0, 0, 0 ]

parameters: [ 0, 0 ]
pa_alpha: 0
pa_beta: 0


pa_alpha_select: false
pa_beta_select: false
pa_alpha_branch: false
pa_beta_branch: false
Period: false

uf_U1: false
uf_U2: false
uf_U2: false


ntst: 40
ncol: 4

amplitude: 1e-06
eps: 1e-06
```

```
 bt_amplitude: 0.0001
      whichNS: 1
bt_ttolerance: 1e-05

  multipliers: true

  eigenvalues: true

     prcInput: 1
   PRCenabled: false
  dPRCenabled: false
```

Various initializers need some specialized settings. The most important of them are the discretization settings (`ntst` and `ncol`) and the options to compute eigenvalues or multipliers.

The names of all options for selecting to detect bifurcations consist of the prefix `test_`, followed by the label of the curve and the label of the bifurcation.

```
   test_EP_BP: true
   test_LP_BT: true
    test_H_BT: true
  test_LC_BPC: true
  test_LPC_R1: true
   test_NS_R1: true
   test_PD_R2: true
  test_Hom_NS: true
    test_EP_H: true
   test_LP_ZH: true
    test_H_ZH: true
   test_LC_PD: true
 test_LPC_CPC: true
   test_NS_R2: true
 test_PD_LPPD: true
 test_Hom_DRS: true
   test_EP_LP: true
   test_LP_CP: true
    test_H_HH: false
  test_LC_LPC: true
test_LPC_LPNS: true
   test_NS_R3: true
  test_PD_GPD: true
 test_Hom_DRU: true
    test_H_GH: true
```

```
        test_LC_NS: true
     test_LPC_LPPD: true
        test_NS_R4: true
      test_PD_PDNS: true
      test_Hom_NDS: true
      test_NS_LPNS: true
      test_Hom_NDU: true
        test_NS_CH: true
test_Hom_test_3LS: true
      test_NS_PDNS: true
test_Hom_test_3LU: true
      test_NS_NSNS: false
       test_Hom_SH: true
      test_Hom_NCH: true
       test_Hom_BT: true
      test_Hom_OFS: true
      test_Hom_OFU: true
      test_Hom_IFS: true
      test_Hom_IFU: true
```

The following settings are used for simulation (time integration):

```
          Interval: 1
     eventfunction: <disabled>
 InitStepSize_sim: <automatic>
  MaxStepSize_sim: <automatic>
     RelTolerance: 0.001
     AbsTolerance: 1e-06
           Refine: 1
      Normcontrol: false
              BDF: false
         MaxOrder: 5
```

Appendix I

List of computations

The class file `CompConfigurations.m` constructs a list of all possible computations. A computation is also referred to as a *Computation Configuration* (internally: *CompConf*). Each computation represents either a simulation or an initializer plus continuation. Other types of computations can be added in this file in future releases. E.g. the computation of the Lyapunov exponents could be added as a new computation like as it is done in the maps version of MatCont.

This part of the code registers all simulations into the GUI:

```
obj.OrbitConfs{end+1} = SimConf('ode45', 4, true, 0);  %default refine: 4
obj.OrbitConfs{end+1} = SimConf('ode23', 1,true, 1); %default refine: 1
obj.OrbitConfs{end+1} = SimConf('ode113', 1,true, 2); %default refine: 1
obj.OrbitConfs{end+1} = SimConf_ode15s(1,true, 3); %default refine1
obj.OrbitConfs{end+1} = SimConf('ode23s', 1,true,  4); %default refine: 1
obj.OrbitConfs{end+1} = SimConf('ode23t', 1,true, 5); %default refine: 1
obj.OrbitConfs{end+1} = SimConf('ode23tb', 1,true, 6); %default refine: 1
obj.OrbitConfs{end+1} = SimConf('ode78', 1,false, 7); %default refine: 1
obj.OrbitConfs{end+1} = SimConf('ode87', 1,false, 8); %default refine: 1
```

For each available simulation, a 'Connect' simulation computation is added which is used in the Homotopy methods. This computation performs a simulation but adds additional instructions for locating a suitable point on the simulation near a saddle.

```
for k = 1:length(obj.OrbitConfs)
   obj.OrbitConfs{end+1} = SimConf_Connect('ConnectionSaddle', ...
                                                obj.OrbitConfs{k});
end
```

This part of the code registers all the initializers in the gui. Each computation performs the initializer and the continuation. Some of the computations are specifically constructed for a certain initializer. Other computations are more generic and can work with an initializer passed along as argument during construction. Note that in the names of the

`ContConf` classes the curve label is placed first and then the point label placed second. The addition of a pointlabel indicates that this computation does something specialized for that pointlabel, e.g. adding an extra setting.

```
conflist{end+1} = ContConf_EP();
conflist{end+1} = ContConf_EP_BP();
conflist{end+1} = ContConf_EP_point('H', @init_H_EP);
conflist{end+1} = ContConf_EP_point('LP', @init_LP_EP);

%neutral saddle equilibrium
conflist{end+1} = ContConf_EP_point('NE', @init_EP_EP);


conflist{end+1} = ContConf_LP_point('BP', @init_BP_LP);
conflist{end+1} = ContConf_LP_point('CP', @init_CP_LP);
conflist{end+1} = ContConf_LP_point('ZH', @init_ZH_LP);
conflist{end+1} = ContConf_LP_point('BT', @init_BT_LP);
conflist{end+1} = ContConf_LP_LP();

conflist{end+1} = ContConf_H('H', @init_H_H);
conflist{end+1} = ContConf_H('GH', @init_GH_H);
conflist{end+1} = ContConf_H('HH', @init_HH_H);
conflist{end+1} = ContConf_H('ZH', @init_ZH_H);
conflist{end+1} = ContConf_H('BT', @init_BT_H);
conflist{end+1} = ContConf_H('NE', @init_H_H); %Neutral Saddle Equilibrium

conflist{end+1} = ContConf_LC_H();
conflist{end+1} = ContConf_LC_PD();
conflist{end+1} = ContConf_LC_BPC();
conflist{end+1} = ContConf_LC_LC(); %LC, BPC and NC (neutral saddle Eq.)

conflist{end+1} = ContConf_LPC('LPC' , @init_LPC_LPC);
conflist{end+1} = ContConf_LPC('BPC' , @init_BPC_LPC);
conflist{end+1} = ContConf_LPC('CPC' , @init_CPC_LPC);
conflist{end+1} = ContConf_LPC('LPNS', @init_LPNS_LPC);
conflist{end+1} = ContConf_LPC('LPPD', @init_LPPD_LPC);
conflist{end+1} = ContConf_LPC('R1'  , @init_R1_LPC);
conflist{end+1} = ContConf_LPC('GPD' , @init_GPD_LPC);
conflist{end+1} = ContConf_LPC_GH();

conflist{end+1} = ContConf_NS('NS' , @init_NS_NS);
conflist{end+1} = ContConf_NS('NC' , @init_NS_NS); %Neutral Saddle Cycle
conflist{end+1} = ContConf_NS('LPNS' , @init_LPNS_NS);
```

```
conflist{end+1} = ContConf_NS('PDNS' , @init_PDNS_NS);
conflist{end+1} = ContConf_NS('CH' , @init_CH_NS);
conflist{end+1} = ContConf_NS('R1' , @init_R1_NS);
conflist{end+1} = ContConf_NS('R2' , @init_R2_NS);
conflist{end+1} = ContConf_NS('R3' , @init_R3_NS);
conflist{end+1} = ContConf_NS('R4' , @init_R4_NS);
conflist{end+1} = ContConf_NS_HH();
conflist{end+1} = ContConf_NS_ZH();

conflist{end+1} = ContConf_PD('GPD', @init_GPD_PD);
conflist{end+1} = ContConf_PD('LPPD', @init_LPPD_PD);
conflist{end+1} = ContConf_PD('PDNS', @init_PDNS_PD);
conflist{end+1} = ContConf_PD('R2', @init_R2_PD);
conflist{end+1} = ContConf_PD('PD', @init_PD_PD);

conflist{end+1} = ContConf_BP();
conflist{end+1} = ContConf_BPC();

conflist{end+1} = ContConf_Hom();
conflist{end+1} = ContConf_Hom_LC();
conflist{end+1} = ContConf_Hom_BT();
conflist{end+1} = ContConf_Hom_NCH();
conflist{end+1} = ContConf_Hom_HTHom();
```