HABILITATION

The AnT project: On the simulation and analysis of dynamical systems

Presented to the faculty of
Computer Science, Electrical Engineering,
and Information Technology

at the University of Stuttgart

by

Dr. rer. nat. Michael Schanz

Department Image Understanding

Institute of Parallel and Distributed Systems

University of Stuttgart

In Memoriam

Erika Sonja Schanz	24. März 1928 - 24. April 1993
Paul Karl Schanz	13. Januar 1926 - 24. Mai 2001

Preface

A software project like the one presented in this work could never have been done without the help and support of many, many people. Among them are people who are more or less directly involved in the project as well as others which supported me in many different ways. The people I want to thank here are in fact so many, that it is not possible to thank them all personally. Therefore, I would like to thank in general the members of the Institute of Parallel and Distributed Systems (IPVS) at the University of Stuttgart, where the AnT project resides and where most of the work was done. However, some persons I want to thank personally:

- The referees of this work, Prof. Bestehorn (Cottbus), Prof. Bungartz (Stuttgart), Prof. Friedrich (Münster) and Prof. Levi (Stuttgart). Moreover, I would like to thank Prof. Levi, head of the department Image Understanding at the IPVS, for his friendly support, for the many discussions and for giving me the opportunity and time to develop and maintain the AnT project and to write this habilitation.

- All the people involved in the AnT project, but especially my friends:
  - Viktor Avrutin, for his brilliant ideas, profound knowledge and never ending interest in nonlinear dynamics, the uncountable valuable discussions with him and finally his affectation for ORIGAMI.
  - Robert Lammert, for his creativity, his art of software engineering and his effort not only during the development of the graphical user interface but in many other parts and modules of the software.
  - Georg Wackenhut, for his support, his holistic thinking, his endurance with the build mechanism especially in critical phases of the project and finally his careful preparations of the distributions.

Additionally I would like to thank:

- Our secretary Ute Gräter for her friendship and her tremendous help in nearly every case and under nearly any circumstances.
- The complete RoboCup team at our institute, but especially Georg Kindermann, Reinhard Lafrenz and Frank Schreiber for the many interesting discussions during our many famous breakfasts.
- My long standing friend Martin Ossig who supervised together with me some diploma theses.
- My friends at basketball who helped me ease the built-up tension in my shoulders and neck caused by the many hours sitting in front of the computer. Here I would like to add a special thanks to Uwe Husmann, who worked with the software and found some bugs and Elmar Groß who is directly involved in the new web front end of the project.
- Last but not least, my friends and former colleagues from the physics department and especially the organizer Robert Hönlinger for the long lasting and always interesting events called "Kaffeerunde".

Furthermore, a special thanks to my sister Martina Frey and my brothers Joachim and Jürgen Schanz and their families for their understanding, their personal support and the many funny parties and other family events.

Finally I want to thank my wife Cordula Ernst for her irresistible love, her amazing spirit, her great humor, her endless care and support and her courage for marrying me.

Stuttgart, July 2004 Michael Schanz

Glossary

$\underline{s}$:	state vector	44
Γ_s:	state space	44
N_s:	state space dimension	44
s_i:	i-th state component or state variable	44
$\underline{p}$:	parameter vector	44
Γ_p:	parameter space	44
N_p:	parameter space dimension	44
p_i:	i-th parameter	44
$\underline{s}(t_n)$:	state vector at time t_n	48
t_n:	discrete time at timestamp n	48
$\underline{f}$:	vector field	48
$\underline{s}_n$:	state vector at timestamp n	49
n:	timestamp n	49
x_i^*:	fixed points	50
α_i:	i-th flip bifurcation point	51
α_∞:	accumulation point	51
λ:	Lyapunov exponent	53

$\underline{q}^{fi}$:	final 2D-scan point	100
$\Delta\underline{q}$:	2D-scan point increment	100
${q_i}^k$:	i-th component of the k-th 2D-scan point	100
${q_i}^{off}$:	offset of the i-th component of the 2D-scan point	100
a_i:	parameter of the scan ellipse	100
trig_1:	cos	100
trig_2:	sin	100
φ^k:	angle of the k-th 2D-scan point	100
$\Delta\varphi$:	2D-scan point angle increment	100
φ^{in}:	angle of the initial 2D-scan point	100
φ^{fi}:	angle of the final 2D-scan point	100
${q_i}^{in}$:	i-th component of the initial 2D-scan point	100
${q_i}^{fi}$:	i-th component of the final 2D-scan point	100
$\mathcal{A}$:	attractor	111
$\mathcal{A}$:	wave number vector	111
(s, m, e):	floating point number representation	224
b:	bias	224
e:	exponent	224
e_i:	exponent bit	224
m:	mantissa	224
m_i:	mantissa bit	224
s:	sign bit	224
$\underline{\underline{M}}$:	matrix	227

$p(\lambda)$:	characteristic polynomial	227
λ_i^a:	analytically calculated i-th eigenvalue	227
λ_i^n:	numerically calculated i-th eigenvalue	227
$\underline{v}_i$:	eigenvector i	228
m:	particle mass	280
f :	friction constant	280
k :	spring constant	280

Summary

In this work, a the software project **AnT 4.669** is presented. This project is about a simulation and analysis tool for dynamical systems, whereby it was aimed right from the start, that not only a broad spectrum of dynamical systems is supported, but many investigation methods as well. This thesis is structured as follows:

In the **Introduction** (Chap. 1) a motivation and an overview about the main topics and the involved scientific disciplines is given and among others, the notions of simulation and analysis and their prerequisites are defined and described. The modeling and especially the mathematical modeling are considered in more detail as well as the tasks of interactive and non-interactive simulation. **The AnT project** chapter provides an overview about the software - its aims, requirements and features. Some historical and technical remarks are given together with some basic principles and facts about the functionality and a short guidance how to use this software and its features. In the third chapter, denoted as **Supported classes of dynamical systems**, the notion of a dynamical system is defined and described in this context and the most important classes of dynamical systems supported by the **AnT 4.669** software package including their main characteristic properties are listed. For each addresses class of dynamical systems at least one typical and illustrative representative is presented. In the fourth chapter - **Scanning of dynamical systems** - the notion of scanning is described and its importance in the field of nonlinear dynamics is emphasized. The different types of scan possibilities and procedures provided by the software are explained as well as the meaning of a scan item and a scan item sequence. Directly connected with the scanning of a dynamical system is the investigation or analysis of the dynamic behavior, because in most of the cases one is not only interested in the investigation under some fixed conditions

but under varying conditions. For the analysis of dynamical systems, several methods are developed in the meanwhile. The methods, which are already implemented in the **AnT 4.669** software and hence available, are presented in chapter five - **Supported investigation methods**. The investigation of dynamical systems is the most important task in the field of nonlinear dynamics, because it represents the basis for prediction and forecasting of the dynamics as well as improvement and enhancement of the used or derived mathematical models. Therefore, a lot of examples are given in this chapter where the supported investigation methods are applied and illustrated. Chapter six is about **Simulating and investigating dynamical systems**. Here, a more detailed description of the software architecture and applied concepts is given. Especially the most important concepts of transitions and machines designed for the simulation and the investigation are explained and illustrated. The necessity of **Distributed computing** is addressed to in chapter seven, where the capability of the software to run in distributed mode to solve time consuming investigation tasks in parallel on several workstations or nodes of a cluster is described. The used client/server architecture is presented together with the developed network protocol. Many diagrams and figures in this work are based on the result of time consuming computations and could not have been prepared without the important distributed computing feature of the **AnT 4.669** software package. In the following two chapters 8 and 9, the graphical user interface, which guides a user through the initialization phase and the visualization capabilities of the software were presented, before in chapter ten **Numerical aspects of simulation** some remarks about numerics and scientific computing are elucidated. The work closes with some additional **Examples** (Chap. 11) and a short **Conclusion** (Chap. 12) about the main advantages of the **AnT 4.669** software package and the future extensions, including an overview about related software projects.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the last decade ***computer-based simulation*** and ***computer-based analysis*** became more and more important in computer science and many other research fields. Not only in natural sciences, technical or engineering disciplines but also in social sciences and other disciplines like advertisement, entertainment and arts. In many industrial applications like for instance research and development, visualization or prototyping, computer-based simulation and the ***simulation-based investigation*** of systems become more and more a crucial point in the competition process between companies. Although there exist many impact factors like for instance the definition of soft- and hardware standards, the availability of specific soft- and hardware or human resources educated in the field of computer science, there are mainly two reasons for that:

1. Due to the high performance of modern computer systems simulation approaches reality in such a way that many expensive and often time consuming experiments and tests can be omitted. Hence, simulation helps in this case to save money and time. A well-known example is the simulation-based investigation of laminar and turbulent flows around any sort of conveyors to supersede expensive experiments in wind channels.

2. In many technical disciplines where experiments are rare, difficult or even impossible to realize, one has to use computer simulations in

oder to get deeper insights in mechanisms and operational principles. Here, the simulation not only makes it possible to investigate such processes in more detail but also supports the repeated execution of corresponding experiments. One interesting example is the simulation of an asteroid impact on earth. The following citation from [24] illustrates the situation (see also [23], [22]):

"A lot of major breakthroughs in science are going to come from these kinds of calculations. Impact simulations are something that can't be done any other way. It's almost like doing an experiment - one you could never do. One you would never want to do."

Examples here are the prototyping of complex products like cars, airplanes or computers, the test of models in astrophysics or the further development of spaceflight. One can also think about the static and dynamic load analysis of buildings in the prior to their construction. Furthermore, in many educational fields and especially in human medicine, where human beings are involved and hence experiments are not possible, computer-based simulation supports and improves learning processes and the acquirement of skills and techniques.

Remark:

1. If the computer and simulation technology were as much developed as nowadays, the famous collapse of the Tacoma Narrows bridge in 1940-11-07 could have been avoided by doing simulations regarding the excitation of resonant oscillations.

2. The collapse of the two towers of the World Trade Center in 2001-09-11, probably could have been also avoided by doing corresponding simulations regarding the thermal properties of the hooks used to attach the individual floors to the supporting steel structure. Although, one has to mention here, that probably nobody expected such unbelievable things to happen.

Thus computer-based experiments or simulations are necessary and become further important. With the increase of computer performance more and more processes or systems get into the focus of simulation in order to save time, money or even lives if one thinks about the numerous experiments with animals which can be omitted in many cases. One famous example of this trend is the decision of the French government in 1996 [16] concerning the abdication of further nuclear weapon tests in favor of computer experiment or simulations[1]. Also the government of the USA has launched a national program to simulate nuclear weapon tests, in order to reduce real tests (see the annual reports 2001 and 2002 of the Lawrence Livermore National Laboratory [2, 3]).

Of course there are so many examples of computer-based simulations and analysis in various fields of research that it is beyond the scope of this work to list them all. However, in the following table 1.1 some illustrative examples and interesting applications are presented:

Artificial life	Design of artificial environments and species or life forms to simulate and explore the basic principles and rules of early life systems ant the emergence of life
Astronomy	Investigation of the life-cycle of stars from their emergence out of the cosmic dust up to their end as white dwarfs, neutron stars or black holes
Automobile research	Here not only the development and design of new prototypes of cars or trucks is supported by computer simulation techniques, but also the testing. Furthermore the dynamic behavior of automobiles can be simulated under various conditions using complex mathematical models. Using this methods of simulation in a virtual reality, one is able to detect faults or errors in design and function of the automobile in an eraly stage of the development process

[1]France's *Préparation a la Limitation des Experimentations Nucléaires* (PALEN) program was originally designed to reduce the number of nuclear weapon tests conducted. It is now intended to develop the means and techniques necessary to maintain the credibility of France's nuclear deterrent in a post-test environment, and as such the program has renamed *Programme de Simulation des Essais Nucléaires* (PaSEN)

Bio informatics	The development of new and more efficient medicaments, drugs and other active substances require simulations for instance to investigate the complicated folding process of bio-molecules or proteins
Climate research	Weather forecasting and investigation of the long-term behavior of the earth's climate
Cosmology	Investigation of the emergence of cosmic structures like galaxies, cosmic strings or the great attractor
Dynamical Systems	Most dynamic processes observable in natural or technical systems are usually nonlinear. As a consequence, their analytic treatment is often difficult or not possible. To analyze the dynamic properties of these systems anyway, for instance to adjust parameters or to determine characteristic dynamic behaviors, one has to simulate and investigate them numerically
Material research	Investigation of behavior and characterisitic properties of new developed or designed materials prior to their application
Nano technology	In chip design in the near future physical limitations are reached which make it impossible to work with averaged physical quantities and hence simulations of the underlying microscopic quantum physical processes become necessary
Nuclear research	Promotion of the development of nuclear fusion reactors and improvment of nuclear fission reactors to ensure the increasing power consumption of mankind
Space technology	In space technology, astronautics and aerospace research in eneral and particulary in the crewded spaceflight, simulations are mandatory to avoid or at least to reduce dangerous hazards and accidents with a high risk of loosing human lives. For instance the training of docking maneuvers or the recovering or maintenance and repair of satellites can be supported by simulations. Of course also the enormous costs of space flights can be reduced

Visualization	This specific form of simulation becomes more and more important not only to get rid of the time consuming and expensive fabrication of prototypes, but also in many other cases, for instance in the field of 3D-planning of manufacturing or power plants

Table 1.1: Some illustrative application fields of computer-based simulation

So far, it was pointed out that the importance of computer-based simulations grows rapidly. However, in most cases the simulation of a ***real world system*** can't be considered as an isolated topic. Instead it has to be regarded in a larger context, where the real simulation is only one phase - not necessarily the main phase - of a process, which is denoted throughout this work as ***simulation process***.

Definition: ***Simulation process***

A simulation process consists of three interconnected phases, namely the modeling phase, the simulation phase and the analysis phase, which are usually traversed sequentially and iteratively. The modeling phase is concerned with developing and improvement of adequate models of the system under consideration. Obviously, in the simulation phase of the process this model is simulated, whereby specific output is produced. In the analysis phase, the behavior of the system is investigated and validated using specific observable quantities. ■

Hereby the following definition of a system is used:

Definition: ***System***

A system is an entity with some characteristic time dependent properties, in which one is interested in. ■

This relatively simple definition of a system is used in this work, to be able to cope with very different classes of systems like for instance the solar system representing a large scale natural system and an electronic oscillator, representing a small scale technical system.
In Fig. 1.1 the simulation process is shown schematically. As one can see the interconnections lead on the one hand to the sequential traverse of the three

phases. On the other hand, there is a direct feedback from each phase to it predecessor. This shortcuts are necessary to eliminate obvious or simple failures in the predecessor phase in an early stage, resulting in an overall performance. However there is a more subtle and sophisticated feedback by the iteration loop which interconnects the outcome from the analysis phase with the modeling phase, leading thus step by step to an improvement of the model and hence the scientific results and statements corresponding to the real system.

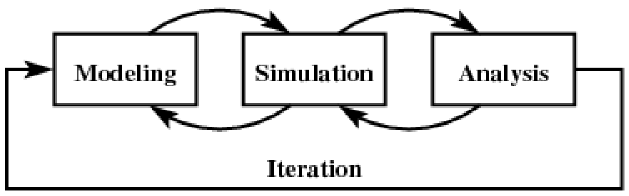


Figure 1.1: The simulation process

When dealing with technical or more generally speaking man-made systems, the simulation process can be enhanced, taking additionally into account the invention and construction or build-up of the system. In this case, the ***enhanced simulation process*** (see Fig. 1.2) has a fourth phase, namely the ***design process***. Because it is again beyond the scope of this work, the design process is not considered. Instead, this man-made systems are considered and treated like natural systems and hence only the pure simulation process is focused on.

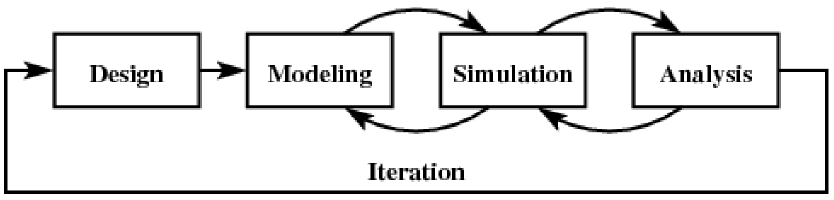


Figure 1.2: The enhanced simulation process

1.2 The simulation process

1.2.1 Modeling

Modeling is a very important task not only when dealing with computer-based simulations. Nearly in all domains of science and also in many domains of our daily life we are used to work with models. Sometimes the application of models is so usual, that we even don't recognize anymore, that we are using models. Consider for instance the advertisements in journals, the public broadcast or the television. Here a typical consumer behavior is assumed which corresponds directly to a model of the aimed target group. Many rules or laws in human societies are also based on models. Here, the models depend on the underlying idea of man of the corresponding legislator and therefore it is by no means guaranteed, that this model is automatically the best choice. Consider for instance the death penalty in some states of the USA. This penalty is based on the assumption, that people won't commit severe crimes because of the severe penalty. But obviously, when looking at the statistics of severe crimes in these states, compared with other countries in the world or even states in the USA, where no death penalties are imposed, the underlying model of the behavior of human beings seems to be questionable.

In the field of computer-based simulations, obviously, the modeling has to be done in such a way, that computers can deal with the resulting models. Although this modeling can be done in many different ways, it is important to remark, that there are some common aspects caused by the basic working principles of computers [190, 193, 183, 195] and the resulting capabilities itself. These principles lead to a more or less generic structure of the models used in computer-based simulations. This is due to the fact, that computers, although in the meanwhile essential for our modern life, in the strict sense, only can store and manipulate data. The storage of data on the one hand leads immediately to the necessity of a ***data model*** whereas on the other hand the manipulation of data involves a corresponding ***action model*** or ***execution model***.

When the simulation deals with static objects only, one may think, that an execution model is not always necessary, but this is definitely wrong. The simulation itself introduces a dynamic aspect and hence leads to the necessity of an execution model. Consider for instance a virtual tour through a museum whereby no moving objects are considered. Although the museum and all objects are static, the tour itself involves a movement of the virtual visitor

or a virtual camera and introduces thereby dynamic aspects in the simulation. However, when dealing with the simulation of dynamic properties, the execution model is usually much more complex.
In the following some definitions of notions concerning computer-based simulation are given not in an exact sense but in a way, that they can be used and referred to in the remaining work.

Definition: ***Simulation entities***

The simulation entities are objects, states, properties and so on, which are involved in the simulation process. ■

Definition: ***Data model***

The data model describes, how simulation entities are represented in the computer memory. ■

Definition: ***Execution model***

The execution model defines in which way the data representing the simulation entities can be manipulated by actions. ■

Due to the restrictions of the finite computer memory and the finite performance of computer systems, one has to prioritize the entities and actions. Often this can't be done easily and not always in a unique manner. Hence, depending on the experience and the knowledge of the modeler, the resulting models may be different.
In ***virtual reality applications***, the priority defines the granularity of the virtual world itself whereas in ***real world applications*** it leads more or less to an approximation which is usually associated with a reduction of the number of considered entities and actions and hence the complexity. The complexity is often further reduced, because the actions in the execution model are usually only more or less accurate approximations of the actions possible in the real world. The internal representation of real world entities and actions in computer systems is often denoted as ***internalization***.
There are so many different modeling approaches and techniques, that it is beyond the scope of this work to discuss them all. However, because all modeling approaches are in some sense application oriented or application centered, the ***mathematical modeling*** is used in this work because it is considered as the most adequate modeling approach for the targeted purposes. The aim of mathematical modeling here is to derive computerized mathematical models of processes in terms of ***dynamical systems*** (see Sec. 3.2).

1.2.2 Mathematical modeling

Throughout this work, the following definition of mathematical modeling according to [41] is used:

Definition: ***Mathematical modeling***

Mathematical modeling is the process of creating a mathematical representation of some phenomenon in order to gain a better understanding of that phenomenon. It is a process that attempts to match observation with symbolic statement. During the process of building a mathematical model, the modeler will decide what factors are relevant to the problem and what factors can be de-emphasized. Once a model has been developed and used to answer questions, it should be critically examined and often modified to obtain a more accurate reflection of the observed reality of that phenomenon. In this way, mathematical modeling is an evolving process; as new insight is gained, the process begins again as additional factors are considered. "Generally the success of a model depends on how easily it can be used and how accurate are its predictions." ■

1.2.3 Simulation

In principle there exist two different types of ***simulation techniques***, or ***simulation methodologies*** namely the ***interactive simulation*** and the ***non-interactive simulation***. In most of the cases the ***simulation task*** itself determines the type of simulation technique to be used. Interactive ***simulation systems*** require ***interactions*** with the user or users of the system and therefore have to have a more or less precise model of the ***human behavior*** or at least of the possible ***human interactions***. This model is often denoted as the ***human model***. It is necessary for the simulation system to react in a proper and for the user in a more or less intuitive or at least understandable way.

Remark:

This human model is often unattended, which results in simulation systems which are difficult to handle and who therefore are often criticized or even not accepted.

Due to their different methodology, both simulation techniques require usually different simulation systems. Interactive simulations are often much more ***time-critical*** and require usually a much more complex ***human-machine interface***. In contrast to that, non-interactive simulation systems require often a more complex initialization phase, because after the initialization human interactions are no longer possible, so the control of the simulation has to be established in advance.

Remark:

There exist also simulation systems which cannot be uniquely assigned to one of the mentioned types of simulation systems and therefore belong in some sense to both classes.

1. The initialization phase of a non-interactive simulation systems is often done in an interactive way using a user interface which of course requires human interactions. Therefore these simulation systems could be considered as some sort of interactive simulation systems. However, because this interaction is usually completed after the initialization phase and could be done also in a non-interactive way, such kind of simulation systems are often considered as pure non-interactive simulation systems, although this assignment is definitely a matter of position or point of view
2. Most of the ***symbolic manipulation programs (SMP)*** or ***computer algebra systems (CAS)*** are typical examples here. On the one hand, they have an ***interactive mode***, where the user of the system can act by giving commands to the system and react according to the corresponding output given by the system. On the other hand, they have also a ***non-interactive mode***, where the ***kernel*** of the simulation system often denoted as ***computation engine*** performs the commands given by a list of pre-defined or pre-configured inputs non-interactively in a so-called ***batch mode***.

1.2.3.1 Interactive simulation

Typical examples for interactive simulation systems are for instance ***flight simulators***, ***drive simulators*** and in principal any simulator which involves movement. Here it is obvious, that the simulation system has to react directly and more or less instantaneously to the user actions according to the execution model (see Sec. 1.2.1). As already mentioned, these type of simulation systems behaves often time-critical. For instance, when dealing with ***real-time scenarios***, of course one has to obey ***real-time constraints***. In this context it is not important whether the simulation system is purely virtual or has also physical components. These components, or to be more precise ***actuators***, make the simulation more realistic for the users of the system and introduce thereby usually a lot of additional constraints but do not change the type of simulation system.

Although they are often not considered as simulation systems at all, ***computer games*** are definitely interactive simulation systems, because they posses all typical features of such systems. Besides the fact, that computer games are interactive simulation systems, it is worth to mention here, that modern computer games use not only mathematical modeling (see also Sec. 1.2.1) but also many sophisticated and complex algorithms from many fields of computer science. For instance, to control virtual beings - so-called bots - or machines or groups of them within the game, they often use excessively approaches, methods and algorithms from the field of ***artificial intelligence***.

There exist also a lot of visualization systems which belong to the class of interactive simulation systems. Here one uses the performance of modern computer systems to visualize complex processes, scenes, products and so on, which supports designers, developers and engineers. In the field of prototyping for instance, one often uses animated visualization to avoid the production of full-sized models which are usually very expensive.

Of course, there exist many other examples of interactive simulation systems. The above mentioned examples are only an illustrative selection, which is not meant to be in any way detailed or complete.

1.2.3.2 Non-interactive simulation

Typical examples of such simulation systems can be found in many scientific disciplines. Here the simulation task requires often a more or less complex initialization phase and after the initialization the system performs the simulation according to the execution model and the list of tasks defined by the user during the initialization.

Depending on the specific application, the simulations then may be very time consuming and may require ***high performance computer systems*** with a huge ***central memory***. In this application field the simulation itself is often not so important although it has to be done in an efficient, accurate, reliable (see Chap. 10) and reproducible way. Here the investigation and interpretation of the ***simulation data***, that is the data produced by the simulation process, is the most important task. When using non-interactive simulation systems, the analysis of the simulation data can be done either during or after the simulation or in both phases, but the interpretation usually is done after the simulation phase by the users of the simulation system.

The simulation and analysis package **AnT 4.669** presented in this work, belongs to the class of non-interactive simulation systems. After the mathematical modeling (see Sec. 1.2.2) and the usually short ***implementation phase*** of the dynamical system (see Sec. 3.2) - corresponding to the dynamical process under consideration - is completed, the simulation system has to be initialized according to the ***simulation tasks*** and ***investigation tasks*** which should be performed.

1.2.4 Analysis

In this last phase of the simulation process (see Fig. 1.1) the investigation or analysis of the simulation data takes place. Because this is the most challenging and sophisticated part of the simulation process, this part has to be done by human users of the simulation system, usually scientists or researchers. One can imagine, that in the future, specifically trained artificial intelligent systems support the users, but that of course depends strongly on the capabilities of the artificial intelligent systems and the investigation tasks to be performed. Usually it takes a lot of experience and knowledge to interpret simulation data correctly. Some of the examples in chapter 10 may justify this opinion. However, the interpretation of simulation data helps not only to understand the simulated process, but also to improve the mathematical model which is the basis of the simulation process.

Applying the procedure of modeling - simulation and analysis in an iterative manner, the knowledge about the observed process and its underlying principles increases step by step. As it is indicated in figure 1.1, the individual steps of the simulation process could of course also be repeated to accelerate and enhance the process of improvement and understanding.

Chapter 2

The AnT project

2.1 Aims

Whenever dynamical processes have to be investigated, one is first concerned with the problem of modeling this processes with appropriate mathematical models in terms of dynamical systems. In natural sciences the modeling process is mainly based on fundamental physical laws and hence often straight forward. In contrast to that in many other scientific disciplines this process is often not straight forward and as a consequence not always unique. But also in natural sciences the problem of abstraction occurs which leads to the modeling of dynamical processes on very different ***length scales*** and ***time scales***. Here one uses often either a ***bottom-up approach*** mainly based on ***averaging techniques*** or a ***top-down approach*** mainly based on some ***heuristics*** to get a mathematical description of the dynamical process under consideration in terms of dynamical systems on the ***mesoscopic level*** or ***macroscopic level***. Depending on the complexity of the system under consideration, this kind of modeling is sometimes very complicated and guarantees not always a unique mathematical description. In this cases the numerical simulation of the dynamical systems can support the modeling process itself. However after the modeling process is finished the dynamical systems have to be simulated and their behavior has to be analyzed in order to get the desired or required insight.

The current simulation and analysis package **AnT 4.669** was mainly developed to support scientists in research but also teachers and professors in education. In computer science it is a matter of fact, that the ***life cycle*** of software projects is decaying rapidly. This yields also for the simulation and analysis package **AnT 4.669**, as the first release which was written in the computer language C [87] (see the historical notes in [142]) is meanwhile obsolete. The current version is based on this predecessor but is completely redesigned in the object-oriented language C++ [158], [160] (see also the historical notes in [159]). In the initial state of this redesign several requirements were formulated and considered in the specification process.

Requirements:

- ▶ modern software concepts
- ▶ reuse-ability
- ▶ support of rapid development, maintenance and advancement
- ▶ open source project (see for instance [49] or [150])
- ▶ scientific computing (precision)
- ▶ distributed computing (performance)

Applied concepts from the field of computer science:

- ▶ modularity
- ▶ structured code
- ▶ separation of data recording administration from data processing and manipulation
- ▶ object-orientation
- ▶ design patterns

Usage of CASE-Tools:

- ▶ computer-based build process (GNU Autotools, [63, 64, 65, 165])
- ▶ computer-based documentation (Doxygen, [36])
- ▶ computer-based versioning (CVS, [25])

2.2 Historical remarks

In the early stages of the AnT project the main focus was the development of a simulation and analysis tool for two specific classes of dynamical systems, namely ordinary maps and ordinary differential equations. The challenge was to keep the architecture as flexible as possible, so that many dynamical systems can be investigated with this software. In this way the aim of the software was to support scientists and engineers. This predecessor of the current **AnT 4.669** software package was denoted as **AnT 4.66** and completely coded in C. Although some of the basic ideas are also incorporated in the new version, this predecessor was only a small tool compared with the totally redesigned current **AnT 4.669** software package. The current software was designed, mainly developed and is maintained by a small group of four people at the Institute of Parallel and Distributed Systems (IPVS) at the University of Stuttgart.

2.3 AnT 4.669: A software package for simulating and analyzing dynamical systems

The **AnT 4.669** software is open source software and published under the GNU [62] General Public License GPL [66, 68], which means that people all over the world are not only allowed to download the source code of the software but can also contribute to it. of the software To support the developer (staff and students) by the implementation and maintenance of the software, the concurrent versions system CVS [25] is used. Because from the beginning of the project it was the aim to support not only one specific operating system, a computer based build process has to be used. The AnT project uses the GNU Autotools [63, 64, 65, 165] to build the system on several platforms like Linux, Solaris, Free BSD but also on the Windows operating systems, where additionally the UNIX-like `cygwin` [26] or `mingw` [111] environments respectively libraries have to be used.

The code of the current **AnT 4.669** software is completely written in C++ and currently the software counts about 120.000 lines of code. The sources can be downloaded from the AnT project site [4]. On UNIX-like operating systems like Linux, Solaris, Free BSD the sources have to be compiled and installed using the usual configure mechanism. For the Windows platforms binaries are provided, because here the abovementioned additional environments and libraries are required for the configuration and compilation of the software package.

2.4 Running the AnT computation engine

This section describes, how to start the **AnT computation engine** directly via a UNIX shell command and how to use the command-line options. Although in chapter 8 the graphical user interface is presented from which the **AnT computation engine** could also be launched, it is important to know how to start with an already created initialization file. This is especially necessary when performing very time consuming computations on large clusters. Of course it is not an adequate way to start such a computations by using the graphical user interface on each node. It is very convenient in such cases to use shell-scripts using the possibility to start the **AnT computation engine** in command-line mode. However, once an initialization file (see Sec. 8.2 and especially Sec. 8.2.1) is created, the **AnT computation engine** could be launched using this initialization file and additional command-line options.

The command-line options have been designed to match standard UNIX behavior. When called without any arguments, the **AnT computation engine** shows the overview of all possible command-line switches and stops with an error message. When called with the usual help switches `-h`, `-H` or `--help` the same overview is shown and the execution stops without an error message.

```
usage: AnT <systemname>
[{-i | -I | --initialization} <configfile>]
[{-m | -M | --mode} <runmode>]
[{-s | -S | --server} <server name>]
[{-p | -P | --port} <portnumber>]
[{-n | -N | --points} <scanpoints>]
[{-t | -T | --time} <seconds>]
[{-v | -V | --version}]
[{-v | -V | --log}]
[{-h | -H | --help}]

<systemname>
    complete path and filename (without extension)
    of the shared library containing at least the system
    function for the dynamical system to be simulated.
```

```
Options:
{-i | -I | --initialization} <initialization file>
    complete path and filename of the initialization file
{-m | -M | --mode} <runmode>
    where runmode is one of 'standalone',
    'server' or 'client'. Default is 'standalone'.
{-s | -S | --server} <server name>
    for runmodes 'server' and 'client' only.
    Default is the standard hostname of the current system.
{-p | -P | --port} <portnumber>
    for runmodes 'server' and 'client' only.
    The default port is 12345.
{-n | -N | --points} <scanpoints>
    for runmode 'client' only.
    The number of scanpoints the client
    should fetch from the server. Default is 50.
{-t | -T | --time} <seconds>
    for runmode 'client' only. The (approximate) number
    of seconds the client should be busy before asking
    for new scan points from the server.
    This option overrides the '-n' option.
{-v | -V | --version}
{-l | -L | --log} write the log-file 'transitions.log'
    which shows the internal structure of the current
    simulator instantiation.
{-h | -H | --help}
```

2.4.1 General options

There are five options which are meaningful for the standalone run-mode as well as for networked operation (see Chap. 7):

▶ The only mandatory option is the `<systemname>`. It determines the dynamical system that will be simulated. The dynamical system is contained in a shared library. This shared library is automatically found when the `LD_LIBRARY_PATH` environment variable contains the directory where the system is installed. Otherwise when the complete path to the library is given with this switch. The name of the system can be be given with or without the extension of the shared object `.so` or dynamic link library `.dll`. If this option is not given, the **AnT computation engine** displays the usage text as seen above and exits immediately.

- The `-i` option can be used to point to an initialization file. By default, the **AnT computation engine** looks for an initialization file with the same name as the given system. If for instance the name of the system is `logistic`, then the **AnT computation engine** will look for the initialization file `logistic.ant` in the current directory. In the client run-mode this switch is ignored, because the initialization of a client is always defined by the initialization file that the server is started with.
- The `-v` option shows the current version of the **AnT computation engine** and exits. Currently an invocation with this option yields to the output: `Version:  AnT-core-4.669-R3a`
- The `-l` option activates the logging mechanism of the **AnT computation engine**. The log-file 'transitions.log' which shows the internal structure of the current simulator instantiation is created during the initialization phase of the execution. This option is mainly for debugging purposes.
- The `-h` option prints the above listed usage.

2.4.2 Client/server specific options

The rest of the options are only useful for the client/server mode of the **AnT computation engine**. All of these switches are optional.

- The `-m` option sets the run-mode of the **AnT computation engine**. This can be either `standalone`, `server` or `client`. The default run-mode is `standalone`.
- The `-s` option sets the network address of the server. This can either be an IP-address or a hostname. By default, the standard hostname as reported by the standard C-library call `gethostname()` is used both in run-mode client and server. In the server run-mode this option can be used to define the address that the server should bind to. In the client run-mode this option can be used to tell the client where the server is located on the network.

- The `-p` option sets the network port of the server. The default port is 12345. In the run-mode server this option can be used to define the network port that the server should bind to. In the client run-mode this option can be used to tell the client on which port the server is listening for incoming connections.

- The `-n` and the `-t` options are both only evaluated in the client run-mode. Both options ultimately specify how many scan-points a client should fetch from the server during one network communication. The `-n` option instructs the client to fetch the specified number of scan-points. The `-t` option instructs the client to fetch as many scan-points as are needed to keep it busy for a certain amount of time (given in seconds). During the first communication the client fetches either the default of 50 scan-points or the number of scan-points specified with the `-n` option. The number of scan-points that are fetched are reconsidered before every subsequent communication with the server (based on the time needed for the previously calculated scan-points). If both options are given, the `-t` option takes priority. As already mentioned, in this case the value given by the `-n` option determines the number of scan-points that are fetched during the first communication with the server. If neither the `-n` nor the `-t` option is given, a fixed amount of 50 scan-points is fetched by the client.

2.4.3 Examples

In this section the usage of the command-line switches of AnT will be illustrated by some examples.
To simply use the **AnT computation engine** in run-mode standalone and simulate for instance the logistic map, the following command-line is sufficient:

```
AnT logistic
```

For this call to be successful, the AnT binary needs to be installed and reside in a directory that is part of the `$PATH` environment variable. Additionally, the position of the shared library of the logistic map (`logistic.so`) has to reside in the `$LD_LIBRARY_PATH` environment variable and the configuration file `logistic.ant` has to be in the current directory.
If another initialization is to be used, it can be specified with the `-i` option:

```
AnT logistic -i ./test.ant
```

To start an AnT-server, it is sufficient to add the option `-m`. No other additional option is strictly necessary:

```
AnT logistic -m server
```

To start an AnT-client, some more options can be useful. First, the network-address of the server has to be given (if the server is not running on the same host as this client). Then it is advisable to specify how many scan-points the client should fetch from the server at once. The recommended option for this is `-t`:

```
AnT logistic -m client -s serverhostname -t 10
```

`-t 10` implies that the client should try to fetch scan-points from the server every ten seconds. The client tries to figure out how many scan-points are needed to keep it busy for that time, based on its current speed. This option will influence the overall performance of the simulation run. If too few scan-points are fetched, the communication overhead will slow down the simulation. If, on the other hand, too many scan-points are fetched, the server has to keep more scan-results in memory - if there is more than one client - and at the end of the scan-run more scan-points will be calculated twice by different clients and therefore computing power is wasted.

Chapter 3

Supported classes of dynamical systems

3.1 Introduction

In this chapter the classes of dynamical systems which can be simulated and investigated by the simulation package **AnT 4.669** are described. Therefore, in a first step the concept of a dynamical system within the framework of mathematical modeling (see Sec. 1.2.1) is considered in more detail and especially the one used in this work is presented.

Remark:
It is beyond the scope of this work to give exact definitions - if possible at all - for all concepts or notions used here. Definitions are only given in the case when they are necessary for the understanding and when they are not - in some sense - common knowledge in the corresponding scientific community. For instance the **state** of a dynamical system, although widely used in this work is not defined here. Especially many of the notions from the field of nonlinear dynamics are not defined here. For definitions on this topics it is referred to the literature or the information provided by the world wide web. A good first choice in this context, where one can get often further references, would be [181].

3.2 Dynamical systems

Depending on the field of interest, there exist several possibilities how to describe or define a ***dynamical system***. In [194] for instance, the following definition is given:
”A dynamical system is a deterministic process in which a variable’s value changes over time according to a rule that is defined in terms of the variable’s current value.”
Because this definition covers not the case of dynamical systems with memory, the following more general definition of a dynamical system according to [172] is used:

Definition: *Dynamical system*

A description how one state of a system develops into another state over the course of time is called a dynamical system. ■

However, according to the notion *dynamical system* all definitions have to define at least the following three ingredients: The part *dynamical* requires not only the definition of the ***concept of time*** but also the ***concept of evolution in time***, whereas the part *system* requires the definition of the ***concept of state***.

Remark:
In relativity theory usually the concept of space time is used, where the time is mathematically treated as a fourth component of the extended space time state usually denoted as a 4-vector. But it is important to notice, that also in this case the time plays a specific role. This can easily be seen, when looking at the components of the metric tensor in Cartesian coordinates, where the component corresponding to the time has a different sign.

The concept of evolution in time leads immediately to the ***concept of transition***, which defines the time evolution of the system. Depending on the fact, whether the transition defines a finite or an infinitesimal small time step of the evolution, the corresponding dynamical system is denoted as discrete or continuous in time.

Definition: *Transition*

The change from one state of a system to another is called a transition. ■

Chapter 3
Supported classes of dynamical systems

According to the abovementioned comments throughout this work the following, transition- and state-oriented description of a dynamical system is used:

A state of a dynamical system is described by a so-called ***state vector*** $\underline{s}$ with as much ***state components*** or ***state variables*** s_i as the mathematical model of the dynamic process under consideration requires usually denoted as the number of ***degrees of freedom*** of the system. The state vector is defined in a corresponding ***state space*** Γ_s, whose dimension N_s is given by the number of components of the state vector denoted as ***state space dimension***. Furthermore, the concept of transition defines an ***instruction set*** (one or more instructions) which specify, how the state vector changes or evolves in the course of time. These instructions are therefore usually denoted as ***evolution equations*** or ***equations of motion***, but generally any ***rule-based system***, for instance a ***state chart***, can be used as instruction set. Of course, the instruction set depends on the state vector but usually also on a set of further quantities denoted as ***system parameters***. The set of system parameters can be interpreted as a ***parameter vector*** $\underline{p}$ with as much components p_i as the mathematical model of the dynamic process under consideration requires. The parameter vector is defined in a corresponding ***parameter space*** Γ_p, whose dimension N_p is given by the number of components of the parameter vector denoted as ***parameter space dimension***.

Concerning the notion or concept of a state space, one has to remark, that related ideas from physics are the notations ***configuration space*** and ***phase space*** (see for instance [184]). Although the configuration space has no single meaning, it is in this context, the set of values of all ***generalized coordinates*** of a system defines a point in the configuration space. For instance, for N particles moving in a M dimensional space, the dimension of the configuration space is $N \times M$ because the configuration is described by the positions, which are in this case the generalized coordinates, of all particles. Analogously, the set of values of all generalized coordinates and ***generalized momenta*** defines a point in the phase space. Hence, the dimension of the phase space of N particles moving in a M dimensional space is $N \times M \times 2$ because each generalized coordinate has a conjugate momentum. In short, a configuration space is typically "half" of the corresponding phase space. The notion of state space in this work is strongly connected to the notion of phase space, because the dimension of the phase space is equivalent to the number of degrees of freedom of a system.

Because the mathematical modeling is often done in a straight forward manner, it introduces parameters which are in the pure mathematical sense not necessary, although they are naturally convenient for the understanding of the underlying principles of the considered dynamical process. In order to reduce complexity one usually gets rid of these ***redundant parameters*** by an appropriate scaling of space or the state variables respectively and time. The resulting model has a reduced complexity because of the ***minimal set of parameters*** and hence a decreased parameter space dimension (see App. A for an illustrating example).

As already mentioned, it is distinguishes between two large classes of dynamical systems depending on the defining transition or mode of operation of the instructions.

1. Systems with an instruction set which affects the state vector directly, describing a single transition from the current timestamp to the next one in the future, are denoted as dynamical systems discrete in time because here the time evolves in discrete steps with a finite step size. As a consequence, states between two successive timestamps are not defined in dynamical systems of this kind.

2. In contrast to that, systems with an instruction set which affects the state vector only indirectly through the derivative of the state vector with respect to time, describing an infinitesimal transition from the current time to the infinitesimal adjacent time in the future, are denoted as dynamical systems continuous in time. As a consequence, the state of dynamical systems of that kind are defined at any time in its domain[1].

Remark:
Numerical computer simulations of dynamical systems are always discrete in time, even when dealing with systems continuous in time. This is due to the fact, that starting with a state vector at a given time t one is only able

[1]Usually, this domain is given by the infinite interval $[t_0, +\infty]$, but can also be a finite interval $[t_0, t_1]$ or even a set of finite intervals

to compute an approximation of the state vector at a certain time $t + \Delta t$ in the future using a numerical integration scheme. There are so many different integration schemes that their description is beyond the scope of this work. The **AnT 4.669** software package supports many integration methods with fixed step size denoted as steppers. Among them are classical one-step ***steppers*** using Runge-Kutta type integration schemes or user defined Butcher arrays as well as multi-step steppers, implicit steppers and predictor-corrector steppers. Additionally three different mechanisms to adjust the step size are implemented. They are referred to as ***wrappers***. These are: a gradient based wrapper, a wrapper based on the comparison of two different steppers and a third wrapper based on the comparison of the integration result obtained by a stepper using some step size and the result obtained with the same stepper using two steps of half the step size.

Due to the fact, that the instruction set may not only depend on the current state vector but also on several state vectors or even a complete time interval in the past one can distinguish between systems with memory and such without memory.

Furthermore the state vector may not only depend on the independent variable describing the time, but also on some other independent variables which often represent the coordinates of the underlying physical space. This systems are denoted as inhomogeneous systems, because the instruction set depends also on this additional independent variables and the derivatives of the state vector with respect to this additional independent variables.

Finally, the state vector can have an internal structure with a continuous and a discrete part. By definition, the state components or state variables of the continuous part take continuous values whereas the components of the discrete part take discrete values. Systems of this kind are denoted as hybrid dynamical systems.

As one can see, there are a lot of characteristic properties of dynamical systems which can be used to classify them more accurately. This will be done in Sec. 3.5.

Due to the applied generic concept of a dynamical system, all the following different classes are supported by **AnT 4.669** and can hence be simulated and investigated. This is an important point, because here a strong connection between the mathematical description and the software architecture exists. Each difference in the mathematical description of the classes has to have an suitable correspondence in the software architecture. How this problem is treated in the **AnT 4.669** software package is described in more detail in chapter 6.

Remarks:

1. Throughout this work only autonomous dynamical systems, that is, systems which do not explicitly depend on time, are considered. This is due to the fact, that non-autonomous dynamical systems can be easily converted into autonomous ones by extending the state space introducing a new state variable representing the time. Hence the new state vector has an additional component and the dimension of the state space is increased by one.

2. Depending on the modeled dynamical process, the components of the continuous part of the state vector may be real or complex numbers. Because any complex-valued number consists of its real and imaginary part which can be considered separately throughout this work the components of the continuous part of the state vector are assumed to be real-valued.

The example systems presented in the following sections are selected and often well-known representatives of the corresponding class. Most of them are mathematical models of real dynamic processes. For the derivation of the equations and their meaning it is referred to the literature, because this is not relevant in the context of this work.

3.3 Dynamical systems discrete in time

As already mentioned, the instruction set describing how the state vector evolves in time affects for this class of dynamical systems the state vector directly. Hence the time evolves in discrete time steps with a finite step size.

Definition: ***Dynamical system discrete in time***

A dynamical system whose time evolution is given in such a way, that the defining transition leads directly from one state of the system to the next one with a finite time step is called a dynamical system discrete in time or a time discrete dynamical system. ■

One step in the time evolution requires the execution of the defining transition, whereby it is not required, that the time steps are all equal during the time evolution.

3.3.1 Systems without memory

For systems belonging to this class the vector field depends not on state vectors with timestamps in the past, it only depends on the current state vector and the set of system parameters, i.e. the parameter vector. Hence an adequate mathematical description of such a system reads:

3.3.1.1 Ordinary maps (Maps)

The evolution equation or equation of motion of the class of dynamical systems denoted as ordinary maps is defined by:

Definition: ***Ordinary map (1)***

$$\underline{s}(t_{n+1}) \quad = \quad \underline{f}\left(\underline{s}(t_n), \underline{p}\right) \tag{3.1}$$

with

$$\underline{f} : \Gamma_s \times \Gamma_p \mapsto \Gamma_s$$

■

An alternative definition considers explicitly the discrete time steps omitting thereby the more or less redundant t in the formulas and is given by:

Definition: ***Ordinary map (2)***

$$\underline{s}_{n+1} \;=\; \underline{f}\left(\underline{s}_n, \underline{p}\right) \tag{3.2}$$

with

$$\underline{f} : \Gamma_s \times \Gamma_p \mapsto \Gamma_s$$

■

Here $\underline{s}(t_n) = (s_1(t_n), s_2(t_n), \ldots, s_{N_s}(t_n)) \in \Gamma_s$ or $\underline{s}_n = (s_{1n}, s_{2n}, \ldots, s_{N_s n}) \in \Gamma_s$ is the state vector of the system, t_n is the discrete time at timestamp $n \in \mathbb{N}_0^+$, $\underline{p} = \left(p_1, p_2, \ldots, p_{N_p}\right) \in \Gamma_p$ is the parameter vector and $\underline{f} : \Gamma_s \times \Gamma_p \mapsto \Gamma_s$ a usually nonlinear vector field representing the interactions between the single components or variables of the state vector. In this work, mainly the definition (3.2) is used.

In the following paragraphs some illustrative examples of this class will be presented.

The logistic map
The most famous example of this class of dynamical systems is the well-known ***logistic map*** defined by the following nonlinear ***recurrence equation***:

Definition: ***Logistic map***

$$x_{n+1} \;=\; f(x_n, \alpha) = \alpha x_n (1 - x_n) \tag{3.3}$$

with

$$\text{with} \quad x \in [0,1] \subset \mathbb{R} \quad \text{and} \quad \alpha \in [0,4] \subset \mathbb{R}$$

■

Some historical remarks
The history of the logistic map at least goes back to the year 1838, when Verhulst [166] published what he called a "logistique" equation. In this

publication he wants to describe mathematically the sigmoidal growth of population density to carrying capacity by a dynamical system continuous in time (see Sec. 3.4). The equation was rediscovered by Pearl and Reed [132] in 1920. In 1925 Lotka [99] derived the same equation, calling it "the law of population growth". Nine years later Gause [57] demonstrated the validity of the logistic growth model in laboratory experiments. See [88] for an review of these historical events. However, the discrete form of the logistic equation presented here was first proposed in 1965 by Cook [20] but it turned out, that it is identical to the equation Ricker published in 1954 [140] (see for instance [11]).

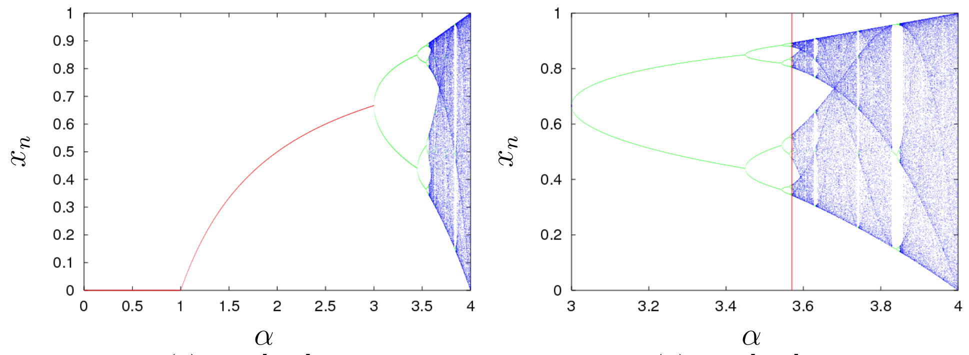
Although the logistic map (3.3) is only one-dimensional and "looks" very simple, it is capable of a very complex dynamics, which is the reason why this relatively simple dynamical system has been investigated up to now and will be so in the future. However, the more it is known about the details of the complex dynamic behavior of this system, the more it becomes clear why many scientists speak about the "theory of deterministic chaos in a nutshell" when referring to the logistic map.

An overview about the complex dynamics of this system is given by its ***bifurcation diagram*** (see Fig. 3.1). From this one can read off:

- ▶ For $\alpha \in [0,1]$ the system has $x_0^* = 0$ as the only stable ***fixed point***, whereby the interval $[0,1]$ is the ***basin of attraction*** of this fixed point, which holds for all types of ***attractors*** of the logistic map. That means, for all ***typical initial values*** x_0, the ***solution curves*** or ***trajectories*** converge to the stable attractors. Hereby the notion of a typical initial values means, that the initial value x_0 belongs not to an unstable invariant set.

- ▶ At the parameter value $\alpha = 1$ a ***transcritical bifurcation*** occurs and the fixed points $x_0^* = 0$ and $x_1^* = \frac{\alpha-1}{\alpha}$ change their stability. The fixed point x_1^* is stable for $\alpha \in [1,3]$. In figure 3.2(a) a typical graphical iteration towards a fixed point existing in this parameter interval at $\alpha = 2.8$ is shown. This type of representation is denoted as ***graphical iteration***, ***cobweb diagram*** or simply ***web diagram*** (see [192]).

- ▶ At the parameter value $\alpha_1 = 3$ the first ***flip bifurcation*** or ***period doubling bifurcation*** occurs and a ***limit cycle*** with period



(a) $\alpha \in [0, 4]$ (b) $\alpha \in [3, 4]$

Figure 3.1: Logistic map: bifurcation diagrams

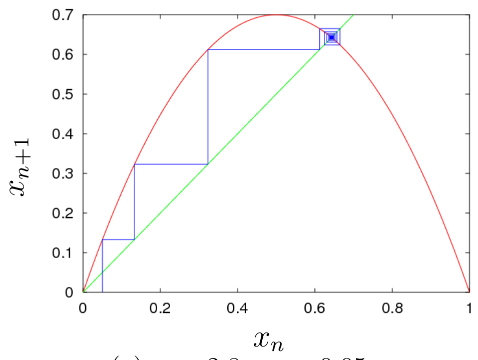
In (a) the complete bifurcation diagram is shown, whereas in (b) only the period doubling scenario and the chaotic regime is shown. The vertical line in (b) marks the accumulation point $\alpha_\infty \approx 3.569945672$, i.e, the end of the period doubling cascade.

two emerges, which is stable for $\alpha \in [\alpha_1, \alpha_2]$, with $\alpha_2 = 1 + \sqrt{6} \approx 3.449489\ldots$. This is the beginning of the ***period doubling cascade*** which ends at the ***accumulation point*** $\alpha_\infty \approx 3.569945672$ with the emergence of the famous ***Feigenbaum attractor*** which is a ***strange attractor*** but not a ***chaotic attractor***. In figures 3.2(b) and 3.2(d) graphical iterations toward limit cycles of periods two and eight at the beginning of the period doubling cascade are shown.

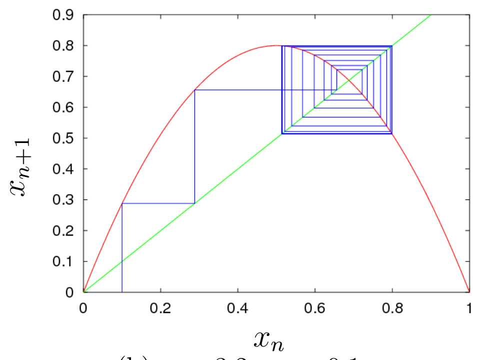
- For $\alpha \in [\alpha_\infty, 4]$ the logistic map shows a ***chaotic behavior*** in the sense of ***deterministic chaos*** in contrast to ***stochastic chaos*** or ***microscopic chaos***. In the ***chaotic regime*** there exist parameter intervals so-called ***parameter windows*** or only windows, where periodic behavior can be found. For instance exist around the parameter value $\alpha \approx 3.83$ a limit cycle with period three (see Fig. 3.2(c)). For a more detailed description and a collection of references see for instance [179].

Some remarks about the parameter value $\alpha = 4$

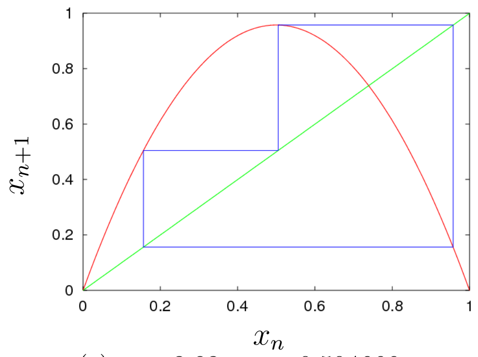
At this parameter value, the logistic map has some remarkable properties which are also useful for tests of investigation methods (see Secs. 5.5 and 11.1).



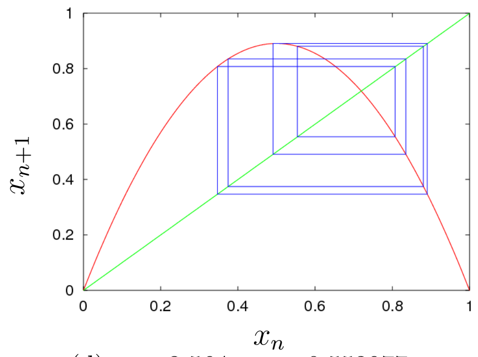
(a) $\alpha = 2.8$, $x_0 = 0.05$



(b) $\alpha = 3.2$, $x_0 = 0.1$



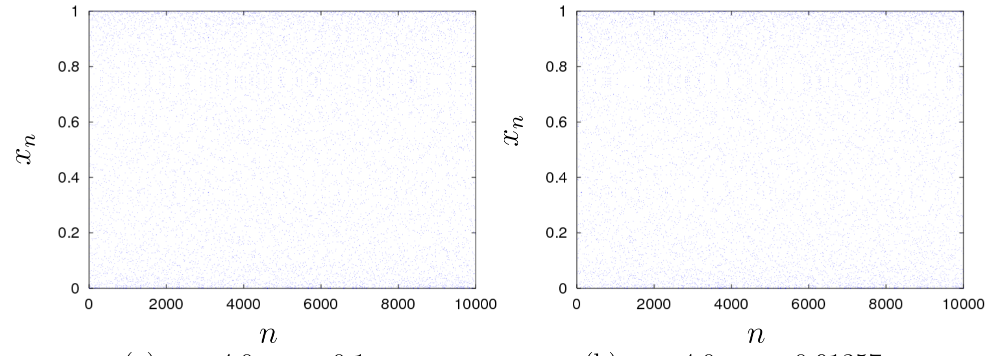
(c) $\alpha = 3.83$, $x_0 \approx 0.504666$



(d) $\alpha = 3.564$, $x_0 \approx 0.553977$

Figure 3.2: Logistic map: cobweb diagrams

Graphical iterations or so-called cobweb diagrams or simply web diagrams of the logistic map. The trajectory converges to a fixed point (a), a limit cycle of period two (b), a limit cycle of period three (c) and one of period eight (d).



(a) $\alpha = 4.0$, $x_0 = 0.1$ (b) $\alpha = 4.0$, $x_0 = 0.91357$

Figure 3.3: Logistic map: dynamic behavior at $\alpha = 4$

The first 10^4 *iterations of the logistic map at the parameter value* $\alpha = 4.0$ *for two different initial conditions* $x_0 = 0.1$ *(a) and* $x_0 = 0.91357$ *(b) demonstrating the chaotic but not uniform distribution of the iterated points on the interval* $[0, 1]$.

- ► It was John von Neumann [183, 195], who suggested in the late 1940s to use the logistic map at the parameter value $\alpha = 4$ as a random number generator, which becomes quite understandable, when, looking for instance at the first 10^4 iterations (see Fig. 3.3).

- ► At this value the ***diffeomorphism*** (see for instance [170]) $y = \frac{2}{\pi}\arcsin(\sqrt{x})$ between the logistic map $x_{n+1} = 4x_n(1 - x_n)$ and the tent map $y_{n+1} = 1 - 2|y_n - \frac{1}{2}|$ exists, which allows the analytic calculation of the ***natural measure*** or ***invariant measure*** $\rho(x) = \frac{1}{\pi\sqrt{x(1-x)}}$ of the chaotic attractor of the logistic map. Due to this fact, all quantities which are determined by the invariant measure can be calculated analytically. For the ***Lyapunov exponent*** for instance one gets the analytic result:

$$\lambda = \int_0^1 \rho(x) \log|\frac{df(x,4)}{dx}|dx = \log 2 \tag{3.4}$$

- ► In [164] it was proved, that the logistic map at the parameter value $\alpha = 4$ can be solved using the following explicit function:

$$x_n = \frac{1}{2}\left(1 - \cos\left(2^n \arccos(1 - 2x_0)\right)\right) \tag{3.5}$$

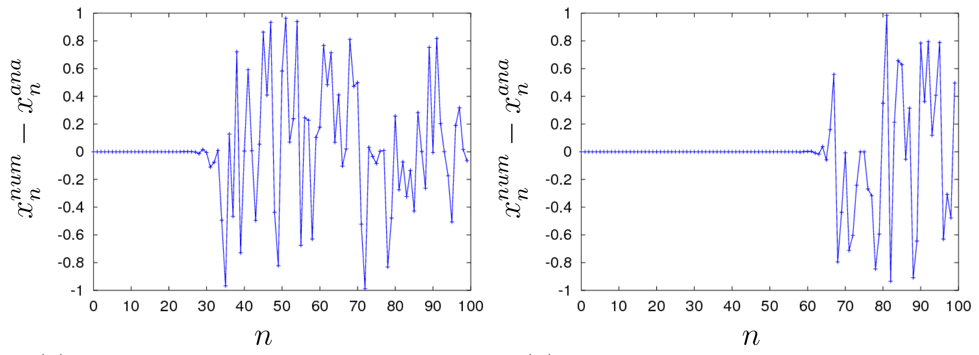
This result can be used to illustrate the behavior which lead to the notion of deterministic chaos, by comparing an iterated trajectory using a certain precision during the calculation with the exact trajectory given by Eq. (3.5).

In figure 3.4 this is done with two different precisions, namely 10 digits and 20 digits accuracy. Shown is the difference of the numerically calculated trajectories to the analytical one determined by Eq. (3.5) in the course of the iteration. As one can see, the trajectories diverge in both precisions although as expected, the more precisely calculated one stays longer at the analytical trajectory determined by Eq. (3.5). This illustrates the well-known phenomenon in the theory of nonlinear dynamics denoted as ***initial condition sensitivity***. This notion is somehow misleading and misunderstanding, because not only very small differences in the initial conditions can lead to a divergence of the trajectories, but any deviation or disturbance, which may occur in the course of time. The deviation could be caused by stochastic processes or truncation effects due to the floating point number representation within the computer memory. In fact, for the trajectories shown in figure 3.4 the same initial condition was used, but the different precision in connection with the dynamics of the system, which shows deterministic chaos, leads to the divergence of the trajectories.

For more information including references about the logistic map see for instance [179, 196].

The Hénon-Lozi map

Another typical example of an ordinary map is the Hénon-Lozi map. It is a map with state space dimension $N_s = 2$ and parameter space dimension $N_p = 3$ defined by:



(a) $\alpha = 4.0$, $x_0 = 0.1$, precision=10 (b) $\alpha = 4.0$, $x_0 = 0.1$, precision=20

Figure 3.4: Logistic map: comparison of analytic and numeric results
Difference of the numerical iterations x_n^{num} with two different precisions, carried out with Maple [105], to the analytical solution x_n^{ana} determined by (3.5). In (a) the used precision during iteration was 10 digits, whereas in (b) it was 20 digits. As expected, the more precisely calculated trajectory in (b) stays longer on the analytically determined trajectory.

Definition: *Hénon-Lozi map*

$$\underline{x}_{n+1} = \underline{f}(\underline{x}_n, \underline{p}) \qquad \begin{aligned} \underline{x}_n &= (x_n, y_n)^T \in \mathbb{R}^2 \\ \underline{p} &= (a, b, \gamma) \in \mathbb{R}^3 \\ \underline{f}(\underline{x}_n, \underline{p}) &= \begin{pmatrix} 1 - a|x_n|^\gamma + y_n \\ bx_n \end{pmatrix} \end{aligned} \tag{3.6}$$

■

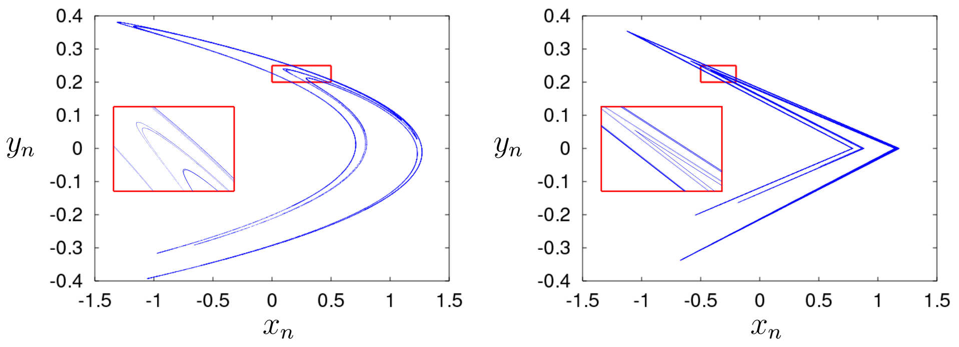
There are two special cases of this map, namely the famous Hénon map [77] for $\gamma = 2.0$ and the Lozi map [100] for $\gamma = 1.0$.

The Hénon map

Definition: *Hénon map*

$$\begin{aligned} x_{n+1} &= 1 - \alpha x_n^2 + y_n \\ y_{n+1} &= bx_n \end{aligned} \tag{3.7}$$

■



(a) $a = 1.4$, $\gamma = 2.0$ (b) $a = 1.8$, $\gamma = 1.0$

Figure 3.5: Hénon-Lozi map: attractors and blow-ups

(a) A chaotic attractor of the Hénon map
(b) A chaotic attractor of the Lozi map

In both figures, blow-ups of the rectangularly marked regions are shown.

The Lozi map

Definition: *Lozi map*

$$\begin{aligned} x_{n+1} &= 1 - \alpha|x_n| + y_n \\ y_{n+1} &= bx_n \end{aligned} \qquad (3.8)$$

■

In figure 3.5 two typical chaotic attractors of system (3.6) corresponding to the two special case of the Hénon map and the Lozi map are shown. For more information about these two systems it is referred to [177, 180].

Remark:
The Hénon-Lozi map is interesting, because in this system a transition from a smooth to a non-smooth vector field takes place, when the system parameter γ is changed from the value of 2, representing the case of the Hénon map to the value of 1, representing the Lozi map. Thereby a ***two-parametric bifurcation*** takes place which is denoted here as a ***period doubling big bang bifurcation***. This and related phenomena were studied in detail in [7], whereby the scan capabilities (see Chap. 4)

and the investigation methods (see Chap. 5) of the **AnT 4.669** software package were used intensively.

As a standard example of two-dimensional maps, the Hénon map has been studied intensively for instance by [145, 69, 79, 60, 93, 116]. In figure 3.6 the most presented and best known attractor of the Hénon map is shown again together with three successive blow-ups demonstrating the geometric self-similarity [133, 94] of chaotic attractors [61]. This geometric self-similarity can be described, in a sloppy way, by the lack of a natural length scale. That is, when zooming into images of perfect self-similar objects, one is never able to indicate or determine a length scale from the considered section of the image, because the characteristic structures of the self-similar object are repeated on smaller and smaller length scales - again and again. This property is clearly illustrated by the successive blow-ups of parts of the attractor in figure 3.6.

Self-similar structures are typical for chaotic attractors. There are mainly three reasons for that: Firstly, because chaotic attractors are typically confined to some region in the state space. Secondly, because the chaotic behavior is defined by the exponential divergence of adjacent trajectories and thirdly, because the equations of motion are deterministic and represent from the viewpoint of geometry often very simple rules. The contradiction of the first two reasons can only be explained by more or less complex folding, squeezing and stretching mechanisms which bend and keep the trajectories in the confined region. In addition to that, one may say, that the simple geometric rules lead to repetitions of the processes and hence cause the emergence of self-similar structures.

3.3.1.2 Coupled map lattices (CMLs)

Coupled map lattices are a specific subclass of dynamical systems discrete in time. The notion is based on the fact, that they are build up from an ordinary map, which defines a single cell of the coupled map lattice. The single cells are specifically structured or arranged and this arrangement defines the lattice. Usually this is a one-dimensional lattice, but multi-dimensional lattices with different lattice-topologies are possible too. The single cells arranged on the lattice, are coupled with other cells by so-called ***coupling parameters***. Usually ***local coupling*** mechanisms occur, that means a

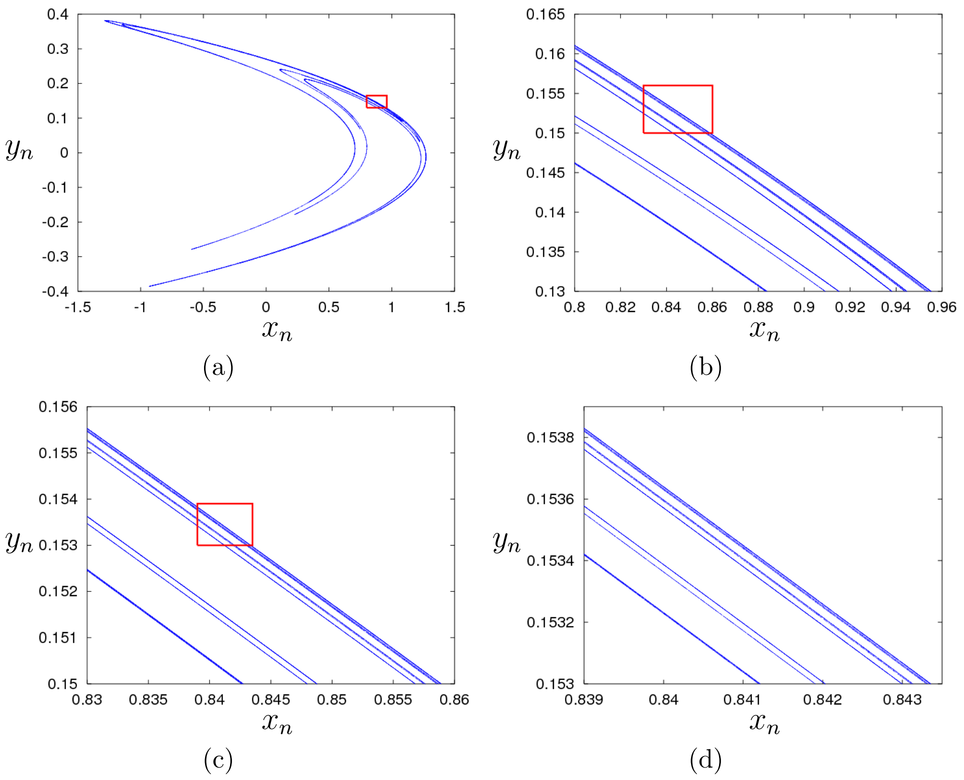


Figure 3.6: Hénon map: attractors and blow-ups

Attractor of the Hénon map (a) and successive blow-ups (b),(c),(d) demonstrating the self-similarity of chaotic attractors. Parameter setting: $a = 1.4$, $b = 0.3$, $\gamma = 2.0$.

coupling, such that an individual cell is influenced only by some cells in its neighborhood. How large this neighborhood is, is part of the definition of a coupled map lattice but can also be determined by a system parameter. Of course also ***global coupling*** mechanisms are possible. Sometimes, the lattice can be considered as space points and hence coupled map lattices are some kind of spatial inhomogeneous systems similar to partial differential equations.

Remarks:

1. CMLs represent not really a new class of dynamical system, because they are still maps. However, the structured arrangement of many identical subsystems is mathematically treated usually by introducing indexed quantities, which describe the subsystems and lead to a compact notation. This idea of structuring and compactification holds also for the implementation of such systems. Instead of implementing the equation of motion of all cells, one has to implement the equation of motion only for one cell, whereby the rest can be done automatically. The idea of supporting a user in this way lead to the implementation of these class in the **AnT 4.669** software package.

2. The lattice may represent not only a spatial structure or arrangement, but any indexable quantity. In the **AnT 4.669** software package only CMLs with a single indexable quantity are supported, because CMLs with multiple indexable quantities can be mapped to that case, although the couplings become much more complicated then.

Coupled map lattices are defined by:

Definition: ***Coupled map lattice (CML)***

$$\underline{S}_{n+1} = \underline{F}\left(\underline{S}_n, \underline{p}\right) \tag{3.9}$$

with

$$\begin{aligned}
&\underline{S}_n = \left(\underline{s}_n^1, \underline{s}_n^2, \ldots, \underline{s}_n^M\right)^T \in \Gamma_s^M \\
&\underline{s}_n^i \in \Gamma_s\,, \quad i = 1, \ldots, M \\
&\underline{F} : \Gamma_s^M \times \Gamma_p \mapsto \Gamma_s^M = \left(\underbrace{\underline{f}, \underline{f}, \ldots, \underline{f}}_{M \text{ times}}\right)^T \\
&\underline{f} : \Gamma_s^K \times \Gamma_p \mapsto \Gamma_s
\end{aligned}$$

■

Here M are the number of cells of the lattice, N is the dimension of the state vectors $\underline{s}_n^i$ of the individual cells and $\underline{S}_n$ is the $N_s = NM$ dimensional state vector of the coupled map lattice. The parameter K is the number of coupled cells. Hence, the state vector $\underline{S}_n$ of a coupled map lattice consists of a number of $M > 1$ sub-states $\underline{s}_n^i$ of the single ordinary maps defining a cell of the lattice. Accordingly, the system function $\underline{F}$ of a coupled map lattice is build up from the system functions $\underline{f}$ of the individual cells, whereby, the parameter vector $\underline{p}$ is taken into account only once. It is important to remark, that for a coupled map lattice, the system functions $\underline{f}$ of all individual cells and the parameter vector $\underline{p}$ are identical. Of course a definition analogous to (3.1) is also possible.

Remark:

One could also think of a coupled map lattice consisting of two types of cells with different system functions arranged on the lattice in an alternating manner. The definition (3.9) given above, covers also this case if one takes into account, that the two cells could be combined to a larger cell with a system function build up from the two different system functions. This new cells are now arranged on a lattice of half the size of the original one and have identical system functions as required by definition (3.9). The definition (3.9) covers even more complex situations as long as the cells with different system functions form a periodic structure.

The equation of motion of the i-th cell of the coupled map lattice is defined by:

Definition: *CML: single cell*

$$\underline{s}^i_{n+1} \quad = \quad \underline{f}\left(\{\underline{s}^{i_k}_n\}, \underline{p}\right) \tag{3.10}$$

with

$$i_k \in \{1, 2, \dots, M\}\,, \quad k = 1, \dots, K \quad \text{and} \quad \underline{f} : \Gamma_s^K \times \Gamma_p \mapsto \Gamma_s$$

■

Hereby the set $\{\underline{s}^{i_k}_n\}$ consists of K states which influence the considered cell i. Note, that according to the definition (3.9) all cells have the same parameter vector $\underline{p}$ representing the set of parameters of the dynamical system defining a single cell.

Types of coupling
As already mentioned, there exist in all coupled systems in principle at least two different types of coupling mechanisms, namely local coupling and global coupling mechanisms.

- The notion of global coupling indicates, that in a dynamical system with such a coupling, each cell is coupled with all other cells in the lattice. In the case of a global coupling (see Fig. 3.7(a)), one gets from (3.9) and (3.10):

$$\underline{s}^i_{n+1} = \underline{f}\left(\underline{s}^1_n, \dots, \underline{s}^M_n, \underline{p}\right) \tag{3.11}$$

with

$$\underline{f} : \Gamma_s^M \times \Gamma_p \mapsto \Gamma_s$$

Physically or technically global coupling mechanisms are caused by ***long range interactions***.

- ▶ The notion of local coupling indicates, that in a dynamical system with such a coupling, each cell is coupled only with cells in the lattice which are in the neighborhood. This leads immediately to the necessity of the definition of this neighborhood. Concerning one-dimensional CMLs, one often defines this neighborhood symmetrically and not too large. Typical ranges for instance are one or two cells in each direction. The physical or technical interpretation in this case is, that the range of the interaction is $r = 1$ or $r = 2$. If the range of the interaction is r and the coupling symmetrically, then each cell is coupled with $K = 2r+1$ adjacent cells of the lattice. Like in other spatial inhomogeneous systems, also in CMLs there exist the problem of the boundary values, that is the cells with indices $i \le r$ and $i > M - r$. Depending on the considered problem, one often chooses here either a ***ring-like topology*** (see Fig. 3.7(b)), where the boundary cells are connected with each other, or a ***chain-like topology*** (see Fig. 3.7(c)), where the boundary cells need a special treatment.

Properties
In principle, coupled map lattices are still ordinary maps, with a specific structure of the vector field or system function and usually a huge number of state components. The advantage of the formulation as a CML is, that in this case the vector field or system function is much more easy to implement as in the case of an ordinary map, because the formulation as a CML is adapted to the specific structure of such maps. This formulation occurs often in physical or technical systems, consisting of many identical subsystems. In this sense, coupled map lattices are suitable or adequate mathematical models (see Sec. 1.2.2) of systems with such a specific structure.

The most important property of this class of dynamical systems is, that in their dynamic behavior the emergence of ***spatio-temporal structures*** or ***spatio-temporal patterns*** can be observed. These structures or patterns emerge as a consequence of the coupling mechanisms and because they are not imposed, this is some kind of ***self-organization***. This is especially an interesting phenomenon when long range structures or patterns emerge in systems with only ***short range interactions*** and hence local coupling mechanisms. Due to their relative simplicity, compared with many real coupled systems, coupled map lattices are ideal test systems to discover and to

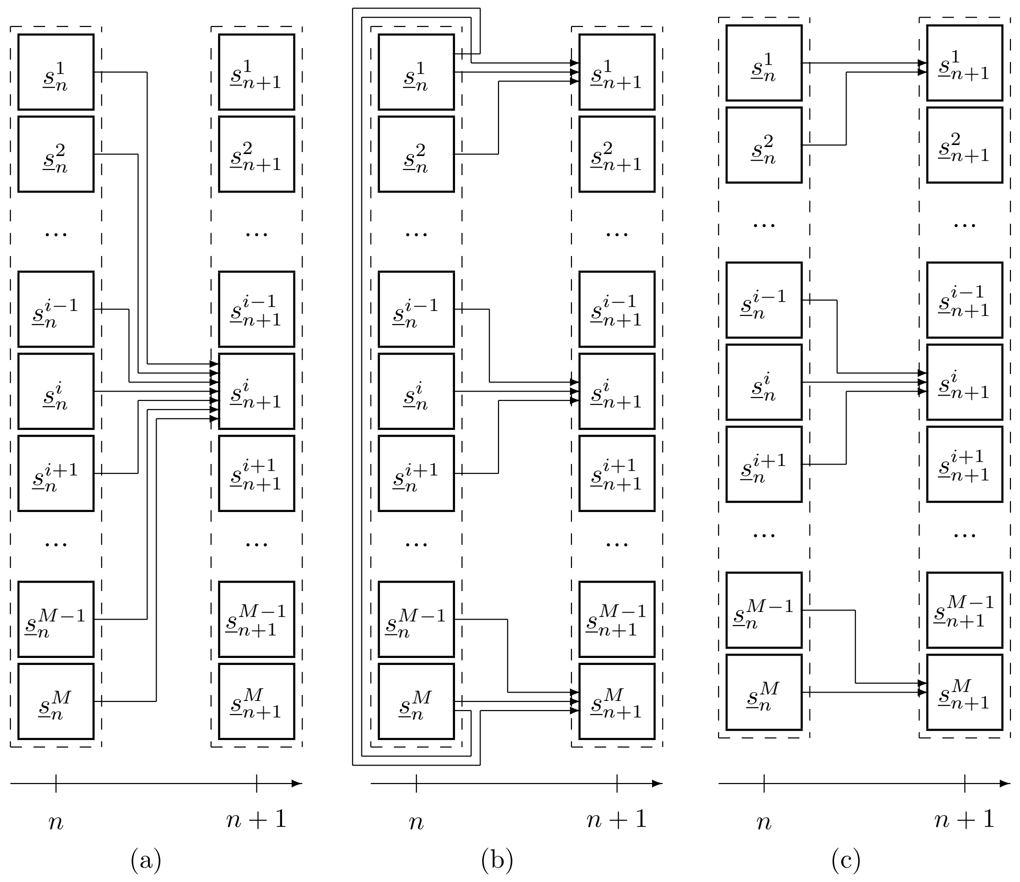


Figure 3.7: CML: coupling mechanisms

(a) global coupling
(b) local coupling with ring like topology
(c) local coupling with chain like topology

In the figures (a)-(c), two time steps n and $n+1$ are shown. The arrows illustrate how the state of a cell at time step $n+1$ depends on the states of the cells at time n. For reasons of a better presentation, only the relevant couplings are shown.

investigate the general ***structure formation principles*** in coupled systems, based on self-organization mechanisms. As an example of this class, the following system is considered:

The plone map CML

Definition: ***Plone map CML***

$$x_{n+1}^i = f(\varkappa_n^i, a) \quad \text{with} \quad f(x,a) = \begin{cases} x+a & \text{falls} \quad x<1 \\ 0 & \text{falls} \quad x \geq 1 \end{cases} \tag{3.12}$$

and

$$\varkappa_n^i = \frac{\gamma_1 x_n^{1+((i-2) \bmod M)} + \gamma_2 x_n^i + \gamma_3 x_n^{1+(i \bmod M)}}{\gamma_1+\gamma_2+\gamma_3}$$

$$x_n^i \in [0, 1+a)\,, \quad i = 1,\ldots,M\,, \quad \underline{p} = (a, \gamma_1, \gamma_2, \gamma_3) \in \mathbb{R}^4$$

■

As shown in figure 3.8, this system shows at the given parameter setting a stable asymptotic dynamic behavior, which is spatially inhomogeneous but periodic in time. As one can see in the middle part of the figure, a ring like or cyclic topology is used. In the transient regime at the beginning, the phenomenon of a glider can be observed. The emergence of this phenomenon, typical for cellular automata, is not so astonishing, when looking at the system function. The piecewise definition, together with the constant increment a and the used coupling cause the system to have also only a finite number of quasi-discrete states comparable to the discrete states used in cellular automata.

3.3.2 Systems with memory

3.3.2.1 Recurrent maps (RMaps)

This class of dynamical systems has some kind of "memory", because the defining transition depends not only on the state of the system at the current

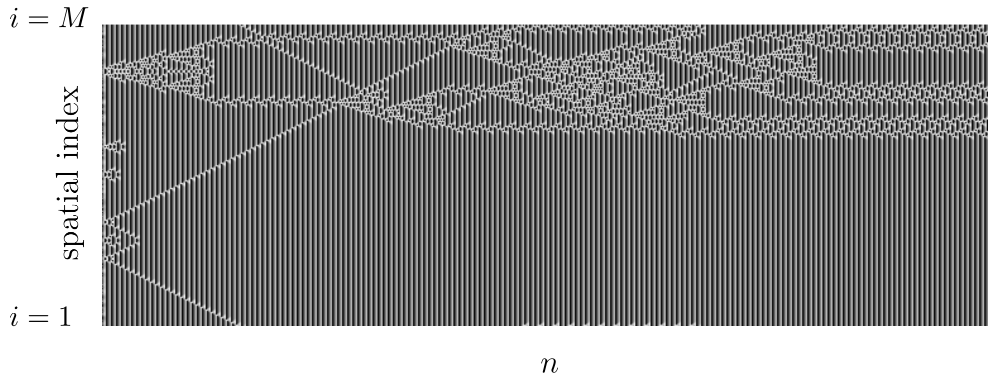


Figure 3.8: CML: emergence of spatio-temporal structures
Starting with a random initialization, system (3.12) shows after a transient behavior an asymptotic dynamics, which is spatially inhomogeneous but periodic in time. In the transient region, the phenomenon of a glider, known from the field of cellular automata, can be observed. The gray-scale values represent the values of the individual state variables $x_n^i \in [0, 1 + a)$. *Parameter setting:* $M = 238$, $a = 0.36$ *and* $\gamma_1 = \gamma_2 = \gamma_3 = 1$.

time t_n, but also on states of the system in the past. Hence this class is defined by the following recurrence equation:

Definition: ***Recurrent map***

$$\underline{s}(t_{n+1}) \quad = \quad \underline{f}\left(\underline{s}(t_n), \underline{s}(t_{n-1}), \ldots, \underline{s}(t_{n-m}), \underline{p}\right) \tag{3.13}$$

with

$$\underline{f} : \Gamma_s^{m+1} \times \Gamma_p \mapsto \Gamma_s$$

■

The number $m + 1$ is denoted here as ***recurrence level***. That means, that the new state $\underline{s}(t_{n+1})$ depends on the current state $\underline{s}(t_n)$ and additionally on the m states $\underline{s}(t_{n-k})$ with $k \in \{1, 2, \ldots, m\}$ in the past. Accordingly, ordinary maps as defined by (3.1 or 3.2) in section 3.3.1.1 have a recurrence level of one.
Remark:

As already mentioned, CMLs can be converted to corresponding ordinary maps by an extension of the state space. Introducing the $m+1$ new states $\underline{s}_i(t_n)$ with $i \in [0, 1, \dots, m]$ by:

$$\begin{array}{rcl} \underline{s}_0(t_{n+1}) & = & \underline{f}\left(\underline{s}_0(t_n), \underline{s}_1(t_n), \dots, \underline{s}_m(t_n), \underline{p}\right) \\ \underline{s}_1(t_{n+1}) & = & \underline{s}_0(t_n) \\ \vdots & \vdots & \vdots \\ \underline{s}_{m-1}(t_{n+1}) & = & \underline{s}_{m-2}(t_n) \\ \underline{s}_m(t_{n+1}) & = & \underline{s}_{m-1}(t_n) \end{array} \tag{3.14}$$

and identifying $\underline{s}_0(t_n)$ with $\underline{s}(t_n)$, then the ordinary map (3.14) is equivalent to the recurrent map (3.13). The ***Bogdanov map*** (see for instance [168]) may be considered as an example of such an equivalence.

The Bogdanov map

Definition: ***Bogdanov map (1)***

$$\begin{array}{rcl} x_{n+1} & = & x_n + y_{n+1} \\ y_{n+1} & = & g(x_n, y_n) = kx_n(1-x_n) + (1+\epsilon+\mu x_n)y_n \end{array} \tag{3.15}$$

■

This definition of the system has state space dimension of two and a recurrence level of one. It represents an ordinary map and is totally equivalent with the following recurrence equation defining a one-dimensional recurrent map with recurrence level two:

Definition: ***Bogdanov map (2)***

$$\begin{array}{rcl} x_{n+1} & = & x_n + g(x_n, x_n - x_{n-1}) \\ & = & x_n + kx_n(1-x_n) + (1+\epsilon+\mu x_n)(x_n - x_{n-1}) \end{array} \tag{3.16}$$

■

Although such a conversion or transformation is always possible, the class of recurrent maps is supported by the **AnT 4.669** software package for convenience purposes. The Bogdanov map has been studied in detail for two important reasons:

1. The original continuous system, which is denoted as ***Bogdanov vector field***, is a good example of the behaviour to be expected in averaged equations. It has some generality as a typical example of an ordinary differential equation with a sink that undergoes a Hopf bifurcation. As the attracting limit cycle grows, it undergoes at a certain parameter value a ***saddle connection bifurcation*** with a remote saddle, whereby the former stable limit cycle is destroyed. The Bogdanov map correspondes to a Poincaré section of the Bogdanov vector field and it was expected, that it is much easier to investigate.

2. The Bogdanov map provides a good model for Poincaré maps of periodically forced oscillators. Like the Bogdanov vector field, it has a Hopf bifurcation, which is in the case of dynamical systems discrete in time denoted as ***Neimark-Sacker bifurcation***. However, in the map the dynamics on the invariant limit cycle is far away from being trivial. The invariant circle experiences mode locking with the infinity of Arnold tongues formed in the Neimark-Sacker bifurcation.

In figure 3.9, two typical periodic solutions after the Neimark-Sacker bifurcation are shown.

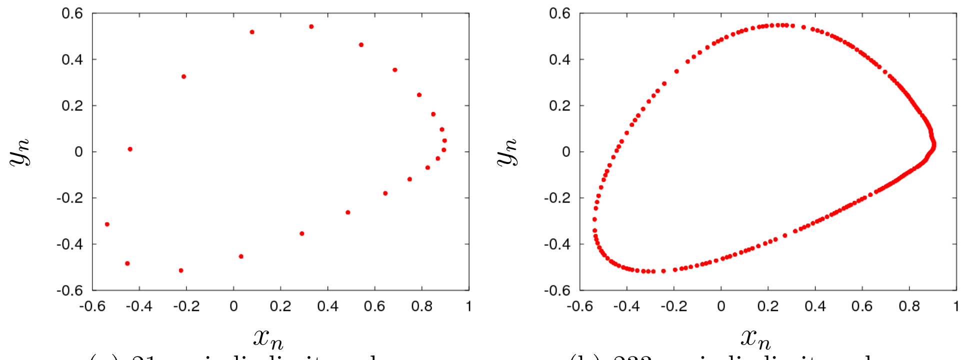
3.4 Dynamical systems continuous in time

In the following sections the very important classes of dynamical systems continuous in time are presented. These classes are mostly used as suitable and adequate mathematical models for many dynamic processes in natural and technical systems which are in most of the cases continuous processes.

Definition: *Dynamical system continuous in time*

A dynamical system whose time evolution is given in such a way, that the transition defines an infinitesimal small time step. ■

A direct consequence of this definition is, that any finite time evolution of such a system requires an integration of the defining transition. Therefore,



(a) 21-periodic limit cycle (b) 233-periodic limit cycle

Figure 3.9: Bogdanov map: periodic solutions

Shown are typical periodic solutions of this map after the Neimark-Sacker bifurcation.Parameter settings:

(a) $k = 0.8, \mu = -0.1, \epsilon = 0.01407$

(b) $k = 0.8, \mu = -0.1, \epsilon = 0.0141264$

numerical simulation packages dealing with such systems have to provide some numerical integration methods often simply denoted as integrators or solvers.

3.4.1 Systems without memory

3.4.1.1 Ordinary Differential Equations (ODEs)

Definition: ***Ordinary differential equation***

$$\frac{d}{dt}\underline{s}(t) \quad = \quad \underline{f}\left(\underline{s}(t), \underline{p}\right) \tag{3.17}$$

with

$$\underline{f} : \Gamma_s \times \Gamma_p \mapsto \Gamma_s$$

■

Like in the case of dynamical systems discrete in time $\underline{s}(t) \in \Gamma_s$ is the dynamic state vector of the system, $t \in \mathbb{R}_0^+$ the continuous time, $\underline{f} : \Gamma_s \times \Gamma_p \mapsto \Gamma_s$ a

usually nonlinear vector field representing the interactions between the single components of the state vector and $\underline{p} \in \Gamma_p$ a vector or set of parameters.
The most famous example of this class is the Lorenz system [97]. During the simulation and investigation of this system, which was derived in 1963 by Lorenz as a mathematical model describing certain weather processes, the phenomenon of deterministic chaos was re-discovered. Re-discovered, because already Poincaré (see [185]) knew at the beginning of the 20th century about solutions of ordinary differential equations which are neither stationary, nor periodic or quasi-periodic, He discovered this dynamic behavior, when he treated the ***three body problem*** [189].

The Lorenz system

Definition: ***Lorenz system***

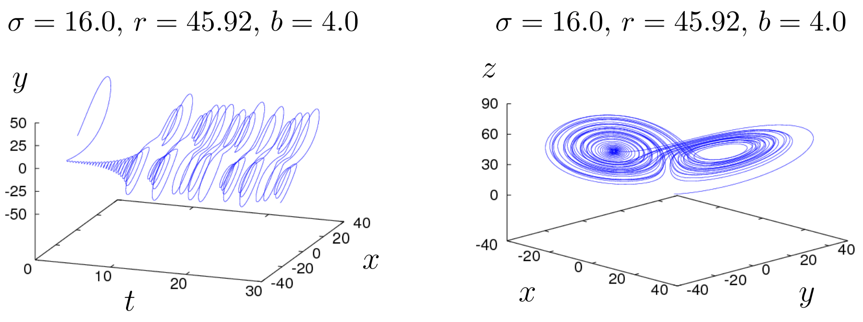
$$\frac{d}{dt}\underline{s}(t) = \underline{f}(\underline{s}(t), \underline{p}) \qquad \begin{aligned} &\underline{s}(t) = (x(t), y(t), z(t))^T \in \mathbb{R}^3 \\ &\underline{p} = (\sigma, r, b) \in \mathbb{R}^3 \\ &\underline{f}(x, y, z, \underline{p}) = \begin{pmatrix} \sigma(y-x) \\ rx - y - xz \\ -bz + xy \end{pmatrix} \end{aligned} \tag{3.18}$$

■

In figure 3.10 a chaotic attractor is shown, which is often denoted as the ***Lorenz attractor***. It is worth mentioning, that this notion is misleading for two reasons:

1. At least for each parameter setting there exist a separate attractor, therefore one can not speak about the Lorenz attractor, although the shape and geometric structure of these attractors are very similar over large ranges of the parameters, especially the most important parameter r.

2. The Lorenz equations (3.18) posses a specific symmetry with respect to the state variables x and y, namely:



(a) chaotic attractor (b) chaotic attractor

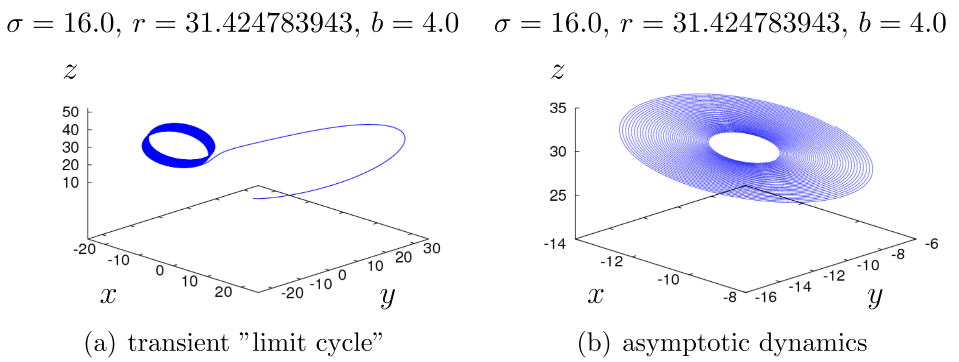
Figure 3.10: Lorenz system: chaotic attractor attractor
In (a), a chaotic attractor of the Lorenz system in a three dimensional projection of the four dimensional extended state space is shown, whereas in (b) the same chaotic attractor is shown in the three dimensional state space.

$$\begin{aligned} x &\leftrightarrow -x \\ y &\leftrightarrow -y \\ z &\leftrightarrow z \end{aligned} \tag{3.19}$$

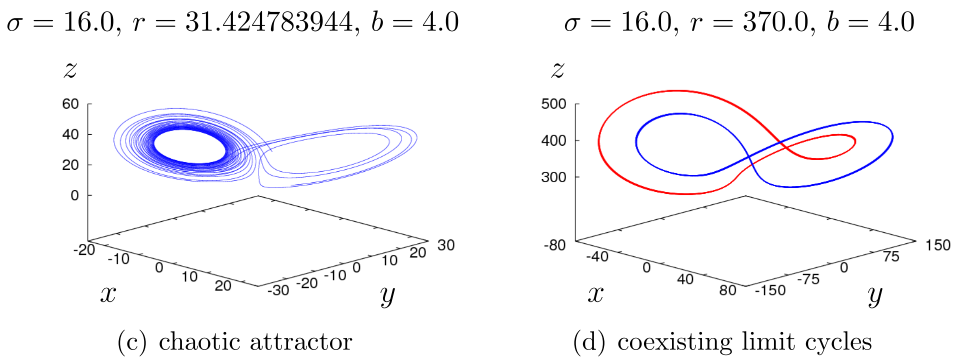
> Symmetry properties like (3.19), usually lead to the emergence of either unique symmetric attractors or coexisting attractors symmetric to each other. Typically there exist then ***symmetry-breaking*** and ***symmetry-recovering*** bifurcations which lead from one regime to the other and vice versa.

The Lorenz system is probably the most investigated dynamical system of this class (see for instance [151]). It is furthermore interesting, because it represents an approximative mathematical model not only in hydrodynamics, as already mentioned, but also in laser physics (see [72, 73, 74]). Similar to the logistic map (see Sec. 3.3.1.1) it has become a paradigm in nonlinear dynamics. It is often used to illustrate the notions of deterministic chaos, ***butterfly effect***, chaotic attractor, strange attractor, the ***concept of Lyapunov exponents*** and much more.

In chapter 11 further examples of the class of ordinary differential equations are presented. They are all used to test and to validate the simulation, investigation and scan capabilities of the **AnT computation engine**, because



(a) transient "limit cycle" (b) asymptotic dynamics



(c) chaotic attractor (d) coexisting limit cycles

Figure 3.11: Lorenz system: several attractors

In (a) and (b) the same parameter setting and initial values were used. The situation differs only in the number of iterations performed. In (a) the transient behavior is shown, which seems to be the dynamic on a limit cycle, but is not. In (b) more iterations were performed, so that the system leaves the transient "limit cycle" and finally converges to the asymptotic stable fixed point. In (c) the parameter r is slightly changed in the **9th** *position after decimal point compared with the setting in (a) and (b). As one can see, the dynamic behavior is chaotic. This happens due to the fact, that at approximately this value of the parameter r, the transition from a fixed point to a chaotic attractor takes place. In (d) two coexisting limit cycles are shown, which are symmetric to each other with respect to the symmetry (3.19).*

they are all well-known in the field of nonlinear dynamics and hence more or less well investigated. The knowledge about these systems is used to cross-check whether the **AnT computation engine** works properly and produces the correct and consistent results under various conditions. Example attractors of some of these systems are visualized in chapter 9 as well.

3.4.1.2 Coupled ordinary differential equation lattices (CODELs)

These class of coupled ordinary differential equation lattices is the time continuous analog to the coupled map lattices (see Sec. 3.3.1.2). Like in the case of the coupled map lattices, the notion is based on the fact, that these systems are build up from ordinary differential equations which define a single cell and that these cells then are arranged on a lattice and represent together a new entity. The single cells are coupled with each other according to local or global coupling mechanisms involving so-called coupling parameters. Again as in the case of the coupled map lattices, the lattice often represents a spatial structure or arrangement and coupled ordinary differential equation lattices can be considered as some kind of spatial inhomogeneous systems like partial differential equations.
Remark:
In fact, the way, how coupled ordinary differential equation lattices are treated numerically is nearly the same as in the case of partial differential equations. This is due to the fact, that the numerical solution of partial differential equations requires a spatial grid or lattice. Hence these systems are then no longer continuous in space, but they are defined on a discrete spatial grid. This spatial discretization here is totally comparable with the discretization of time which lead to the fact, that the solution of a differential equation is numerically done by a map. The discretization in time requires the numerical integration, which can be done in several ways leading immediately to different integration schemes. The discretization in space requires the numerical differentiation in space, which can also be done in several ways leading to several differentiation schemes. These schemes represent then different coupling mechanisms of the cells o the grid or lattice. Summarizing, the only difference between coupled ordinary differential equation lattices and partial differential equations are the type and the origin of the coupling mechanism.
The remarks about CMLs on page 59 hold also for coupled ordinary differential equation lattices, which are defined by:

Definition: ***Coupled ordinary differential equation lattice (CODEL)***

$$\frac{d}{dt}\underline{S}(t) \;=\; \underline{F}\left(\underline{S}(t), \underline{p}\right) \tag{3.20}$$

with

$$\begin{aligned}
&\underline{S}(t) = \left(\underline{s}^1(t), \underline{s}^2(t), \ldots, \underline{s}^M(t)\right)^T \in \Gamma_s^M \\
&\underline{s}^i(t) \in \Gamma_s\,, \quad i = 1, \ldots, M \\
&\underline{F} : \Gamma_s^M \times \Gamma_p \mapsto \Gamma_s^M = \left(\underbrace{\underline{f}, \underline{f}, \ldots, \underline{f}}_{M \text{ times}}\right)^T \\
&\underline{f} : \Gamma_s^K \times \Gamma_p \mapsto \Gamma_s
\end{aligned}$$

■

The equation of motion of the i-th cell of the coupled ordinary differential equation lattice is defined by:

Definition: ***CODEL: single cell***

$$\underline{s}^i(t) \;=\; \underline{f}\left(\{\underline{s}^{i_k}(t)\}, \underline{p}\right) \tag{3.21}$$

with

$$i_k \in \{1, 2, \ldots, M\}\,, \quad k = 1, \ldots, K \quad \text{and} \quad \underline{f} : \Gamma_s^K \times \Gamma_p \mapsto \Gamma_s$$

■

K is here the total number of cells in the set $\{\underline{s}^{i_k}(t)\}$ with $k = 1, \ldots, K$, which have an influence via the coupling mechanism on the considered cell i. In the following two interesting examples of this class are presented, namely the ***selection equations*** and the ***coupled selection equations***, which have both a global coupling mechanism.

The equation of motion for a single cell of the selection equations is defined by:

Selection equations

Definition: ***Selection equations***

$$\frac{d}{dt}s^i(t) \quad = \quad f(\{s^k(t)\},\beta) \tag{3.22}$$

with

$$\begin{aligned} s^i(t),\beta &\in \mathbb{R}\,, \quad i=1,\ldots,M \\ f(\{s^k(t)\},\beta) &= s^i(t)\left(1+(\beta-1)\left(s^i(t)\right)^2-\beta\sum_{k=1}^{M}\left(s^k(t)\right)^2\right) \end{aligned}$$

■

The dynamic behavior of the selection equations (3.22) has the interesting property, that the state component or cell which was initialized with the largest initial value wins a ***competition process*** or ***selection process*** and converges asymptotically to the value 1, whereas all other state components converge asymptotically to 0. The competition process is illustrated in figure 3.12.
Another example of this class based on the selection equations (3.22) are the so-called ***coupled selection equations***. This CODEL is very interesting, because it can be used to solve multi-index ***assignment problems*** known from the field of ***combinatorial optimization*** [152, 75, 156]. The assignment searched for, is determined by the asymptotic values of this specifically constructed dynamical system, whereby the competition process accounts for the optimization. Assignment problems can be used to model some problems in flexible manufacturing systems (FMS), distributed autonomous robotic systems [155, 91, 147], and cellular robotic systems (CEBOT) [50, 51, 52, 153]. For details and further references about assignment problems, it is referred to [154, 75, 156].

The following dynamical system can be used to find solutions to the NP-hard three -index assignment problem, where an assignment of the elements of three different sets has to be found in such a way, that to each element of the first set one and only one element of the second set is assigned to

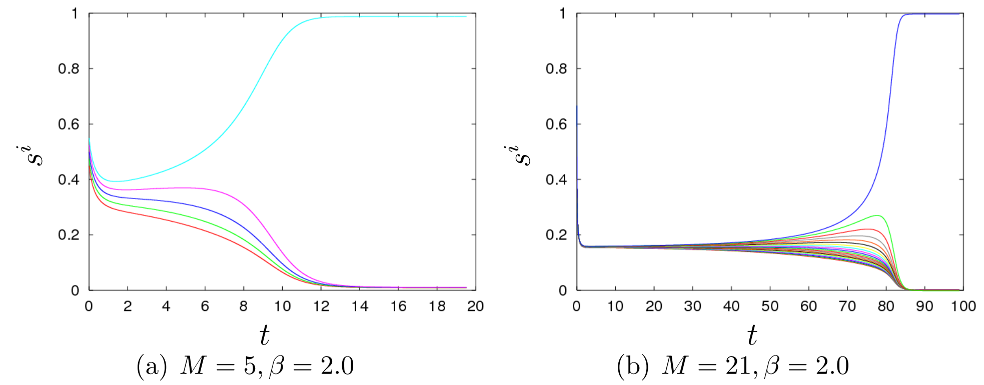


Figure 3.12: Selection equations: time series for different lattice sizes M
The competition or selection process between the single state components or cell states can be clearly observed. Finally, the state component with the largest initial value wins the competition process and converges asymptotically to the value 1. All other state components converge asymptotically to the value 0.

and that to this sub-assignment one and only one element of the third set is assigned to.

Definition: ***Three-index assignment problem***

In the three-index assignment problem, the Boolean variables $x_{ijk} \in \{0,1\}$ with $i, j, k \in \{1, ..., n\}$ have to be determined such that the total costs

$$c := \sum_{i,j,k} c_{ijk} \cdot x_{ijk} \tag{3.23}$$

are minimal with respect to the constraints

$$\sum_{i,j} x_{ijk} = 1 \quad \forall k \in \{1, ..., n\}, \tag{3.24}$$

$$\sum_{i,k} x_{ijk} = 1 \quad \forall j \in \{1, ..., n\}, \tag{3.25}$$

$$\sum_{j,k} x_{ijk} = 1 \quad \forall i \in \{1, ..., n\}, . \tag{3.26}$$

■

The three-index assignment problem is shown to be $\mathcal{NP}$-hard [54]. The CODEL, which can be used to solve the three-index assignment problem is defined by:

Coupled selection equations

Definition: ***Coupled selection equations***

$$\frac{d}{dt}s^{ijk}(t) \quad = \quad f(\{s^{i'j'k'}(t)\},\beta) \tag{3.27}$$

with

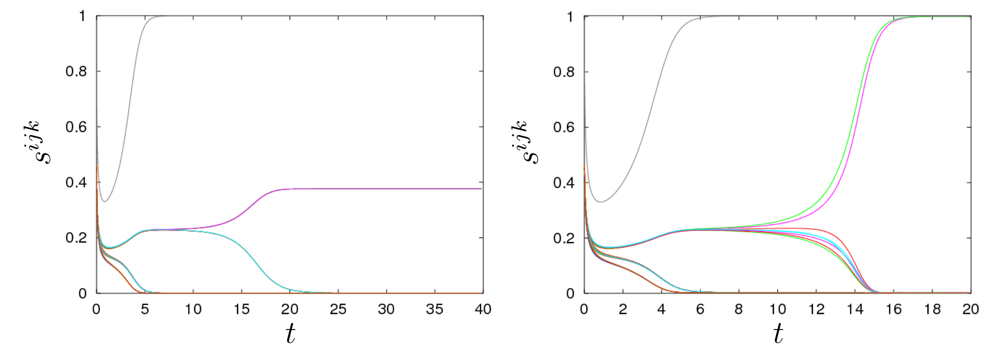
$$s^{ijk}(t),\beta \quad \in \quad \mathbb{R}\,, \quad i,j,k \in \{1,\ldots,M\}$$

$$f(\{s^{i'j'k'}(t)\},\beta) = s^{ijk}(t)\Bigg(1+(3\beta-1)\left(s^{ijk}(t)\right)^2 - \beta\sum_{j'=1}^{M}\sum_{k'=1}^{M}\left(s^{ij'k'}(t)\right)^2 \\ -\beta\sum_{i'=1}^{M}\sum_{k'=1}^{M}\left(s^{i'jk'}(t)\right)^2 - \beta\sum_{i'=1}^{M}\sum_{j'=1}^{M}\left(s^{i'j'k}(t)\right)^2\Bigg)$$

■

In this case, there are three indexable quantities, which correspond to the three-index assignment problem. Because one can always reduce such a multi-index to only one index, this system is also supported by the **AnT 4.669** software package. The notation with the three indices is used here only for convenience purposes.

The times series in figure 3.13 show results of two simulation runs of the dynamical system corresponding to the three-index assignment problem with problem size $M = 3$ and a specific cost matrix which is used for the initialization of the system. The systems has a state space dimension of $N_s 3^M = 27$. The competition process selects the assignment which minimizes the costs given by the initialization of the dynamical system. The assignment is determined by the state components which asymptotically converge to the value 1. As on can see in figure 3.13(a) the dynamics got stuck on a saddle-point.



(a) without noise (b) with noise

Figure 3.13: Coupled selection equations: numerical solution

Numerical solution to a three-index assignment problem. Parameter setting: $\beta = 2.0$. In (a) the dynamics got stuck on a saddle-point. Although a saddle point is unstable, there is no escape in a finite time because the dynamics slows down. Hence, no feasible solution can emerge. In (b) small additive noise was added to the equation of motion, so that the dynamics can escape from the saddle-point and can finally converge to a feasible solution of the three-index assignment problem.

Although a saddle point is unstable, there is no escape in a finite time because the dynamics slows down. Hence, no feasible solution can emerge. In figure 3.13(b), a small ***additive noise*** was added to the equation of motion, so that the dynamics can escape from the saddle-point and can finally converge to a feasible solution shown in figure 3.14.

3.4.1.3 Partial differential equations (PDEs)

For dynamical systems of this class the state vector $\underline{s}$ is not only a pure function of time t, but also a function of additional independent variables. These additional independent variables correspond often to coordinates describing the underlying physical space. In general, the evolution function of this large class of dynamical systems may be defined by the following compact notation:

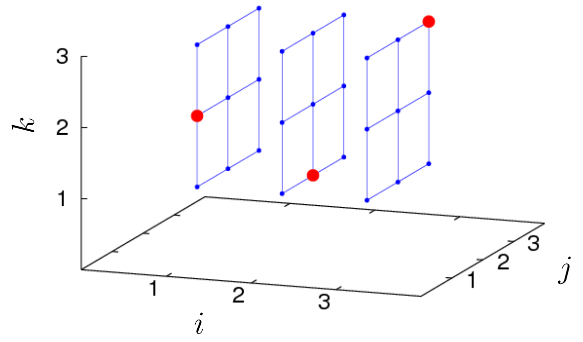


Figure 3.14: Three-index assignment problem: a feasible solution

The three red points are the final assignment corresponding to the three solution curves in figure 3.13(b), which asymptotically converge to the value one. The small blue points correspond to the curves which asymptotically converge to zero. To each element of the first set (index i) one and only one element of the second set (index j) and one and only one element of the third set (index k) is assigned to. Note, the blue mesh has no meaning, it is drawn for representation reasons only.

Definition: ***Partial differential equation (PDE)***

$$\underline{0}=\underline{F}\left(\underline{s}(\underline{x}),\left\{\frac{\partial\underline{s}(\underline{x})}{\partial x_i}\right\},\left\{\frac{\partial^2\underline{s}(\underline{x})}{\partial x_i \partial x_j}\right\},\ldots,\left\{\frac{\partial^k\underline{s}(\underline{x})}{\partial x_0^{k_0}\partial x_1^{k_1},\ldots\partial x_m^{k_m}}\right\},\underline{p}\right) \quad (3.28)$$

$$\text{with} \qquad i,j=0,\ldots,m \quad \text{and} \quad \sum_{l=0}^{m} k_l = k$$

■

Here $\underline{s}(\underline{x}) \in \Gamma_s$ is a state vector of dimension N_s, depending on $m+1$ independent variables, whereby the convention is used, that the 0-th component of the vector $\underline{x}$ represents the time t whereas the remaining m components represent the additional independent variables. The terms in curly brackets denote any possible subset of partial derivatives of the corresponding order. The function $\underline{F} \in \Gamma_s$ represents here the time evolution of the state vector only implicitly.

However, in the cases where the separation of a pure time derivative is possible (see App. D), equation (3.28) may be rewritten as:

$$\frac{\partial}{\partial t}\underline{s}(\underline{x},t) \quad = \quad \underline{f}\left(\underline{s}(\underline{x},t), \frac{\partial \underline{s}(\underline{x},t)}{\partial x_i}, \frac{\partial^2 \underline{s}(\underline{x},t)}{\partial x_i \partial x_j}, \ldots, \underline{p}\right) \tag{3.29}$$

Here the vector field $\underline{f}$ is not only a function of the state $\underline{s}(\underline{x},t)$, but also of its derivatives with respect to the additional independent variables x_i which represent often a spatial dependence of the state vector. Hence, this class of dynamical systems is the adequate mathematical model - in terms of dynamical systems - describing processes which are inhomogeneous in space. Well-known examples of this class of dynamical systems are for instance:

- heat conduction equation [176]
- diffusion equations
- reaction diffusion equations
- wave equation [191]
- Maxwell equations [182]
- Sine-Gordon equation [188]
- Korteweg-de Vries equation [178]
- Klein-Gordon equation [188]
- Schrödinger equation [187]
- Dirac equation [171]
- ...

As a typical example, the heat conduction equation is considered here:

Heat conduction equation

Definition: ***Heat conduction equation***

$$\frac{\partial T(x,t)}{\partial t} \quad = \quad \kappa \frac{\partial^2 T(x,t)}{\partial x^2} \tag{3.30}$$

■

Hereby $T(x,t)$ is a one-dimensional temperature profile, which evolves in the course of time. The parameter κ is connected with the diffusion constant and is denoted as thermal diffusivity. In figure 3.15, the transient behavior of this system is shown, as it relaxes from the temperature profile given at time $t = 0$ to the thermal equilibrium at $t = \infty$. This system is used to test and evaluate the relatively simple PDE solver which is implemented in the **AnT computation engine**, because one can compare the results here with analytical solutions (see for instance [176]).

Currently, only one-dimensional PDEs, which can be described by (3.29) are supported. Furthermore, no adaptive grid method is provided and, as already mentioned only relatively simple ***spatial differential operators*** can be used. As a consequence, the current support of the PDE class is not sufficient for many scientific problems. However, the concept used for instance for the implementation of the spatial differential operators is quite elegant in its usage, as one can see from the example system function for the abovementioned heat conduction equation.

```
#include "AnT.hpp"

#define kappa parameters[0]
#define T (Temperature,cellIndex,0,deltaX)

bool heat_conduction ( const CellularState& Temperature,
                       const Array<real_t>& parameters,
                       int cellIndex,
                       real_t deltaX,
                       StateCell& rhs)
{
  rhs[0] = kappa * D<0,0> T;
  return true;
}

extern "C" { void connectSystem ()
{ PDE_1d_Proxy::systemFunction = heat_conduction; }}
```

Here, the text marked blue is the interface, which is designed to meet the requirements of the PDE class, the green text illustrates, how the system function is connected to the **AnT computation engine** and the red one is the implementation of this specific partial differential equation. Note, that

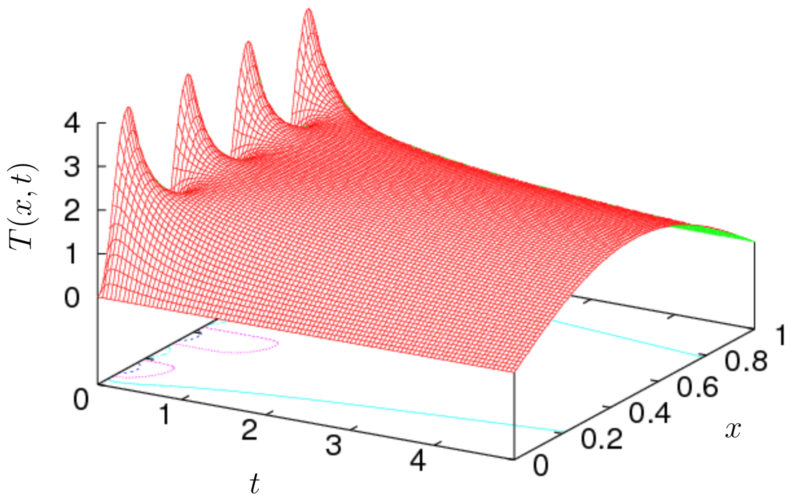


Figure 3.15: Heat conduction equation: numerical solution
Parameter setting: $\kappa = 0.1$. Shown is the relaxation of the temperature profile $T(x, 0)$ at time $t = 0$ to the thermal equilibrium at time $t = \infty$.

the implemented spatial differential operators are already prepared for the case of multi-dimensional PDEs. In the used notation, for instance a mixed partial derivative would look like this:

$$\frac{\partial^3}{\partial x \partial y \partial y} \quad \Rightarrow \quad \texttt{D<0,1,1>}$$

3.4.2 Systems with memory

In many natural and technical systems memory effects are acting and therefore adequate mathematical models continuous in time for dynamic processes with memory effects are required. Depending on the mechanisms leading to the memory effects, one can distinguish between three large subclasses.

3.4.2.1 Delay differential equations (DDEs)

One such subclass are ***delay differential equations***, which are defined by:

Definition: ***Delay differential equation***

$$\frac{d}{dt}\underline{s}(t) \quad = \quad \underline{f}\left(\underline{s}(t), \underline{s}(t-\tau), \underline{p}\right) \tag{3.31}$$

with

$$\underline{f} : \Gamma_s^2 \times \Gamma_p \mapsto \Gamma_s$$

■

Here τ is the so-called ***time delay*** of the system. For dynamical systems belonging to this subclass, the vector field $\underline{f}$ depends not only on the state vector $\underline{s}(t)$ at time t but also on exactly one state vector $\underline{s}(t-\tau)$ at time $t-\tau$ in the past. This characteristic dependence on exactly one state vector in the past represents so-called singular memory effects of the dynamic processes which can be adequately modeled by delay differential equations.

A phase locked loop (PLL) with time delay

A typical example of such a delay differential equation is the ***phase locked loop (PLL)*** with time delay. The Experimental set-up for the electronic system of a first-order phase-locked loop is shown in Fig. 3.16. In many applications, the PLL serves for synchronizing the phases of a reference oscillator and a ***voltage-controlled oscillator (VCO)***. Thereby, the frequency of the output signal of the VCO depends linearly on the input signal. The output signals of both oscillators are multiplied by the aid of a mixer. The induced high-frequency components are then eliminated by a low-pass filter. The resulting signal is fed back to the input of the VCO. A delay line between the VCO and the mixer, implemented analogously or numerically, induces the time delay $\tau \geq 0$. The dynamical variable of interest (see for instance, is the phase difference $\Phi(t) = \Phi_1(t) - \Phi_2(t-\tau)$ between both incoming signals of the mixer. Under quite simple assumptions it becomes possible to derive a nonlinear scalar delay differential equation for this phase difference (see for instance [197], [146]):

$$\frac{d}{dt}\Phi(t) \quad = \quad -K \, \sin[\Phi(t-\tau)] \tag{3.32}$$

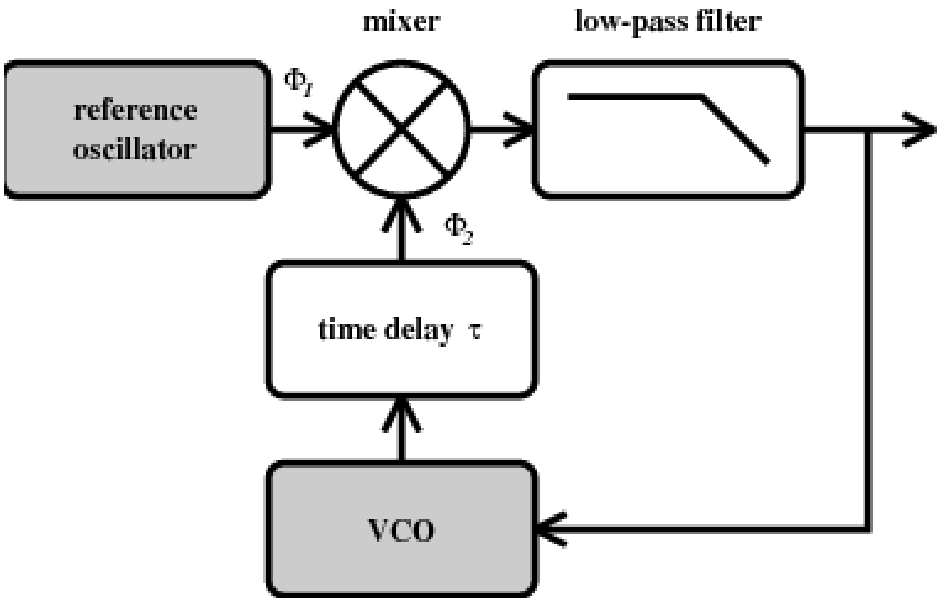


Figure 3.16: PLL with time delay: experimental setup

The parameter $K \geq 0$ denotes the so-called open loop gain of the PLL. Performing an appropriate scaling of the time converts the PLL equation (3.32) to its standard form:

Definition: ***PLL with time delay***

$$\frac{d}{dt}s(t) \quad = \quad -R\,\sin[s(t-1)] \tag{3.33}$$

■

Hereby, the two parameters τ, K in (3.32) are reduced to one effective parameter $R = K\tau$. Thus varying the delay time τ corresponds to changing the parameter R, because in this standard representation of DDEs, the time delay τ is fixed. From the viewpoint of numerics, this fixation of the delay τ to the value 1 has the advantage, that one always works with the same granularity in time, that means with a certain fixed grid on the delay interval. The delay induced changes of the dynamic behavior can still be investigated, by changing the system parameters according to the scaling in time which lead to the standardized representation. In figures 3.17 and 3.18 example attractors of this system, illustrating the rich dynamic behavior of this system, are presented:

3.4.2.2 Multi-delay differential equations (MDDEs)

This subclass has to be used, when the considered dynamic process has memory effects, which are not singular, but which occur at several fixed delay times $t - \tau_i\,, i = 1, \ldots, d$ in the past. The adequate mathematical model is defined by:

Definition: ***Multi-delay differential equation***

$$\frac{d}{dt}\underline{s}(t) \quad = \quad \underline{f}\left(\underline{s}(t), \underline{s}(t-\tau_1), \ldots, \underline{s}(t-\tau_d), \underline{p}\right) \tag{3.34}$$

with

$$\underline{f} : \Gamma_s^{1+d} \times \Gamma_p \mapsto \Gamma_s$$

■

Here, the vector field $\underline{f}$ depends not only on the state vector $\underline{s}(t)$ at time t but in addition also on the state vectors $\underline{s}(t - \tau_i)\, i = 1, \ldots, d$ at times $t - \tau_i$ in the past.

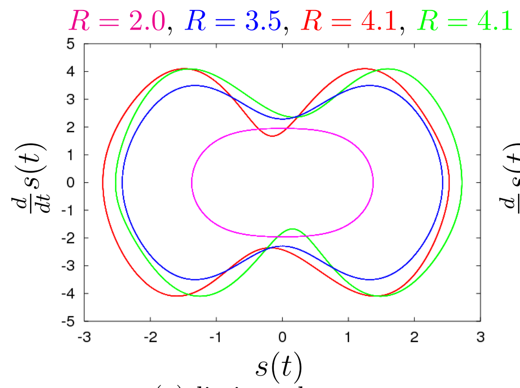
An example of that class is the following model of two coupled neurons with time delayed connections as suggested in for instance [149]. This system is defined by:

Coupled neurons with multiple time delays

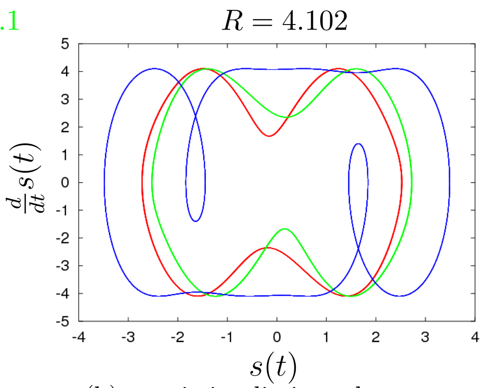
Definition: ***Coupled neurons with multiple time delays***

$$\begin{aligned}\frac{d}{dt}s_1(t) &= -\kappa s_1(t) + \beta \tanh[s_1(t-\tau_s)] + a_{12} \tanh[s_2(t-\tau_2)] \\ \frac{d}{dt}s_2(t) &= -\kappa s_2(t) + \beta \tanh[s_2(t-\tau_s)] + a_{21} \tanh[s_1(t-\tau_1)]\end{aligned} \tag{3.35}$$

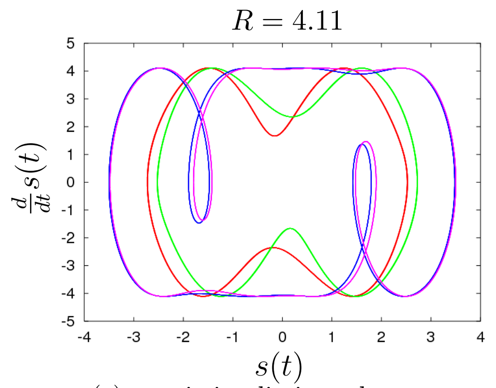
■



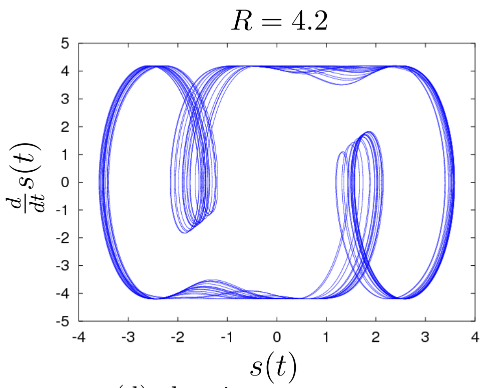
(a) limit cycles



(b) coexisting limit cycles



(c) coexisting limit cycles



(d) chaotic attractor

Figure 3.17: PLL with time delay: example attractors

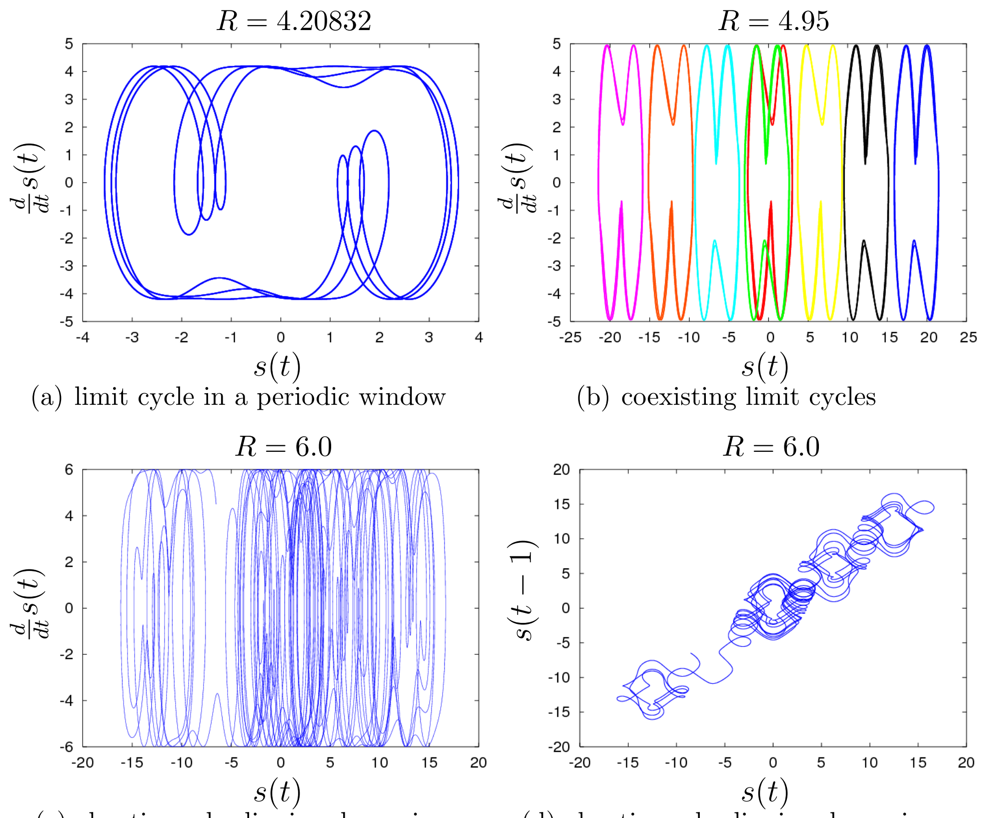
All initializations were done with a constant initial function with value c on the delay interval $[-1, 0]$.

(a) *symmetric limit cycle at $R = 2.0$ and $R = 3.5$, which splits at a symmetry breaking bifurcation into two coexisting limit cycles symmetric to each other plotted at $R = 4.1$ ($c = 1$, $c = -1$)*

(b) *coexisting limit cycles $c = 1$, $c = -1$, $c = \pm 2$*

(c) *coexisting limit cycles $c = 1$, $c = -1$, $c = 2$, $c = -2$*

(d) *chaotic attractor after one of the period doubling scenarios*



(a) limit cycle in a periodic window
(b) coexisting limit cycles
(c) chaotic cycle slipping dynamics
(d) chaotic cycle slipping dynamics

Figure 3.18: PLL with time delay: example attractors

All initializations were done with a constant initial function with value c on the delay interval $[-1, 0]$.

(a) *three-periodic limit cycle within a window in the chaotic regime*
(b) *coexisting limit cycles in different regions in the state space*
$c = -2.5$, $c = -2$, $c = -1.5$, $c = -1$, $c = 1$, $c = 1.5$, $c = 2$, $c = 2.5$
(c) *phase portrait of the chaotic cycle slipping dynamics*
(d) $(s(t-1)$ *versus* $s(t))$*-plot of the chaotic cycle slipping dynamics*

This system has four ordinary parameters κ, β, a_{12} and a_{21} as well as three time delays τ_1, τ_2 and τ_s. By a scaling of time, one can derive the following standard representation of this system:

$$\begin{aligned} \frac{d}{dt'}s_1'(t') &= -\kappa' s_1'(t') + \beta' \tanh[s_1'(t' - \frac{\tau_s}{\tau_m})] + a_{12}' \tanh[s_2'(t' - \frac{\tau_2}{\tau_m})] \\ \frac{d}{dt'}s_2'(t') &= -\kappa' s_2'(t') + \beta' \tanh[s_2'(t' - \frac{\tau_s}{\tau_m})] + a_{21}' \tanh[s_1'(t' - \frac{\tau_1}{\tau_m})] \end{aligned} \tag{3.36}$$

with

$$\begin{aligned} \tau_m &= \max\{\tau_1, \tau_2, \tau_s\} \\ t' &= \frac{t}{\tau_m} \\ s_i'(t') &= s_i(\tau_m t') \\ \kappa' &= \tau_m \kappa \\ \beta' &= \tau_m \beta \\ a_{12}' &= \tau_m a_{12} \\ a_{21}' &= \tau_m a_{21} \end{aligned}$$

Because τ_m is defined as the maximal delay time, at least one of the delay times $\frac{\tau_1}{\tau_m}, \frac{\tau_2}{\tau_m}, \frac{\tau_s}{\tau_m}$ in the standard representation (3.36) is equal to one. As in the case of a single time delay (compare (3.32) and (3.33)), the number of system parameters could be reduced by one due to the time scaling. Additionally, the numerical integration scheme is standardized to the delay interval $[-1, 0]$. The dynamics of system (3.35) is analytically well investigated in [149], therefore it is used in this work to validate the **AnT 4.669** simulation package for the class of multi-delay differential equations. As one can see, the system posses a symmetry with respect to the origin, namely:

$$\begin{aligned} s_1 &\leftrightarrow -s_1 \\ s_2 &\leftrightarrow -s_2 \end{aligned} \tag{3.37}$$

Due to this symmetry, it is expected, that symmetry-breaking and symmetry-recovering bifurcations occur in this system. This is illustrated in figure 3.19, where some example attractors and the transient behavior corresponding to the initial functions for different values of the system parameter a_{21} are presented. The phase portraits and their symmetry properties shown in figure 3.19 are consistent with the two parameter scans (see Fig. 5.2) using the minimum maximum analysis method presented in section 5.2.2.

3.4.2.3 Functional differential equations (FDEs)

This subclass has to be used, when the memory effects are no longer acting at several fixed delay times, but on a complete interval $[t-\tau, t]$ in the past. This is a further extension and the adequate mathematical model describing such dynamic processes can be defined by:

Definition: *Functional differential equation*

$$\frac{d}{dt}\underline{s}(t) \quad = \quad \underline{\mathcal{F}}\left(\left[\underline{s}_t\right], \underline{p}\right)(0) \tag{3.38}$$

with

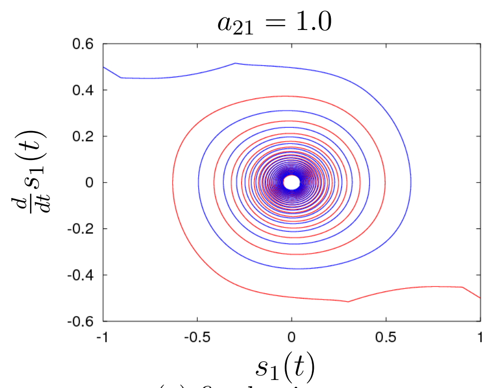
$$\underline{s}_t :\in \Gamma_{\mathcal{C}}^{N_s}\,,\text{and} \quad \mathcal{C} = [-\tau, 0]$$
$$\underline{\mathcal{F}} : \Gamma_{\mathcal{C}} \times \Gamma_p \mapsto \Gamma_s$$

■

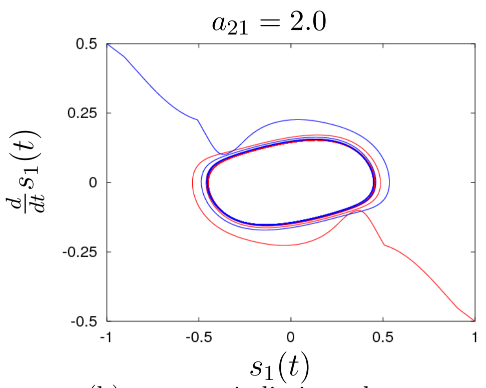
For dynamical systems of this class the vector field $\underline{\mathcal{F}}$ is a functional of the extended state vector $\underline{s}_t$ whose components are real valued functions on the interval $[-\tau, 0]$. In contrast to delay differential equations this class of dynamical systems is the adequate mathematical model - in terms of dynamical systems - describing processes with distributed memory effects.

Remarks:

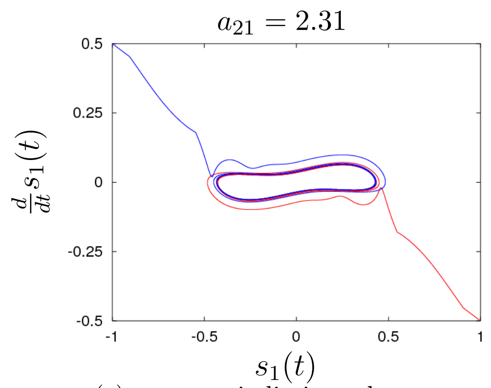
- ▶ The **AnT computation engine** supports all three subclasses of dynamical systems with memory, although only the DDEs and FDEs are implemented. For the DDE class a separate implementation with an interface, which is adapted to the two arguments $\underline{s}(t)$ and $\underline{s}(t-\tau)$ of the vector field in equation (3.31), is provided. Because the MDDEs can in principle have an arbitrary large number of arguments in the vector field corresponding to the in principle arbitrary large number of time delays, this subclass has to be implemented by the user as a



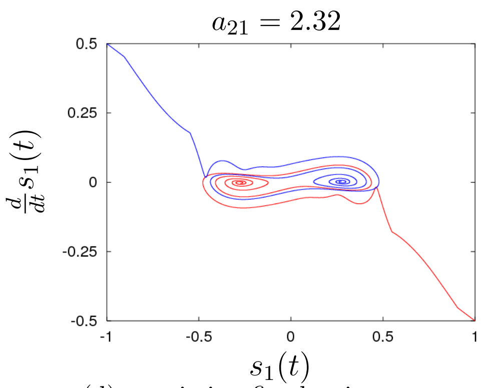
(a) fixed point



(b) symmetric limit cycle



(c) symmetric limit cycle



(d) coexisting fixed points

Figure 3.19: Coupled neurons with time delay: example attractors

All initializations were done with constant initial functions with value c *on the delay interval* $[-\tau_m, 0]$*, namely* $c_1 = c_2 = 1$, $c_1 = c_2 = -1$. *The parameter setting was:* $\kappa = 0.5$, $\beta = -1.0$, $a_{12} = 1.0$, $\tau_s = 1.5$, $\tau_1 = 0.3$, $\tau_2 = 0.2$.

(a) fixed point before the supercritical Hopf bifurcation
(b) symmetric limit cycle after the supercritical Hopf bifurcation
(c) symmetric limit cycle
(d) two coexisting fixed points symmetric to each other

special case of a FDE, which is always possible. The numerical solution of this differential equations causes a discretization of the time interval $[-\tau, 0]$, whereby the granularity is defined by the step size Δt of the corresponding integration method. This holds also for adaptive step size methods, because also in this case the memory interval $[-\tau, 0]$ is represented in the computer memory as an array with sampled values at multiple values of the fixed initial step size.

► At this point one has to remark, that the initial value problem of (3.31), (3.34) and (3.38) is ill-posed in the usual state space Γ_s. This is due to the fact, that an initial value $\underline{s}(t_0) \in \Gamma_s$ at an arbitrary initial time t_0 is not sufficient to calculate the right hand sides of (3.31), (3.34) and 3.38. Therefore the problem has to be reformulated in the extended state space $\mathcal{C}$ of real valued vector functions $\underline{s}_t(\Theta)$, with $\underline{s}_t(\Theta) = \underline{s}(t+\Theta)$ and $-\tau \leq \Theta \leq 0$. For details about this topic see [146] and the references therein. This reformulation has no influence on the numerical solution of DDEs, MDDEs and FDEs, because the representation of the memory interval $[-\tau, 0]$ in the computer memory is already the appropriate way to treat functionals numerically.

3.4.3 Stochastic dynamical systems

In principle, all the different presented classes of dynamical systems are only the deterministic variants and hence have also a stochastic extension. For instance, to describe dynamic processes which are continuous in time and where no memory effects are acting, but which are in some sense noisy, one would use stochastic ODEs as adequate mathematical model. In natural sciences, ***noise*** is often denoted as ***microscopic chaos*** and introduced to model effects caused by systems with a very large number of degrees of freedom, where only a subpart, in which one is interested in, is considered and modeled in detail. The influence of the many subparts not modeled in detail, which are often denoted as ***heat bath***, on the part modeled in detail, is often taken into account only in a summarized or averaged manner as some ***stochastic interaction*** or ***stochastic force***.
The **AnT 4.669** software package currently supports only specific ***noisy systems*** or ***stochastic systems***, namely such with additive noise. That are systems, where in the mathematical model one or several components of the state have an additional additive noise. Hereby the noise can either be a

Gaussian white noise or a ***uniform noise*** which is uniformly distributed on some interval. An example of the class of stochastic ODEs is for instance the ***Ornstein-Uhlenbeck process***, which is mathematically defined by the following ***stochastic differential equation***:

Ornstein-Uhlenbeck process

Definition: ***Ornstein-Uhlenbeck process***

$$d\,\underline{s}_t = -\underline{\underline{M}}\,\underline{s}\,dt + \sigma\,d\,\underline{W}_t \tag{3.39}$$

■

Here, $\underline{\underline{M}}$ and σ are constant and $\underline{W}_t$ denotes the stochastic ***Wiener process***. In figure 3.20, two example trajectories of the three-dimensional Ornstein-Uhlenbeck process with the specific realization given by (3.40) are presented.

$$\sigma = 1 \quad \text{and} \quad \underline{\underline{M}} = \begin{pmatrix} -10^{-4} & 0.1 & -0.2 \\ -0.1 & -10^{-4} & 0.2 \\ 0.5 & -0.5 & -10^{-4} \end{pmatrix} \tag{3.40}$$

3.5 Classification of dynamical systems

From the viewpoint of implementation, the support of the different classes of dynamical systems results in different interfaces and proxies. Thereby, soon the question arises, which classes of dynamical systems share the same interface and hence, how many interfaces can in principle occur. The answer to this question requires a broad classification scheme of all the different classes and, as it turned out, is not so easy to obtain. This is caused by the many degrees of freedom which distinguish the classes. The following list gives an overview about some of the possible degrees of freedom.

A dynamical system can be

- discrete or continuous in time

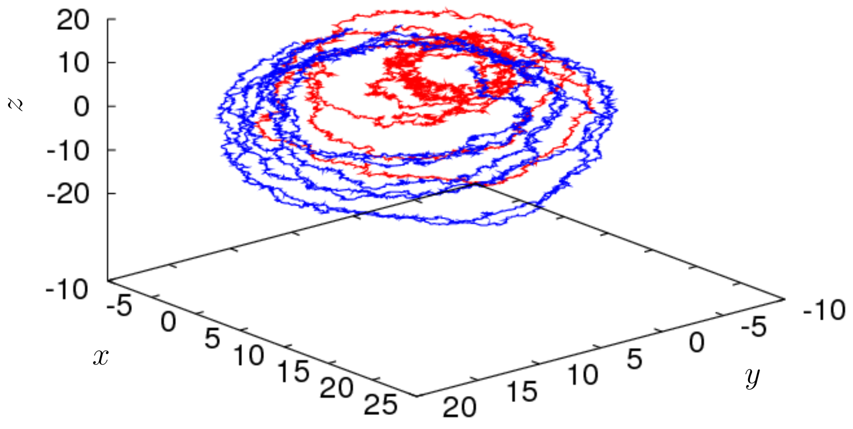


Figure 3.20: Ornstein-Uhlenbeck process: numerical solution
Shown are two numerical solutions of the three-dimensional Ornstein-Uhlenbeck process with the parameter setting given by (3.40).

- ▶ autonomous or non autonomous
- ▶ state discrete or continuous
- ▶ non indexable or indexable (spatial homogeneous or inhomogeneous)
- ▶ with or without memory effects
- ▶ non-hybrid or hybrid
- ▶ stochastic or deterministic
- ▶ explicit or implicit
- ▶ dissipative or conservative

Additionally a dynamical system can include

- ▶ fractional derivatives or ordinary derivatives
- ▶ integrals or not
- ▶ algebraic equations or not

As one can see, a consistent and overall classification scheme is difficult to derive. However, one can focus on that degrees of freedom which are most important or relevant with respect to the aimed goal. Concerning the AnT project, table 3.1 is a first step towards such a classification scheme, which is useful not only in the case, when a new class of dynamical systems has to be supported and the question of the interface to be used arises.

	memory	state	indexable	time	representative
1	-	c	-	c	ODEs
2	-	c	-	d	Maps
3	-	c	c	c	PDEs
4	-	c	c	d	*(not known)*
5	-	c	d	c	CODELs
6	-	c	d	d	CMLs
7	-	d	-	c	discrete component of hybrid ODEs
8	-	d	-	d	finite automata, Petri nets, discrete component of hybrid Maps
9	-	d	c	c	*(not known)*
10	-	d	c	d	*(not known)*
11	-	d	d	c	*(not known)*
12	-	d	d	d	cellular automata
13	+	c	-	c	DDEs (MDDEs, FDEs)
14	+	c	-	d	RMaps
15	+	c	c	c	PDDEs (PMDDEs,PFDEs)
16	+	c	c	d	*(not known)*
17	+	c	d	c	CDDELs (CMDDELs,CFDELS))
18	+	c	d	d	CRMLs
19	+	d	-	c	discrete component of hybrid DDEs (MDDEs,FDEs)
20	+	d	-	d	stack automata discrete component of hybrid RMaps
21	+	d	c	c	*(not known)*
22	+	d	c	d	*(not known)*
23	+	d	d	c	*(not known)*
24	+	d	d	d	cellular automata with memory effects

Table 3.1: Some classes of dynamical systems
legend*:* 'c' *- continuous,* 'd' *- discrete,* '+' *- present,* '-' *not present*

Remarks:

► The large class of hybrid systems, where the state vector $\underline{s}$ has not only continuous, but also some discrete components is, although supported by the **AnT 4.669** software package, not presented here. In principle, each of the classes presented in the previous sections has a hybrid variant. There are two reasons for that:

1. Hybrid systems can be implemented also using the basic classes using
   If *condition* **Then** *statements* [**Else** *else statements*]
   constructs.
2. Hybrid systems are a research field on their own, and beyond the scope of this work.

► There exist further classes of dynamical systems continuous in time, which are not presented here, because they are not supported by the **AnT 4.669** software package. However, they are worth mentioning not only because some of them may be supported in the future. These classes of time continuous dynamical systems are:

- Integro Differential Equations (IDEs)
- Implicit Differential Equations (ImDEs)
- Differential Algebraic Equations (DAEs)
- Fractional Differential Equations (FrDEs)

In this chapter all the classes of dynamical systems whose simulation and investigation is supported by the **AnT 4.669** software package are presented together with some examples of typical attractors or transient dynamic behavior.

Chapter 4

Scanning dynamical systems

4.1 Motivation

As already mentioned in Sec. 3.2 the behavior of a dynamical system depends not only on the state vector $\underline{s}$, but is usually also influenced by some additional ***influencing quantities***. When dealing with dynamical systems, it is obviously, that for instance, system parameters or initial values are such influencing quantities. Often, one is then interested in the ***qualitative changes*** and ***quantitative changes*** of the dynamic behavior of the system with respect to changes of this influencing quantities. In the case of a qualitative change, characteristic properties of the dynamic behavior of the system change, whereas in the case of quantitative changes this is not the case. For instance, local and global bifurcations or crisis of a dynamical system represent qualitative changes, whereas a change in the offset, frequency or amplitude of a periodic solution are typical examples of quantitative changes.

However, sometimes it is also required to investigate the result of an investigation method with respect to the change of a parameter of the investigation method itself, for instance to adjust the investigation method to the specific dynamical system under consideration if necessary. This adjustment of the investigation method is obviously necessarily during the testing phase of a new implemented investigation method or if one has to extend the investigation method for a new supported class of dynamical systems. In all this cases, the parameters of the investigation method may be the influencing quantities.

4.2 Scan procedures

Of course there exist many ways to change the influencing quantities of a dynamical system, but often one is interested in the investigation of the system's behavior at several specific points or even in a complete range or interval of an influencing quantity. The procedure how exactly to do this, that means in which systematic way the change of the influencing quantities is performed, is denoted as ***scan procedure***.

Although it is more or less obvious and common knowledge - at least in the scientific community - what scans are and in which way they have to be performed, in the following two sections definitions of scan procedures are given. Whenever it is referred to a scan in this work, it has to be interpreted according to these definitions.

4.2.1 One-dimensional scans

Of course, the most simple type of scan procedure involves only one influencing quantity. However, because this can be regarded as a special case, the following more general definition covers a larger class of scan procedures.

Definition: ***One-dimensional scan***

The procedure of investigation of a system's behavior - according to some pre-defined rules - in dependence of some influencing quantities at several specific points, such that the sequence of ***scan points*** lie on a curve which can be parameterized by only one parameter, is denoted as ***one-dimensional scan*** or simply ***1D-scan***. ■

This definition is somehow sophisticated, because - as already mentioned - it should not only cover the investigation in dependence of only one influencing quantity at several points or a complete interval but also in dependence of several influencing quantities at once, but in such a way, that it can be parameterized by only one parameter. For instance, the investigation of a system's behavior in a three dimensional parameter space along a straight line in that parameter space can of course be parameterized by one parameter and is hence a one-dimensional scan.

4.2.2 Multi-dimensional scans

The notion of a one-dimensional scan or 1D-scan implies immediately, that the ***scan procedure*** can be expanded to more than one dimension. These ***2D-scans***, ***3D-scans*** or even higher dimensional scans can be defined analogously to the definition of a one-dimensional scan given above.

Definition: *Multi-dimensional scan*

The procedure of investigation of a system's behavior - according to some pre-defined rules - in dependence of some influencing quantities at several specific points, such that this scan points lie in a region which can be parameterized by a set or specifically n parameters, is denoted as ***multi-dimensional scan*** or specifically n***-dimensional scan*** or n***D-scan***.
■

For instance, the investigation of a system's behavior in a two-dimensional region of the parameter space, that is in dependence of two system parameters independently from each other, is denoted as a 2D-scan. Or the investigation of a system's behavior in a two-dimensional region of the parameter space in combination with a two-dimensional region in the initial value space, is denoted as a ***4D-scan***.

4.3 The scan capabilities of AnT 4.669

A simulation system for dynamical systems should provide methods and techniques to change the influencing quantities in an automatic way or manner in order to support a user of the system by its investigations. As a consequence of its architecture (see Chap. 6), the software package **AnT 4.669** supports the scanning of dynamical systems in a quite general way, whereby it provides several specific scan types. However, within the framework of the software, the influencing quantities supported by the **AnT computation engine** are denoted as ***scannable objects***. During the initialization phase, the ***configuration file*** or ***initialization file*** is read and processed and according to the parsed entries not only a list of possible scannable objects is created, but also the current ***scan item sequence***. The scan item sequence is a list of ***scan items*** chosen by the user from the list of possible scannable objects for the current ***scan run***. The number of entries in the scan item sequence defines hereby the dimension of the multi-dimensional scan.

4.3.1 The case of a single scan item

The following sections give an overview about the ***supported scan types*** for a single scan item implemented in the **AnT computation engine**. In contrast to a pure ***simulation run*** - denoted as ***scan mode 0*** where no influencing quantity is changed or varied, this most simple scan run is denoted as ***scan mode 1***. This is due to the fact, that in this case only one influencing quantity is varied or two quantities in such a way, that it can be parameterized by only one scan parameter. In the following the notion real means, that the change of the influencing quantity can be done in principle in a continuous way. Of course, because a computer system is used, this is not possible (see Chap. 10). Instead, the variation or change can only be done in terms of machine numbers according to the chosen floating point number representation (see Sec. 10.2.1). One should always keep this in mind, because the results of a scan run may depend on how the scan is performed exactly, that is for instance depending on whether or not the individual scan points are exact machine numbers or not. The initial value scan of the gingerbread-man map (see Sec. 11.2) demonstrates this dependency drastically.

4.3.2 Real linear scan

In this case, the scan item sequence consists of one scan item, with a scannable object that is varied in real steps according to the chosen floating point precision. The variation is done in a linear manner, that means in equidistant steps on a linear scale.

$$q^k = q^{in} + (k-1)\Delta q\,, \quad k = 1 \dots N^s \tag{4.1}$$

$$\Delta q = \frac{q^{fi} - q^{in}}{N^s - 1} \tag{4.2}$$

Hereby q^k represents the k-th scan point, q^{in} the initial scan point, q^{fi} the final scan point, Δq the real valued scan point increment and N^s the number of scan points.
A simple example may be: $q^k \in [0.1, 0.2, 0.3, 0.4]$

4.3.3 Real logarithmic scan

In this case, the scan item sequence consists of one scan item, with a scannable object that is varied in real steps according to the chosen floating point precision. The variation is done in a logarithmic manner, that means in equidistant steps on a logarithmic scale.

$$\begin{aligned} q^k &= e^{\ln(q^{in})+(k-1)\Delta q_i}\,, \quad k = 1 \dots N^s && (4.3)\\ \Delta q_i &= \frac{\ln(q^{in}) - \ln(q^{fi})}{N^s - 1} && (4.4) \end{aligned}$$

A simple example may be: $q^k \in [0.1, 0.1587401052, 0.2519842100, 0.4]$

4.3.4 Integer linear scan

In this case, the scan item sequence consists of one scan item, with a scannable object that is varied in integer steps. The variation is done in a linear manner, that means in equidistant steps on a linear scale.

$$\begin{aligned} q^k &= q^{in} + (k-1)\Delta q\,, \quad k = 1 \dots N^s && (4.5)\\ \Delta q &= \frac{q^{fi} - q^{in}}{N^s - 1} && (4.6) \end{aligned}$$

Here, the only difference to the real linear scan is, that the scan point increment is integer valued.
A simple example may be: $q^k \in [2, 4, 6, 8]$

4.3.5 Integer logarithmic scan

In this case, the scan item sequence consists of one scan item, with a scannable object that is varied in integer steps. The variation is done in a logarithmic manner, that means in equidistant steps on a logarithmic scale.

$$\begin{aligned} q^k &= e^{\ln(q^{in})+(k-1)\Delta q_i}\,, \quad k = 1 \dots N^s && (4.7)\\ \Delta q_i &= \frac{\ln(q^{in}) - \ln(q^{fi})}{N^s - 1} && (4.8) \end{aligned}$$

Again, the only difference to the real logarithmic scan is, that the scan point increment is integer valued.
Simple examples may be: $q^k \in [10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$, or $q^k \in [1, 2, 4, 8, 16]$

4.3.6 Real linear scan in two dimensions

In this case again, the scan item sequence consists of one scan item, with a scannable object that is varied in real steps according to the chosen floating point precision. The variation is done in a linear manner, that means in equidistant steps on a linear scale. In contrast to the real linear scan, the scan points are now two-dimensional objects.

$$\begin{aligned} \underline{q}^k &= \underline{q}^{in} + (k-1)\Delta\underline{q}\,, \quad k = 1 \ldots N^s && (4.9) \\ \Delta\underline{q} &= \frac{\underline{q}^{fi} - \underline{q}^{in}}{N^s - 1} && (4.10) \end{aligned}$$

Hereby $\underline{q}^k = (q_1, q_2)$ represents the k-th 2D-scan point, $\underline{q}^{in}$ the initial 2D-scan point, $\underline{q}^{fi}$ the final 2D-scan point, $\Delta\underline{q}$ the real valued 2D-scan point increment and N^s again the number of scan points.
A simple example may be: $q^k \in [(0.1, 10.0), (0.2, 5.0), (0.3, 0.0), (0.4, -5.0)]$

4.3.7 Real elliptic scan in two dimensions

In this case, the scan item sequence consists of one scan item, with a scannable object that is varied in real steps according to the chosen floating point precision. The variation is, such that the 2D-scan points lie on an ellipse, whereby the spacing of the 2D-scan points on the ellipse is equidistant, that means that the angle of the corresponding representation in ***polar coordinates*** is varied linearly.

$$\begin{aligned} {q_i}^k &= {q_i}^{off} + a_i \mathrm{trig}_i\left(\varphi^k\right)\,, \quad i = 1, 2\,, \quad k = 1 \ldots N^s && (4.11) \\ \varphi^k &= \varphi^{in} + (k-1)\Delta\varphi && (4.12) \\ \Delta\varphi &= \frac{\varphi^{fi} - \varphi^{in}}{N^s - 1} && (4.13) \\ \varphi^{in} &= \arctan\left(\frac{a_1({q_2}^{in} - {q_2}^{off})}{a_2({q_1}^{in} - {q_1}^{off})}\right) && (4.14) \\ \varphi^{fi} &= \arctan\left(\frac{a_1({q_2}^{fi} - {q_2}^{off})}{a_2({q_1}^{fi} - {q_1}^{off})}\right) && (4.15) \\ & && (4.16) \end{aligned}$$

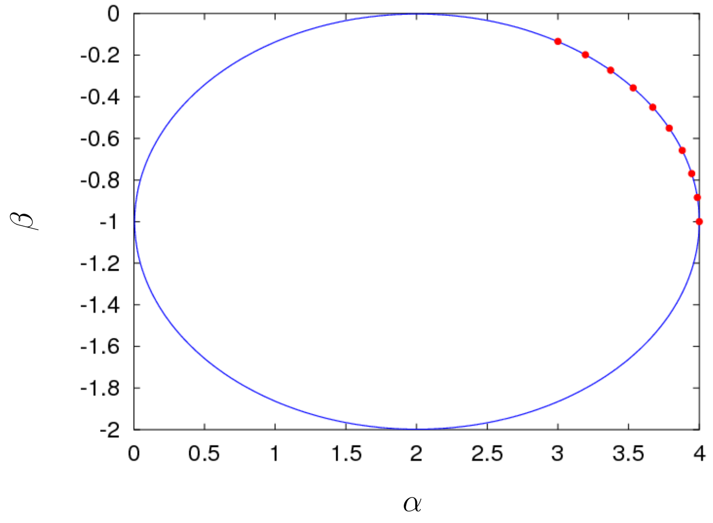


Figure 4.1: Illustration of an elliptic scan.
The figure shows an illustration of the real elliptic scan in two dimensions using a total number of $N^s = 10$ scan points in a two-dimensional parameter space with parameters α and β.

Hereby ${q_i}^k$ represents the i-th component of the k-th 2D-scan point, ${q_i}^{off}$ the offset of the i-th component of the 2D-scan point, a_i the parameter of the scan ellipse, trig_1 the cos function, trig_2 the sin function, φ^k the angle of the k-th 2D-scan point, $\Delta\varphi$ the 2D-scan point angle increment, φ^{in} the angle of the initial 2D-scan point, φ^{fi} the angle of the final 2D-scan point, ${q_i}^{in}$ the i-th component of the initial 2D-scan point, ${q_i}^{fi}$ the i-th component of the final 2D-scan point and N^s again the number of scan points.
Figure 4.1 illustrates this type of scan in a two-dimensional parameter space. The filled circles in the ellipse represent the scan points of a real elliptic scan with a total number of $N^s = 10$ scan points.

4.3.8 User defined scans

The **AnT 4.669** software package supports one further type of scans which can be freely defined by the user. In this case the software reads the scan points from an external file which allows in principle arbitrary scans. Although this option provides the most flexible scan type, it has the disadvantage, that the user has to prepare the file of scan points prior to the scan run.

4.3.9 The case of a scan item sequence

As a consequence of its architecture, the **AnT 4.669** software package allows also scan item sequences and supports hence multi-dimensional scans in a generic way, whereby single scan items are combined to a scan item sequence. As already mentioned, the number of scan items in the scan item sequence defines the scan mode. For instance, if the scan item sequence consists of two scan items, whereby both scan items correspond to system parameters, then the scan mode is two and after the initialization, the **AnT computation engine** performs a 2D-scan in the parameter space. In this way a specific two-dimensional area in the parameter space can be covered by the scan run. A scan item sequence consisting of two scan items corresponding to system parameters and two scan items corresponding to initial values of the dynamical system to be investigated defines a scan mode 4 scan run.

In principle the scan mode is not confined and one can define scan runs with very large scan modes. However, because the performance of computer systems is limited, there exists - depending on the investigated system - a performance limitation in the scan mode. Consider for instance the abovementioned case of a 4D-scan. If the number of scan points for each scan item is only 10, then the total number of scan points is $N^s = 10^4$. If thee are 100 scan points for each scan item, then the total number of scan points is $N^s = 10^8$, which is already a huge number. Especially when the system to be investigated has a large state space dimension, such scan runs are very time-consuming.

Especially for these time-consuming multi-dimensional scan runs, the client/server variant of the **AnT computation engine** was developed. It allows the distribution of such time-consuming calculations among several computing nodes. This allows a parallel execution whereby as much scan points can be treated simultaneously as computing nodes are available.

Chapter 5

Supported investigation methods

5.1 Motivation

In all scientific and engineering disciplines, one is usually not only interested in the pure simulation of dynamical processes, but mainly in the investigation of their dynamic behavior. This leads in principle to a much better understanding of the considered processes and an improved insight in the underlying mechanisms causing the observed dynamics. It is beyond the scope of this work to point out the many advantages of this fact, but some of them are so important, that they should be mentioned here.

- Obviously, the obtained results can often be used to forecast the dynamic behavior of the investigated dynamical process usually for a longer time and more precisely, although it has to be remarked and accentuated here explicitly, that there are basic and principle restrictions concerning this forecasting, especially when dealing with stochastic dynamical systems or such systems which exhibit the phenomenon of deterministic chaos.
- When dealing with technical systems, the obtained knowledge about the dynamic behavior can be used to improve and enhance this systems in a desired way.
- In some cases, the better insight and understanding of the processes leads to the development of improved or even new observation tech-

niques and last but not least to the invention and development of new technical systems.

As already mentioned in Sec. 3.2, only such dynamical processes, which can be adequately modeled as a dynamical system supported by the **AnT 4.669** software package, can be simulated and investigated.

The following list gives an overview about the supported investigation methods currently implemented in the **AnT 4.669** software package.

- General trajectory evaluations
- Period analysis
- Lyapunov exponents analysis
- Region analysis
- Dimension analysis
- Frequency analysis
- Singular value analysis
- Check for conditions
- Symbolic sequence analysis
- Symbolic image analysis
- Calculation of generalized Poincaré sections

Details about the characteristic properties of these methods and example applications are presented in the following sections 5.2 - 5.12.

5.2 General trajectory evaluations

5.2.1 Basic saving

Probably any software tool for simulation of dynamical systems has to support the saving of trajectories. The **AnT 4.669** package provides a large collection of options with respect to this topic:

- The simulated trajectory or orbit can be saved as a time series. For the most classes of dynamical systems (maps, ODEs, DDEs, etc.) this kind of saving means, that the output file has the following format:

column				
1	2	3	...	N_s+1
t_1	$s_1(t_1)$	$s_2(t_1)$	...	$s_{N_s}(t_1)$
t_2	$s_1(t_2)$	$s_2(t_2)$	...	$s_{N_s}(t_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_n	$s_1(t_n)$	$s_2(t_n)$	...	$s_{N_s}(t_n)$

Many figures presented in this work, are based on this saving option. For instance the figures 5.15(a) and 5.15(b), 5.19(a) and 5.19(b), 5.23(a) - 5.23(d) 5.29(a) - 5.29(d) and 5.30(a) - 5.30(d) in this chapter.

- ► For spatial inhomogeneous dynamical systems (CMLs, CODELs, PDEs) it is sometimes more suitable to save the trajectories with explicitely given spatial information. In this case the output file has the format:

column					
1	2	3	4	...	N_s+2
t_1	$r_{\min}$	$s_1(t_1,r_1)$	$s_2(t_1,r_1)$	...	$s_{N_s}(t_1,r_1)$
...	...	...	...	...	...
t_1	$r_{\max}$	$s_1(t_1,r_{\max})$	$s_2(t_1,r_{\max})$	...	$s_{N_s}(t_1,r_{\max})$
t_2	$r_{\min}$	$s_1(t_2,r_1)$	$s_2(t_2,r_1)$	...	$s_{N_s}(t_2,r_1)$
...	...	...	...	...	...
t_2	$r_{\max}$	$s_1(t_2,r_{\max})$	$s_2(t_2,r_{\max})$	...	$s_{N_s}(t_2,r_{\max})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_n	$r_{\min}$	$s_1(t_n,r_1)$	$s_2(t_n,r_1)$	...	$s_{N_s}(t_n,r_1)$
...	...	...	...	...	...
t_n	$r_{\max}$	$s_1(t_n,r_{\max})$	$s_2(t_n,r_{\max})$	...	$s_{N_s}(t_n,r_{\max})$

Here $r \in r_{min}, \dots, r_{\max}$ denotes the position within the spatial domain for PDEs or the cell index for cellular dynamical systems like CMLs or CODELs.

- ► Additionally to the current trajectory one can save the current velocity as well. The format of the output file can be either similar to the format of time series:

column				
1	2	3	$\dots$	N_s+1
t_1	$v_1(t_1)$	$v_2(t_1)$	$\dots$	$v_{N_s}(t_1)$
t_2	$v_1(t_2)$	$v_2(t_2)$	$\dots$	$v_{N_s}(t_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_n	$v_1(t_n)$	$v_2(t_n)$	$\dots$	$v_{N_s}(t_n)$

or more oriented to phase portraits:

column								
1	2	3	$\dots$	N_s+1	N_s+2	N_s+3	$\dots$	$2N_s+1$
t_1	$s_1(t_1)$	$s_2(t_1)$	$\dots$	$s_{N_s}(t_1)$	$v_1(t_1)$	$v_2(t_1)$	$\dots$	$v_{N_s}(t_1)$
t_2	$s_1(t_2)$	$s_2(t_2)$	$\dots$	$s_{N_s}(t_2)$	$v_1(t_2)$	$v_2(t_2)$	$\dots$	$v_{N_s}(t_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_n	$s_1(t_n)$	$s_2(t_n)$	$\dots$	$s_{N_s}(t_n)$	$v_1(t_n)$	$v_2(t_n)$	$\dots$	$v_{N_s}(t_n)$

In both cases $\underline{v}(t)$ denotes the current velocity. For dynamical systems continuous in time, the velocity is given by the right hand side of the equation of motion, $\underline{v}(t) = \frac{d}{dt}\underline{s}(t)$. For dynamical systems discrete in time it is defined as the difference of the current state vector to the previous one, $\underline{v}_n = \underline{s}_n - \underline{s}_{n-1}$.

► In many cases one is more interested in the asymptotic dynamics of the investigated dynamical system. A useful option in this situation is to specify the transient time t_{trans}, so that the trajectories will be saved only after this time. In this case the transient dynamics can be omitted in the saved data. For instance, the trajectory will be saved in the format:

column				
1	2	3	$\dots$	N_s+1
$t_{\text{trans}+1}$	$s_1(t_{\text{trans}+1})$	$s_2(t_{\text{trans}+1})$	$\dots$	$s_{N_s}(t_{\text{trans}+1})$
$t_{\text{trans}+2}$	$s_1(t_{\text{trans}+2})$	$s_2(t_{\text{trans}+2})$	$\dots$	$s_{N_s}(t_{\text{trans}+2})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_n	$s_1(t_n)$	$s_2(t_n)$	$\dots$	$s_{N_s}(t_n)$

The formats of all other files is adjusted correspondingly.

- In order to compress the output data it can be preferable to save the current state not for each time step, but each t_s-th state only. Note, that for dynamical systems discrete in time this kind of saving corresponds to the saving of the trajectory for the t_s-th iterated function. This option has an effect in state space and in velocity space as well. Combining this option with the transient time option described above, the output file for the trajectory becomes the following format:

	column		
1	2	...	$N_s + 1$
$t_{\text{trans}+1}$	$s_1(t_{\text{trans}+1})$	...	$s_{N_s}(t_{\text{trans}+1})$
$t_{\text{trans}+t_s+1}$	$s_1(t_{\text{trans}+t_s+1})$	...	$s_{N_s}(t_{\text{trans}+t_s+1})$
$t_{\text{trans}+2t_s+1}$	$s_1(t_{\text{trans}+2t_s+1})$	...	$s_{N_s}(t_{\text{trans}+2t_s+1})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$t_{n-(n \bmod t_s)}$	$s_1(t_{n-(n \bmod t_s)})$	...	$s_{N_s}(t_{n-(n \bmod t_s)})$

  The formats of all other files is adjusted correspondingly.

- During the investigation of the geometric structure of chaotic attractors or whenever it is necessary to zoom into a region in the state or velocity space, it is very useful to save only states from a specific area. For blow-ups for instance, this option plays an important role to reduce the amount of data to be saved. All blow-ups of trajectories in this work are calculated using this option (see for instance figures 3.5 and 3.6). Note, that some of these figures could not be calculated without this option because the data amount becomes too large.

- The cobweb diagram option allows the saving of graphical iterations also known as web or cobweb diagrams. The figures 3.2(a) - 3.2(d) were produced using this method. Note, that combining this option with the saving of each t_s-th state as described above, the cobweb diagrams for the t_s-th iterated function are saved.

- For spatial inhomogeneous dynamical systems an overview about the complete state can be represented using some graphical formats. The simplest possibility to do this is given by PGM (Portable Gray Map) images. For each component of the state vector $\underline{s}(r, t)$ an image can be created for the temporal interval $t_{\min}, \ldots, t_{\max}$ and spatial interval

$r_{\min}, \ldots, r_{\max}$ via the following transformation:

$$g_{i,j}^{k} = g_{\min} + \left\lfloor \frac{s_k(r,t) - s_{k,\min}}{s_{k,\max} - s_{k,\min}} \left(g_{\max} - g_{\min}\right) \right\rfloor \tag{5.1}$$

Here the index k denotes the k-th component of the state vector $\underline{s}(r,t)$, the indices correspond to the space r and the time t. The minimal and maximal values of the specific component are used:

$$s_{k,\min} = \min_{\substack{r \in [r_{\min}, r_{\max}] \\ t \in [t_{\min}, t_{\max}]}} s_k(r,t) \tag{5.2}$$

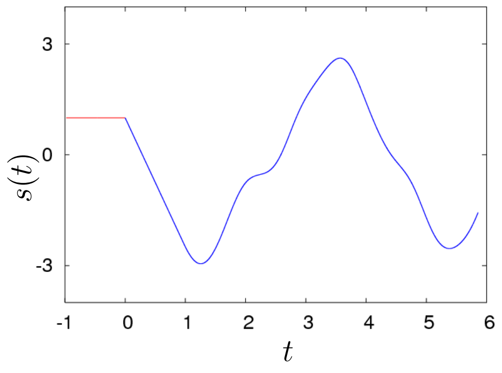
$$s_{k,\max} = \max_{\substack{r \in [r_{\min}, r_{\max}] \\ t \in [t_{\min}, t_{\max}]}} s_k(r,t) \tag{5.3}$$

as well as the minimal and maximal gray map values, which are typically set to the values $g_{\min} = 0$ and $g_{\max} = 255$.

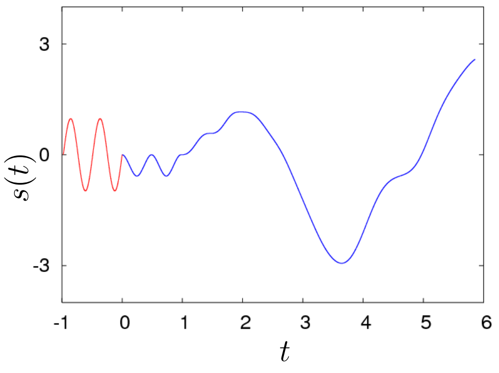
▶ Additionally to the trajectories and velocities the initial states can be saved as well. For some classes of dynamical systems like maps and ODEs this option seems to be redundant, because the initial states for these systems must be given by the user within the initialization phase. However, in many other cases the saving of the initial states may be a useful option, like for instance in the following situations:

- When dealing with dynamical systems with memory, the initial states are given as a set of N_s temporal initial function on the interval $[-\tau_{\max}, 0]$ or $[-1, 0]$ in the standardized representation.
- When dealing with spatial inhomogeneous dynamical systems, the initial states are given as a set of N_s spatial initial function on the spatial domain $[r_{\min}, r_{\max}]$.
- When dealing with Poincaré sections, the initial states for the Poincaré map are calculated by iteration of the dynamical system inside until the Poincaré condition is fulfilled the first time.
- When dealing with scans, the initial states for the next simulation run can be reseted from the last states of the previous simulation run.

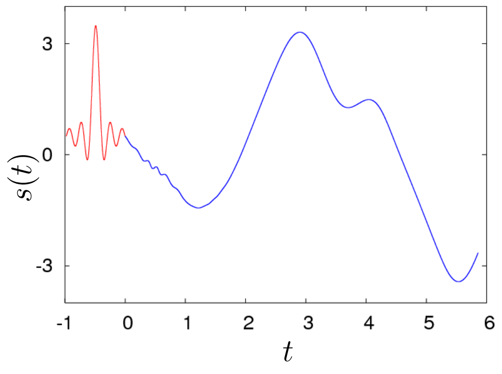
In figures 5.1(a) - 5.1(f) the trajectories of the PLL system with time delay (see Sec. 3.4.2.1) for six different initial functions are shown.



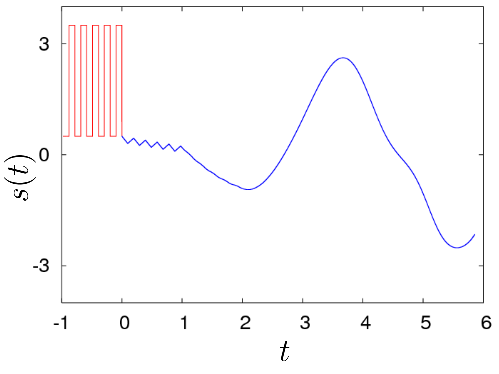
(a) constant initial function



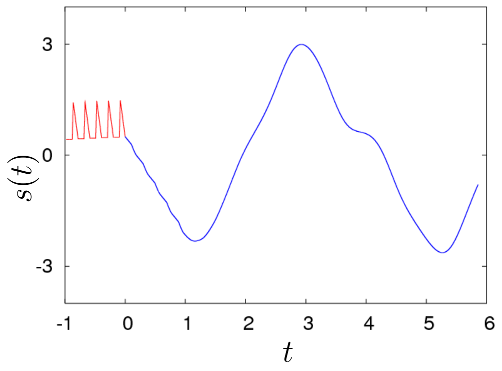
(b) sin initial function



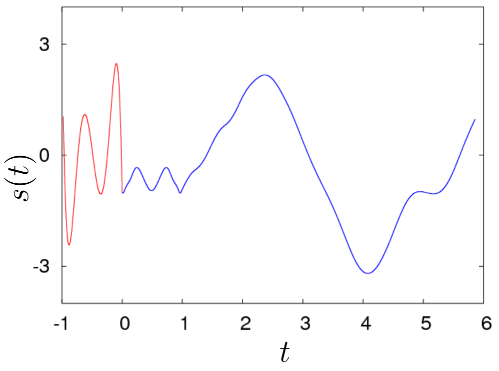
(c) sinc initial function



(d) periodic step initial function



(e) sawtooth initial function



(f) polynomial initial function

Figure 5.1: PLL system: trajectory $s(t)$ for several temporal initial functions

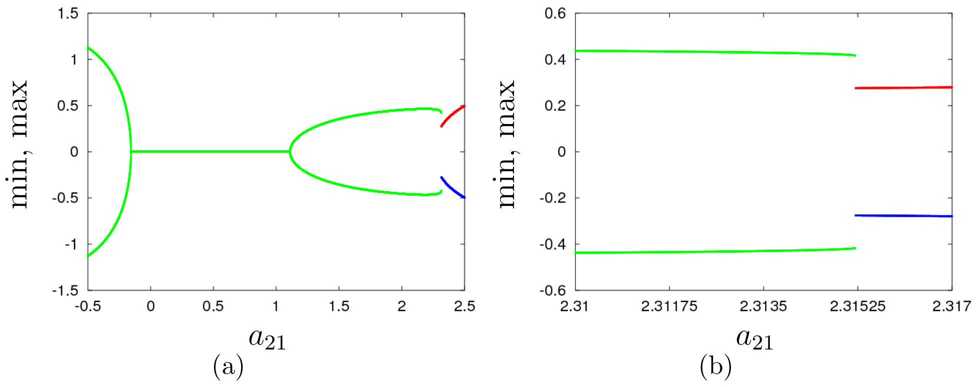


Figure 5.2: Coupled neurons with time delay: minimum - maximum analysis

Presented are in (a), the minimum - maximum analysis performed on the interval $a_{21} \in [-0.5, 2.5]$ *and in (b) a blow-up of the most interesting region, showing the symmetry breaking - symmetry recovering phenomenon.*

5.2.2 Determination of minimum and maximum values

This sub-method of the general trajectory analysis method is, although really simple, sometimes quite efficient to detect basic bifurcation types like for instance transcritical bifurcations, pitchfork bifurcations or Hopf bifurcations. This method is also very suitable for the detection of symmetry breaking - symmetry recovering phenomena as demonstrated in figure 5.2. Shown are the minimum and maximum values of the asymptotic dynamics in dependence of a_{21} which is one of the two coupling parameters of system (3.36) (see Sec. 3.4.2.2). In (a) one can clearly detect the two supercritical Hopf bifurcations occurring in this system at the parameter values $a_{21} \approx -0.155$ and $a_{21} \approx 1.1$. Additionally one can see the symmetry breaking - symmetry recovering bifurcation at $a_{21} \approx 2.31525$ which destroys the limit cycle created at the second Hopf bifurcation and causes two stable fixed points to emerge. The green maximum - minimum curves indicate the symmetric attractors, i.e., a limit cycle for $a_{21} < -0.155$, a fixed point for $-0.155 < a_{21} < 1.1$ and another limit cycle for $1.1 < a_{21} < 2.31525$, whereas the red and blue curves correspond to the two different fixed points symmetric to each other (compare Fig. 3.19).

As already mentioned in chapter 4, the **AnT 4.669** software package allows not only scans with respect to system parameters or parameters of investigation methods but also with respect to initial values. In the case, where the initial values are not given explicitly but automatically generated by the **AnT computation engine** itself, according to the specifications given by the user, scans with respect to parameters of the function generating the initial values are possible as well. In figure 5.3 an example of such a scan applied to the PLL system with time delay (see Sec. 3.4.2.1) using the minimum - maximum analysis is presented.

5.2.3 Determination of the wave number

For an attractor $\mathcal{A}$ of a dynamical system discrete in time the i-th component of the ***wave number*** vector $\underline{\omega}$ is defined by:

$$\omega_i(\mathcal{A}) = \frac{1}{N}\sum_{j=1}^{N} M_{ij} \quad \text{with} \quad M_{ij} = \begin{cases} 1 & \text{if} \quad s_{ij-1} > s_{ij} < s_{ij+1} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Hereby, s_{ij} is the i-th component of the state vector $\underline{s}_j$ at the discrete time j. Roughly speaking one can say, that the wave number represent the number of local minima in the corresponding component of the orbit, averaged in time ([34], [35]). In figure 5.4(a) the wave number of the logistic map is shown in the parameter interval $\alpha \in [3, 4]$. As one can see, between the first period doubling bifurcation ($\alpha_1 = 3.0$) and the last band merging bifurcation ($\bar{\alpha}_1 \approx 3.678535$) the wave numbers of all attractors have the value $\omega = \frac{1}{2}$. With the occurrence of the one-band chaotic attractors, the wave number decreases slowly. It is worth mentioning, that the wave number remains constant not only within the period doubling scenario but within the subsequent band merging scenario as well, indicating, that these two phenomena are closely related to each other.

Figure 5.4(b) shows a further application example of the wave number analysis for another dynamical system discrete in time, namely the ***Agnesi map*** defined by:

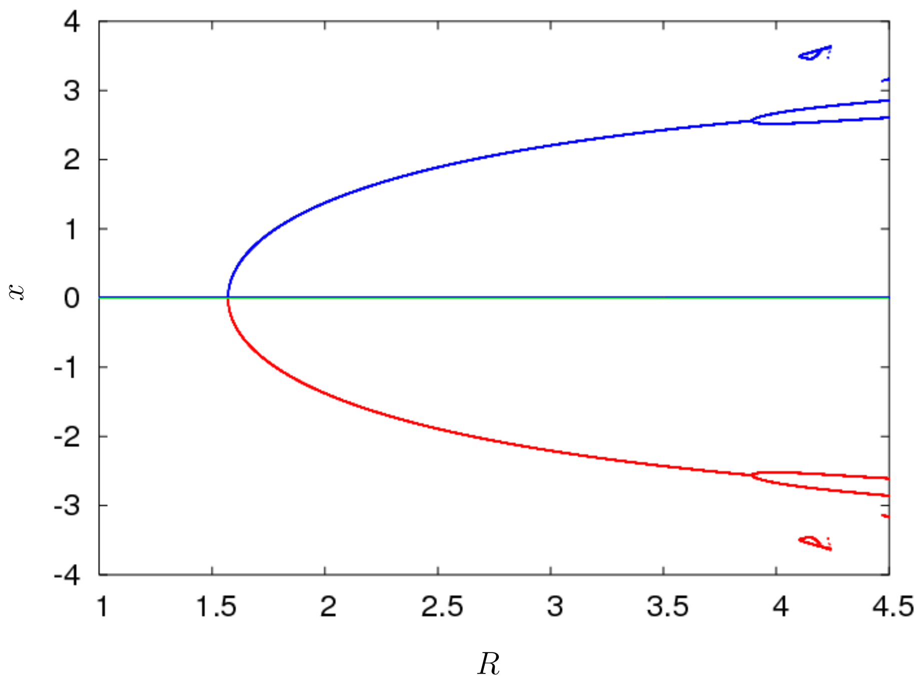


Figure 5.3: PLL with time delay: initial value and system parameter scan
This bifurcation diagram was obtained performing a scan with respect to the system parameter R and the parameter c of the initial value generating function using the minimum - maximum option. The parabola starting at $R = \frac{\pi}{2}$ indicates a supercritical Hopf bifurcation. The unstable fixed point 0 after the Hopf bifurcation is visible here because the initial function $s = 0$ occurs in the scan. The splitting at $R \approx 3.8$ indicates a symmetry breaking - symmetry recovering bifurcation where the symmetric limit cycle emerged at the Hopf bifurcation splits into two coexisting limit cycles symmetric to each other. At $R \approx 4.1$ a new symmetric limit cycle emerges which undergoes at $R \approx 4.103$ also a symmetry breaking - symmetry recovering bifurcation. These two limit cycles undergo a period doubling scenario shown in figure 5.31 for one of them. At $R \approx 4.188$ again a symmetry breaking - symmetry recovering bifurcation causes a symmetric chaotic attractor to emerge.

Definition: ***Agnesi map***

$$x_{n+1} \;=\; f(x_n, \alpha) = \frac{\alpha}{\alpha + x_n^2} \tag{5.5}$$

■

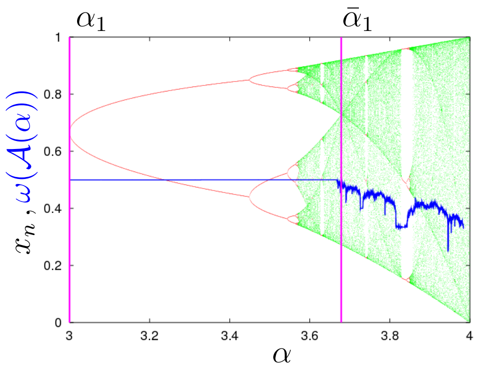
In contrast to the logistic map, the wave numbers of the Agnesi map show a complex self-similar structure corresponding to the infinite number of period doubling scenarios occurring in this system, which can be observed also in the bifurcation diagram. Looking closer at the wave number in figure 5.4(b) one can observe the following:

1. There exists a period increment scenario with an increment of one and a starting period of two for the parameter value $\alpha = 0$, that is on the right part of the bifurcation diagram.
2. Between each two adjacent regions of this scenario, a period doubling scenario takes place each starting with the period of the region to the right.
3. Consequently, the wave numbers start with the value $\omega = \frac{1}{2}$ from the right corresponding to the period two region and its subsequent period doubling scenario. In the next region of the increment scenario with period three and the subsequent period doubling scenario, the wave numbers have the value $\omega = \frac{1}{3}$. As one can see, this behavior holds on until finally the fixed point is reached at the parameter value $\alpha = 6.75$.
4. For $\alpha < -6.75$, the wave numbers have the value $\omega = 0$.

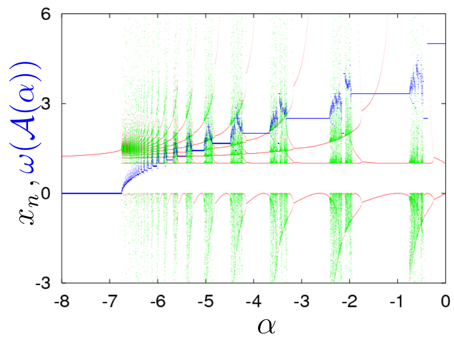
5.2.4 Basic statistics

This method performs the calculation of basic statistic properties of the simulated trajectory, namely the mean values, standard deviations, averaged velocities and the correlation coefficients.

Chapter 5
Supported investigation methods



(a) Logistic map: wave number and bifurcation diagram



(b) Agnesi map: wave number and bifurcation diagram

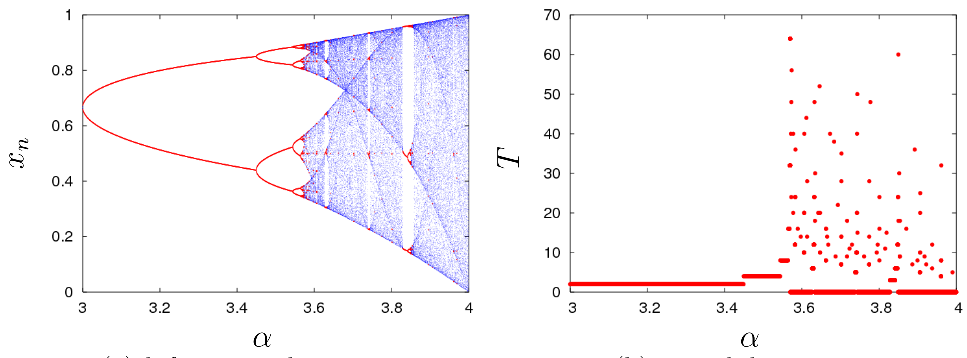
Figure 5.4: Wave number analysis

Shown are the wave numbers *of the logistic map (a) and the Agnesi map (b) together with the* periodic *parts and the* aperiodic *parts of the bifurcation diagrams corresponding to* periodic *or* aperiodic *attractors $\mathcal{A}$. In (a), the point $\alpha_1 = 3.0$ of the first period doubling bifurcation and the point $\bar{\alpha}_1 \approx 3.678535$ of the last band merging bifurcation are marked, to illustrate the fact, that the wave numbers remain constant not only within a period doubling scenario, but within the subsequent band merging scenario as well. In (b) one can observe, that the wave numbers have different values for different period doubling scenarios, but remain constant within one scenario. In the case of the Agnesi map, the wave numbers were multiplied by a factor of 10.*

5.3 Period analysis

This method implements the calculation of the period (length of a ***periodic attractor*** or ***limit cycle***) for dynamical systems discrete in time. The search for the period is realized using a direct comparison of the last state of the orbit with previous states proceeding backward in time. Therefore the maximal period to be searched for must be specified by the user with a corresponding entry in the initialization file (see Sec. 8.2.1) or of course via the graphical user interface (see Chap. 8). The accuracy, used for the comparison of the states can be specified by the user as well. The appropriate setting hereby is strongly depending on the investigated dynamical system, or, more precisely, on the convergence rate to the attractor. For discrete maps the value 10^{-12} may be in many cases a suitable choice. Due to the phenomenon of ***critical slowing down***, in the vicinity of instabilities or bifurcation points, this value should be enlarged. When dealing with Poincaré maps, larger values (up to 10^{-6}) should be used, because in this case the convergence is also slow due to the iteration of a dynamical system inside the Poincaré map.

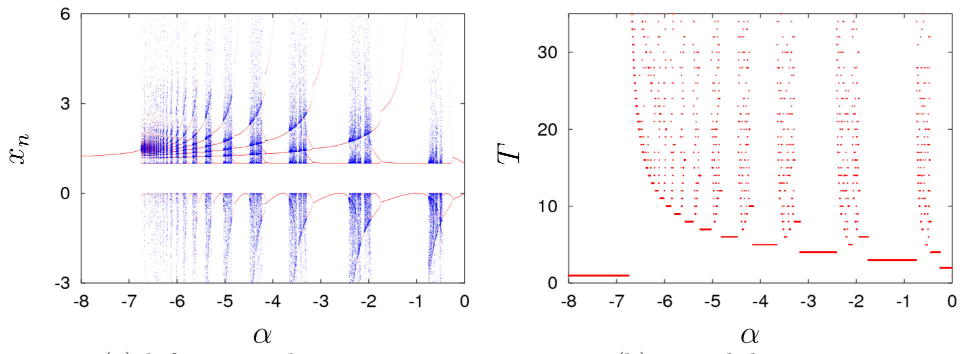
Based on the period calculation not only the period (if any detected) but several other values can be saved as well. For instance, the saving of bifurcation diagrams is realized as a part of the period analysis, because it is preferable to save the periodic and the aperiodic bifurcation diagrams separately. This can be seen for instance in the figures 5.4(a), 5.4(b), 5.5 and 5.6. Additionally, periodic cobweb diagrams, can be saved as well. Furthermore, parameter settings leading to a given period, so-called period selections can be saved. In figures 5.7 and 5.8 regions with different periodic solutions of the logistic map are shown. The figures can be produced using the period selection option.



(a) bifurcation diagram (b) period diagram

Figure 5.5: Logistic map: bifurcation and period diagram

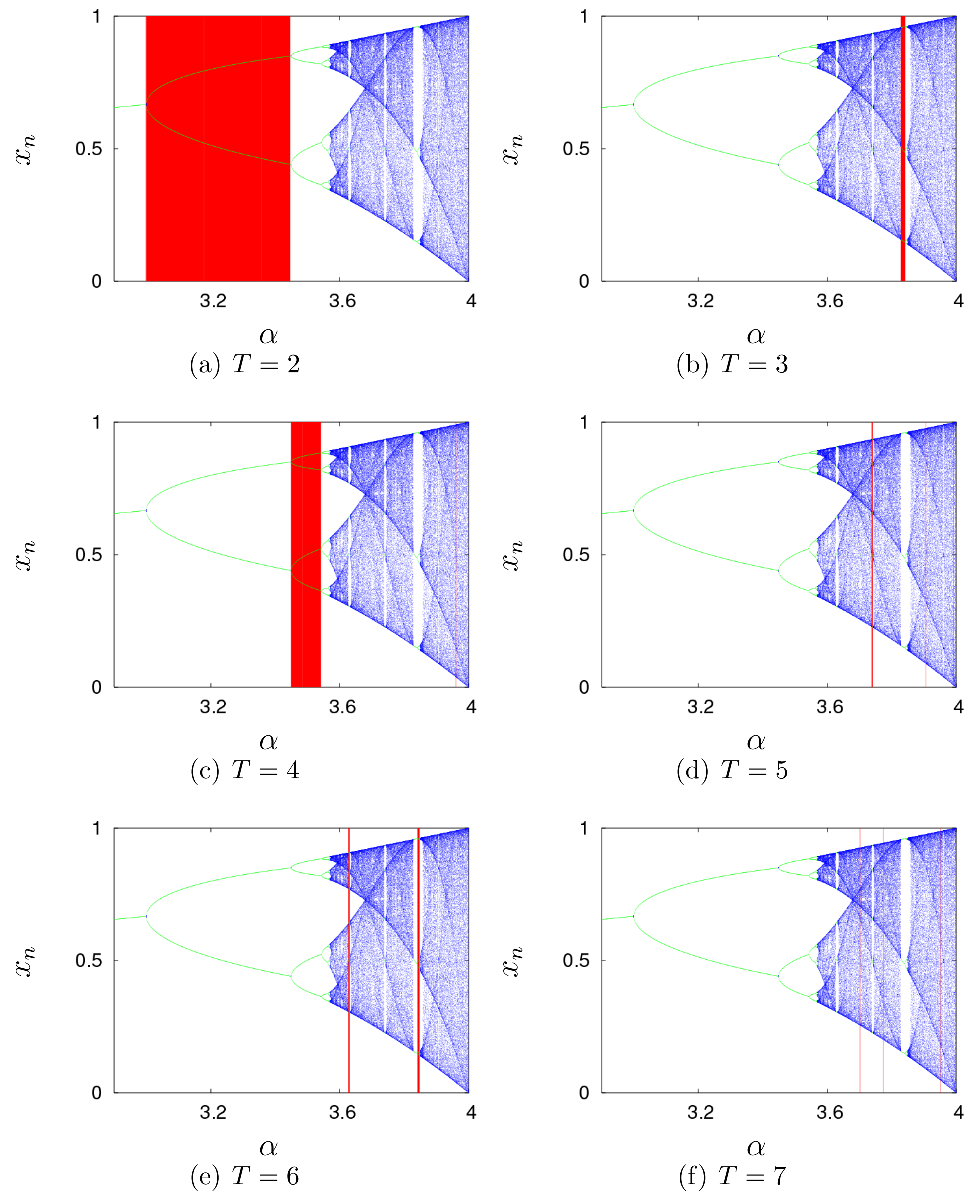
Shown is the interesting part of the bifurcation diagram with periodic and aperiodic solutions (a) and the corresponding period (b). When in the given part of the orbit a period cannot be detected, the period calculation method returns the value 0. This can be clearly observed in the chaotic region.



(a) bifurcation diagram (b) period diagram

Figure 5.6: Agnesi map: bifurcation and period diagram

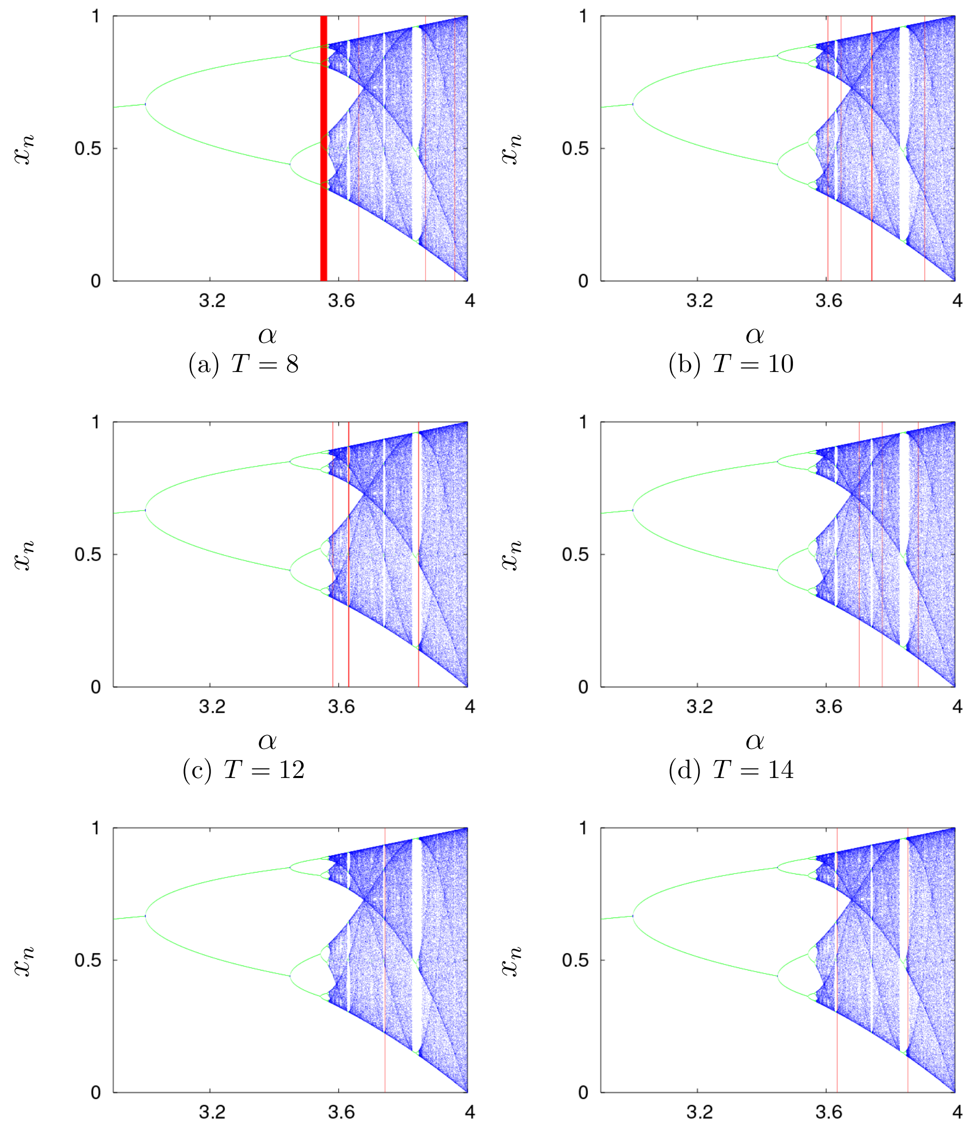
Shown is the interesting part of the bifurcation diagram with periodic and aperiodic solutions (a) and the corresponding period (b). When looking closer, in (b) the infinite sequence of period doubling scenarios can be observed.



(a) $T = 2$ (b) $T = 3$ (c) $T = 4$ (d) $T = 5$ (e) $T = 6$ (f) $T = 7$

Figure 5.7: Logistic map: period selections

In the red colored regions solutions with the labeled period were found: $T = 2$ (a), $T = 3$ (b), $T = 4$ (c), $T = 5$ (d), $T = 6$ (e), $T = 7$ (f)



(a) $T = 8$ (b) $T = 10$ (c) $T = 12$ (d) $T = 14$ (e) $T = 15$ (f) $T = 18$

Figure 5.8: Logistic map: period selections

In the red colored regions solutions with the labeled period were found: $T = 8$ (a), $T = 10$ (b), $T = 12$ (c), $T = 14$ (d), $T = 15$ (e), $T = 18$ (f)

5.4 Region analysis

This method is developed in order to analyze two-dimensional parameter spaces, or more general, arbitrary two-dimensional scan spaces. The goal of this method is to detect areas in these spaces, where the investigated dynamical system shows a qualitatively similar dynamic behavior. In order to detect these areas some other investigation method has to be applied before the region analysis. Then the similarity of the dynamic behavior can be determined according to the results of this investigation method. In the current realization the region analysis is performed based on the period analysis. In this case the scan setting leading to the same period of the asymptotic dynamics of the investigated system are defined as similar and therefore belong to the same area. The borders of these areas are the points where the period of the asymptotic dynamics changes. Therefore, the borders represent the bifurcation lines and play an important role for the description of the dynamic behavior of the investigated system. Especially, it turned out, that the bifurcation lines can form some characteristic structures in the parameter space, which lead to the concept of two-parametric bifurcations and two-parametric bifurcation scenarios. For instance, by applying the region analysis some new phenomena in this field, like big bang bifurcations introduced in [8] and big bang bifurcation scenarios described and investigated in [7], were discovered. In many other cases, the region analysis can be used to determine the structure of the two-dimensional parameter space. Knowledge about this structure is useful for choosing the most suitable parameter settings for one-dimensional scans, because these scans represent obviously some one-dimensional cuts of the more general two-dimensional parameter scans.

Note, that from the image processing point of view, the region analysis corresponds to a simple edge detection in an image, which corresponds to the two-dimensional scan space.

Some application examples for the region analysis are shown in the figures 5.9, 5.10, and 5.11. Three characteristic types of big bang bifurcations, namely period increment big bang, period adding big bang and period doubling big bang are presented. Big bang bifurcations are relatively new and interesting bifurcation phenomena, which can be observed only in two or higher dimensional parameter spaces. In this case, an infinite number of different asymptotic dynamics can be observed within any open convex neighborhood of the bifurcation point. For the classification of two-parametric big bang bifurcations, specific one-parametric bifurcation scenarios along the border of this neighborhood can be used.

The period increment big bang and period adding big bang bifurcations can be observed for instance in the ***piecewise-linear map***, defined by:

Definition: ***Piecewise-linear map***

$$\begin{array}{c} x_{n+1} = f(x_n) \\ f : [0,1] \mapsto [0,1] \end{array} \qquad f(x) = \left\{ \begin{array}{lcll} f_l(x) & = & bx + c & \text{if} \quad x < \frac{1}{2} \\ f_r(x) & = & x - a & \text{if} \quad x \geq \frac{1}{2} \end{array} \right. \tag{5.6}$$

■

This system has three parameters a, b and c, which have to fulfill the following conditions:

$$a \in \left[0, \tfrac{1}{2}\right] \qquad |b| < 1 \qquad \max\left\{0, \tfrac{1}{2} - \tfrac{b}{2}\right\} < c \leq \min\left\{1, 1 - \tfrac{b}{2}\right\} \tag{5.7}$$

It can be shown, that if this conditions are fulfilled, then the function f maps the interval $[0,1]$ on itself.

In figure 5.9.(a), the structure of the two-dimensional parameter space $a \times c$ caused by the period increment bing bang bifurcation at the point $(0, 0.5)$ is shown. Figures 5.9.(b) and 5.9.(c) represent the bifurcation scenario along the curve around the bifurcation point marked in 5.9.(a). Note, this is an example of a real elliptic scan in two dimensions as presented and discussed in section 4.3.7. In figure 5.10, the behavior of this system is shown for another parameter setting, where a period adding big bang bifurcation occurs.

An example for a period doubling big bang bifurcation can be observed in the ***power law map***, defined by:

Definition: ***Power law map***

$$x_{n+1} = f(x_n) \qquad f : [0,1] \mapsto [0,1], \quad a, b > 0 \tag{5.8}$$
$$f(x) = a2^b \left(\left(\frac{1}{2} \right)^b - \left| x - \frac{1}{2} \right|^b \right)$$

■

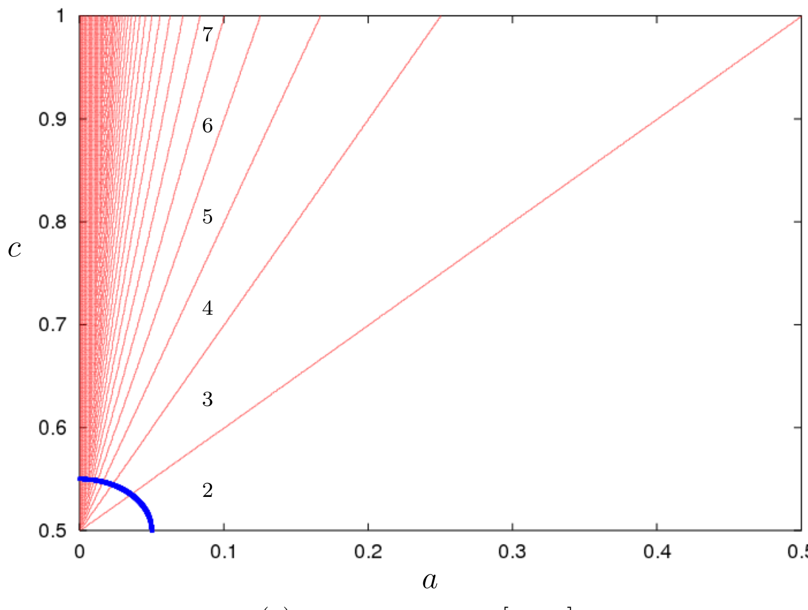
Note, that for $b = 1$ this system corresponds to the well-known ***tent map*** defined by:

Definition: ***Tent map***

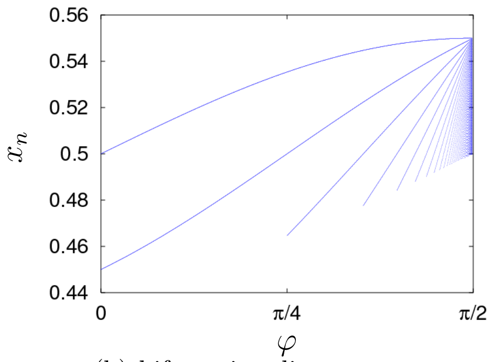
$$x_{n+1} = f(x_n) \qquad f : [0,1] \mapsto [0,1], \quad r \in [0,2] \tag{5.9}$$
$$f(x) = \begin{cases} rx & \text{if} \quad x \leq \frac{1}{2} \\ r(1-x) & \text{if} \quad x > \frac{1}{2} \end{cases}$$

■

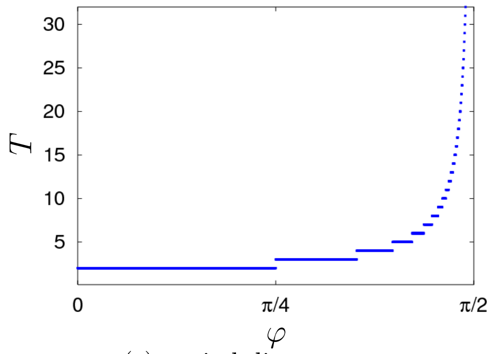
and for $b = 2$ to the logistic map (see Sec. 3.3.1.1).
The figure 5.12 shows the result of the region analysis for the two-dimensional Bogdanov map (see Sec. 3.3.2.1). As one can see, the structure of this region analysis diagram is much more complex than that of the previous ones. However, a lot of interesting results can be obtained from this representation indicating, that the creation of region diagrams in two dimensional parameter spaces is a very powerful investigation method especially when compared with one-dimensional scans, such as shown in figures 5.13 and 5.14



(a) parameter space $[a \times c]$



(b) bifurcation diagram



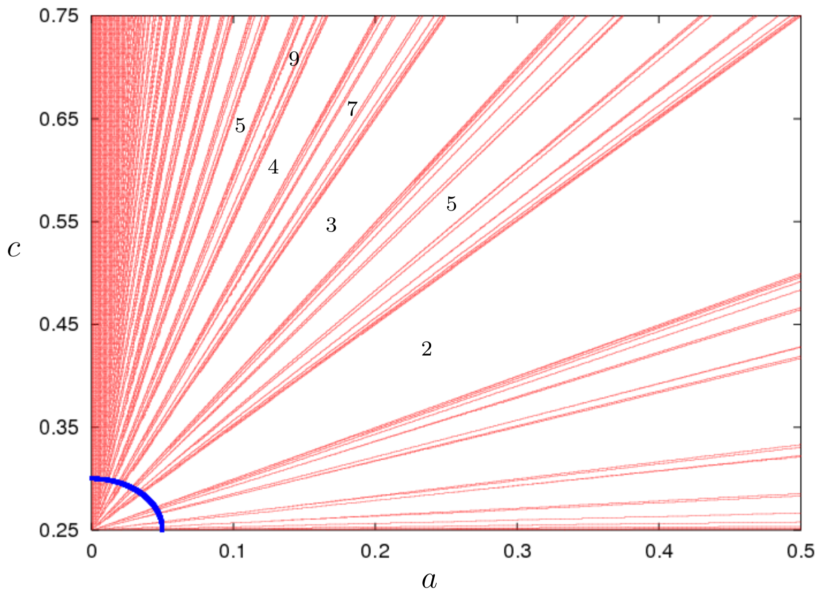
(c) period diagram

Figure 5.9: Piecewise linear map: period increment big bang bifurcation

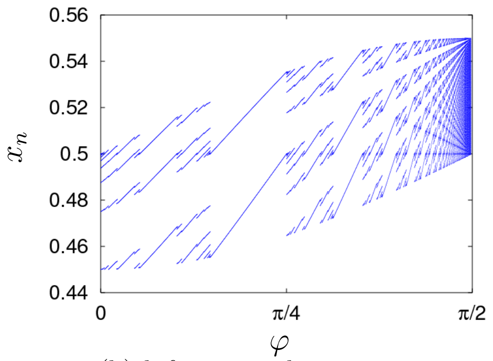
(a) For the parameter value $b = 0$ there occurs a period increment big bang bifurcation in this system at the point $(a = 0, c = 0.5)$. The regions with the periods $T = 2$ up to $T = 7$ are marked.

(b) bifurcation diagram of the period increment scenario along the elliptic curve marked in (a)

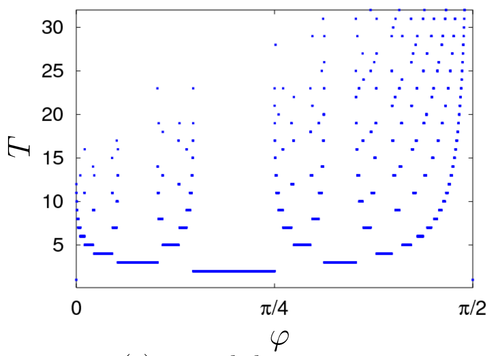
(c) period diagram of the period increment scenario along the elliptic curve marked in (a)



(a) parameter space $[a \times c]$



(b) bifurcation diagram



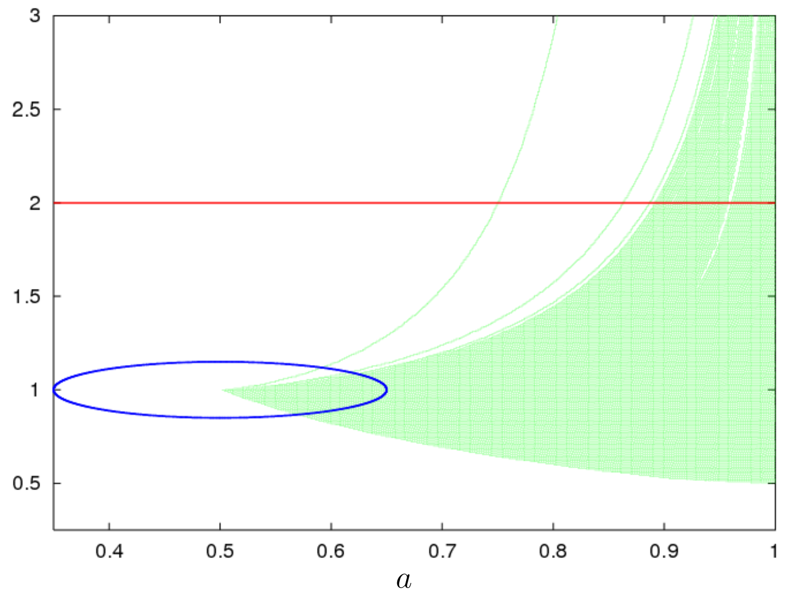
(c) period diagram

Figure 5.10: Piecewise linear map: period adding big bang bifurcation

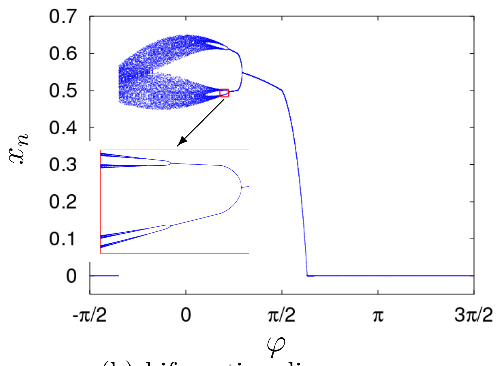
(a) *For the parameter value $b = 0.5$ there occurs a period adding big bang bifurcation in this system at the point $(a = 0, c = 0.25)$. Some regions with the periods $T = 2$ up to $T = 7$ are marked.*

(b) *bifurcation diagram of the period adding scenario along the elliptic curve marked in (a)*

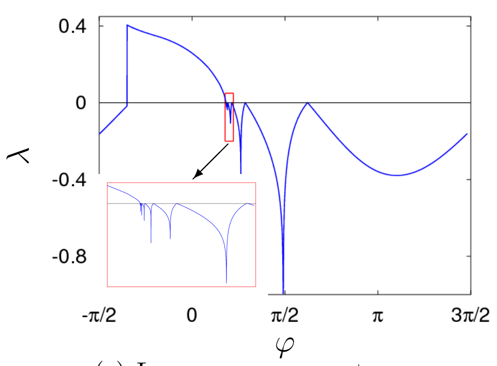
(c) *period diagram of the period adding scenario along the elliptic curve marked in (a)*



(a) parameter space $[a \times b]$



(b) bifurcation diagram



(c) Lyapunov exponent

Figure 5.11: Power law map: period doubling big bang bifurcation

At the point $(a = 0.5, b = 1.0)$*, there occurs a period doubling big bang bifurcation in this system. The period doubling scenario along the blue elliptic curve marked in (a) is shown in (b) and (c), together with blow-ups of characteristic regions. Note, the red straight line marked in (a) corresponds to the usual logistic map.*

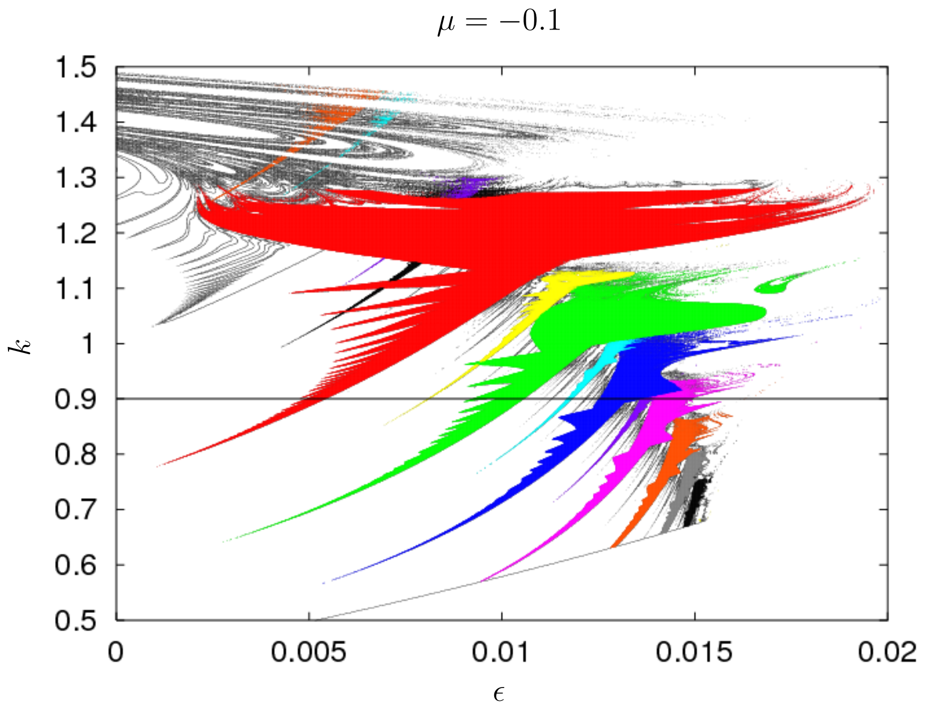
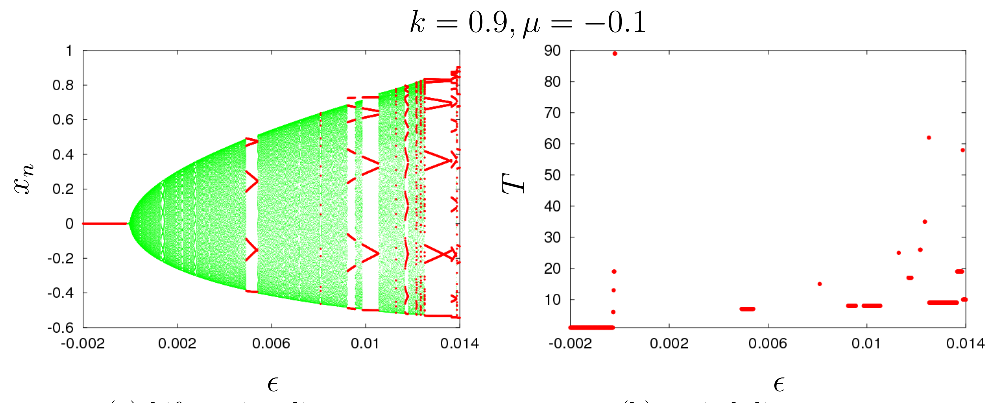


Figure 5.12: Bogdanov map: region diagram

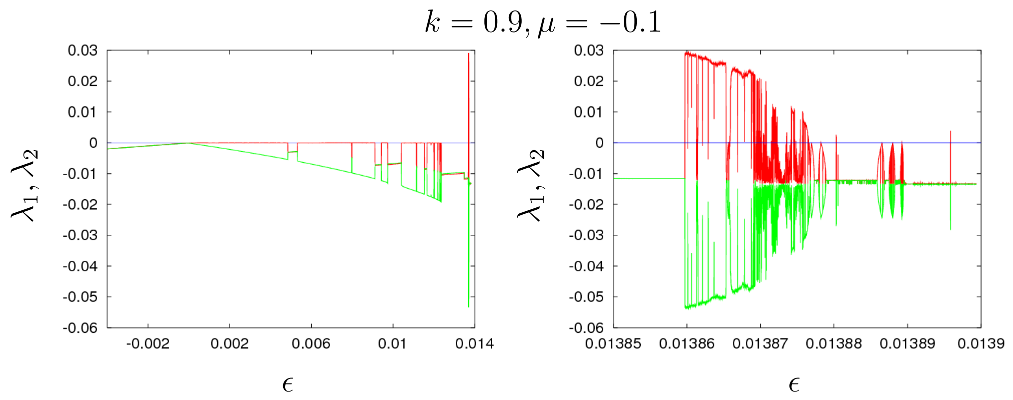
This region diagram shows a really complex structure which is not clearly understood until now. The colored regions belong to a period adding scenario which is not created by a two-dimensional period adding big bang bifurcation. Marked are the following periodic solutions: 7-periodic, 8-periodic, 9-periodic, 10-periodic, 11-periodic, 12-periodic, 13-periodic, 15-periodic, 17-periodic, 19-periodic. The horizontal line at $k = 0.9$ *marks the one-dimensional parameter scans presented in figures 5.13 - 5.14.*



(a) bifurcation diagram (b) period diagram

Figure 5.13: Bogdanov map: bifurcation and period diagram

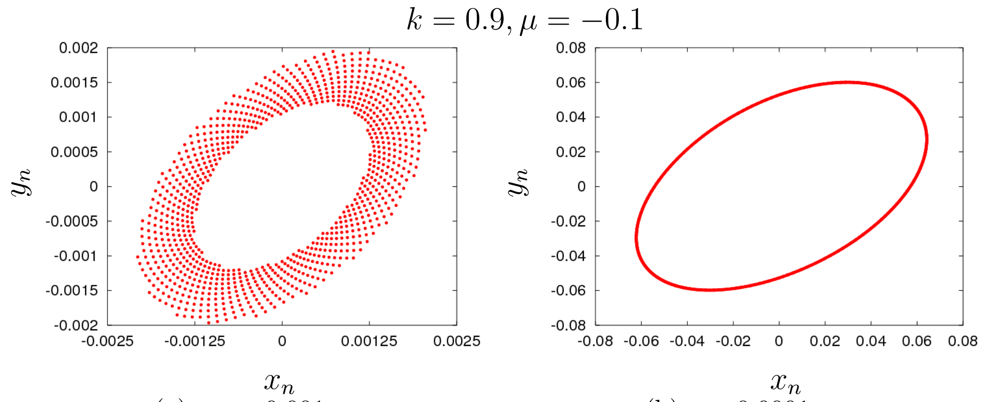
In (a) clearly the Neimark-Sacker bifurcation at $\epsilon = 0$ can be observed. Shown are the periodic and aperiodic solutions indicating, that immediately after the bifurcation quasiperiodic solutions emerge. This can be also clearly seen in figure 5.14 when looking at regions where the largest Lyapunov exponent is zero. The periodic solutions in (b) at the bifurcation point $\epsilon = 0$ are numerical artifacts caused by the critical slowing down phenomenon (compare Fig. 5.15).



(a) Lyapunov exponents (b) Lyapunov exponents (blow-up)

Figure 5.14: Bogdanov map: Lyapunov exponents

In (a) one can clearly observe, the quasi periodicity of the solutions in the regions where the largest Lyapunov exponent is zero. In (b) a blow-up of the region to the right is shown, where also chaotic solutions exist as indicated by the positive largest Lyapunov exponent.



(a) $\epsilon = -0.001$ (b) $\epsilon = 0.0001$

Figure 5.15: Bogdanov map: two selected orbits

In (a) a typical solution before but close to the Neimark-Sacker bifurcation is shown. Due to the phenomenon of critical slowing down, the orbit is not converged to the fixed point 0 although $2 \cdot 10^6$ *iterations were performed. Although not at this value of the parameter* ϵ *but for values much closer to the instability, this behavior leads to the detection of for instance the period 19 shown in figure 5.13(b). A quasiperiodic orbit after the Neimark-Sacker bifurcation is shown in (b).*

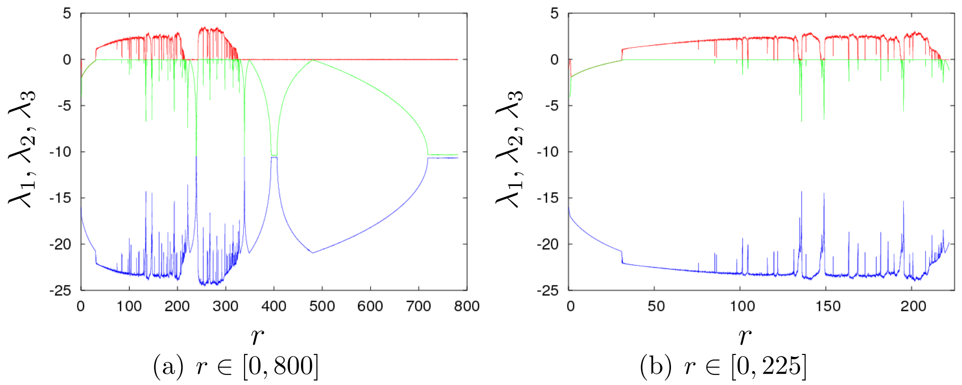


Figure 5.16: Lorenz system: Lyapunov spectrum

The three Lyapunov exponents of the Lorenz system are shown in the interval $r \in [0, 800]$ (a) and $r \in [0, 225]$ (b). The other two parameters were kept fixed at the values $\sigma = 16$ and $b = 4$. In the case of unique attractors, one can read off the qualitative dynamics of the system from the Lyapunov spectrum. Regions, where all Lyapunov exponents are negative correspond to fixed points, regions where the largest exponent is zero correspond to periodic solutions, i.e., limit cycles. Regions, where the largest exponent is positive correspond to chaotic attractors. The characteristic self-similar tongue-like structures (often seen clearly only in blow-ups) indicate period doubling scenarios.

5.5 Lyapunov exponents analysis

The calculation of ***Lyapunov exponents*** is implemented based on the approach of Wolf et al. [198]. In fact, there are two variants of this method. The first one, is based on the original approach, which uses the linearized system which has to be provided by the user of the **AnT 4.669** software package in addition to the system function itself. The second one, is a slightly modified approach, which tracks adjacent trajectories and hence uses the system function only. The method calculates the temporal averaged local divergence rates to approximate the Lyapunov exponents. In figure 5.16 the complete ***Lyapunov spectrum*** for the Lorenz system (see Sec. 3.4.1.1), that is all three Lyapunov exponents are shown.

In figure 5.17 a two-dimensional system parameter scan of the Lorenz system using the Lyapunov exponent analysis is shown from two different viewpoints.

Both views indicate the possibility, that in this system at least two big bang bifurcations take place causing the complex dynamic behavior. Note, that the Lyapunov spectrum shown in figure 5.16(a) corresponds to a section of this diagram at the parameter value $\sigma = 16$.
In figure 5.18 an incomplete Lyapunov spectrum of the ***Mackey Glass system*** is shown together with the corresponding Lyapunov dimension. This dynamical system of the DDE class was proposed in [58] as a model for the production of white blood cells in the human body. Like in the case of the PLL with time delay (see Sec. 3.4.2.1) a scaling of time leads to the following standardized form:

Definition: *Mackey Glass system*

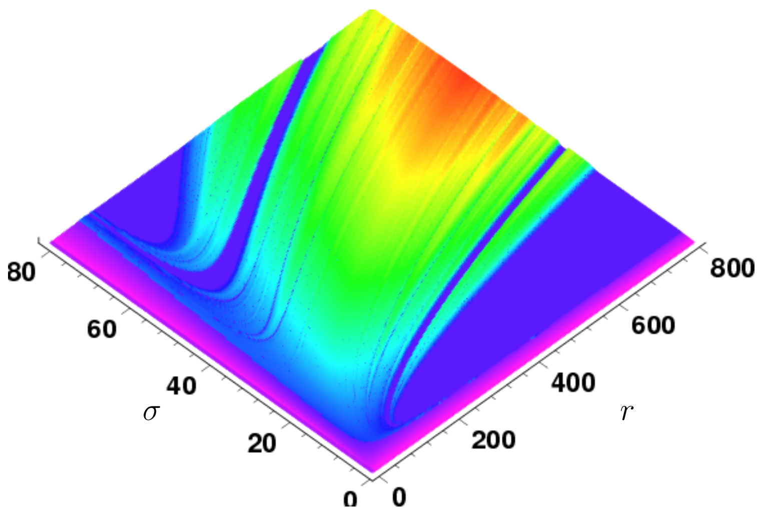
$$\begin{aligned} \frac{d}{dt}s(t) &= f\left(s(t), s(t-1), \underline{p}\right) \\ s(t) &\in \mathbb{R} \\ \underline{p} &= (a,b) \in \mathbb{R}^2 \\ f\left(s(t), s(t-1), \underline{p}\right) &= a\,\frac{s(t-1)}{1+s(t-1)^{10}} - b\,s(t) \end{aligned} \tag{5.10}$$

■

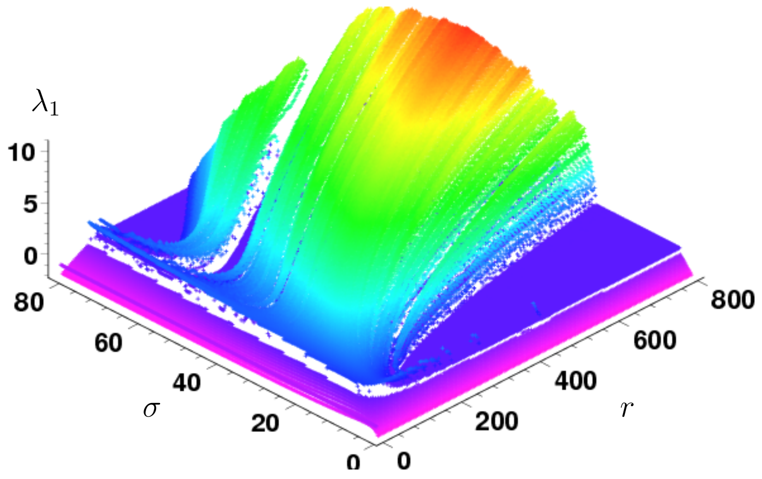
Here, $s(t)$ is the density of the circulating white blood cells and a and b are the system parameters. The parameter a corresponds to the creation rate of the white blood cells, whereas the parameter b is the destruction rate of the white blood cells. The function

$$F(s(t-1), a) = a\,\frac{s(t-1)}{1+s(t-1)^{10}}$$

is the current flux of new produced white blood cells into the blood in response to the demand created at time $t-1$ in the past. Due to the scaling of time, which leads to the fixed time delay of 1, a change of the delay time τ corresponds to a simultaneous and appropriate change of both system parameters a and b. Although not so well-known as the Lorenz system, the Mackey Glass system has become a paradigm of scalar delay differential equations showing a rich dynamic behavior including deterministic chaos. Important for the complex dynamics is the fact, that the flux function F corresponding to the retarded or memory part of the system has a hump.



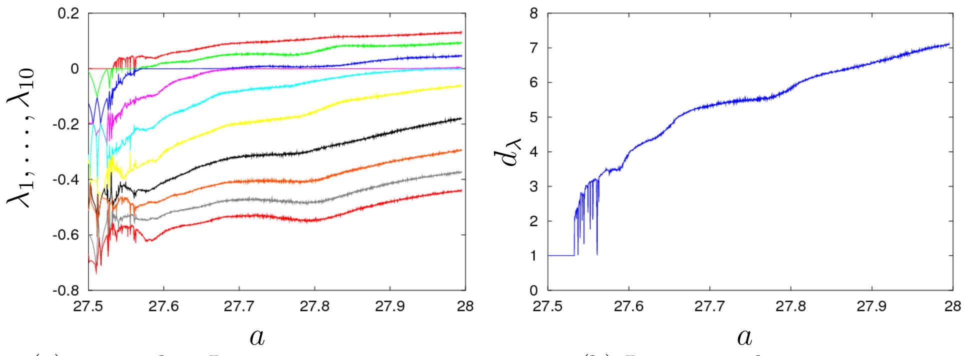
(a) largest Lyapunov exponent (view from above)



(b) largest Lyapunov exponents

Figure 5.17: Lorenz system: largest Lyapunov exponent

This figure shows evidence, that in the Lorenz system big bang bifurcations cause the complex dynamic behavior. The calculation of such diagrams with a high resolution is important, but can only be performed using the client/server mode of the **AnT computation engine** *and a large number of workstations or a cluster of computing nodes.*



(a) incomplete Lyapunov spectrum (b) Lyapunov dimension

Figure 5.18: Mackey Glass system: Lyapunov spectrum and dimension
In (a) the ten largest Lyapunov exponents of the Mackey Glass system are shown in the interval $a \in [27.5, 28]$, whereas in (b) the corresponding Lyapunov dimension is presented.

Details about this dynamical systems can be found for instance in [59].

The fact, as already mentioned in section 3.4.2.3, that the initial value problem of dynamical systems with memory continuous in time can be defined properly only in an extended state space has a remarkable consequence: The state space extension causes an infinite but countable number of Lyapunov exponents to exist. However, the inevitable discretization in time, required by the numerical integration of dynamical systems continuous in time, leads to an approximation of the time continuous system by a high-dimensional map, which has only a finite number of Lyapunov exponents. Hence, when integrating a standardized delay differential equation (in this case the time delay τ is exactly one unit in time) numerically with a step size of Δt leads to an approximation by a $\frac{1}{\Delta t}$-dimensional map. Of course, this map has $\frac{1}{\Delta t}$ Lyapunov exponents. Accordingly, changing the discretization, i.e., the step size Δt changes the number of possible Lyapunov exponents to be calculated.

5.6 Dimension analysis

This method is based on the box counting approach and allows the approximation of the invariant or natural measure of an attractor. Based on this approximation, the following characteristics quantities of an attractor can be calculated:

- metric entropy (Kolmogorov-Sinai)
- capacity dimension
- information dimension
- correlation dimension

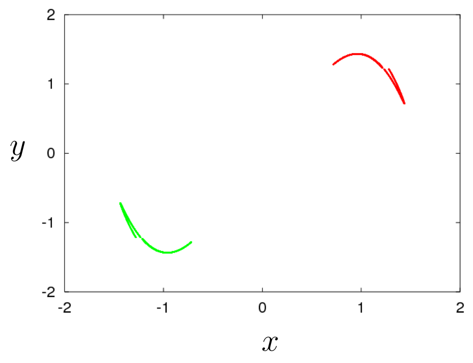
Additionally the variation of the invariant measure of the attractor can be calculated as well. This method is illustrated using the ***Duffing map*** defined by:

Definition: *Duffing map*

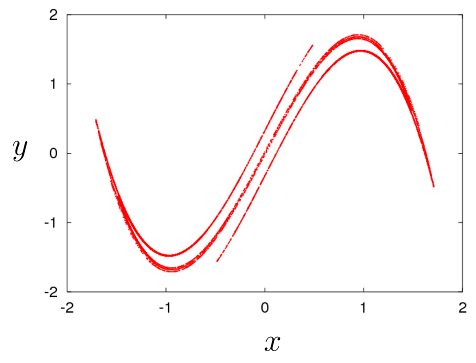
$$\begin{aligned} x_{n+1} &= y_n \\ y_{n+1} &= -bx_n + ay_n - y_n^3 \end{aligned} \tag{5.11}$$

■

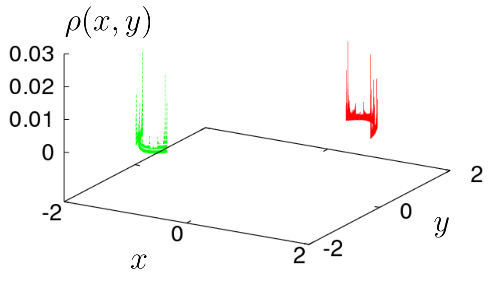
In figure 5.19, the invariant or natural measure of the Duffing map is shown together with the corresponding chaotic attractors. It represents a measure for the probability density of the attractor. In the left part of figure 5.19, two coexisting chaotic attractors symmetric to each other are shown, whereas in the right part one symmetric chaotic attractor is shown. Note, that in the connecting parts in the middle of the symmetric chaotic attractor the probability density is small whereas in both parts belonging to the former coexisting attractors the probability density is large. This indicates, that the symmetric attractor was created by symmetry breaking - symmetry recovering ***crisis*** or ***global bifurcation***.



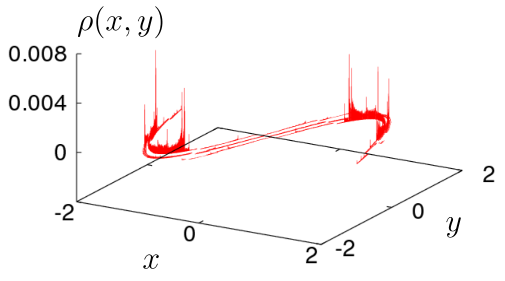
(a) $a = 2.7$



(b) $a = 2.75$



(c) $a = 2.7$



(d) $a = 2.75$

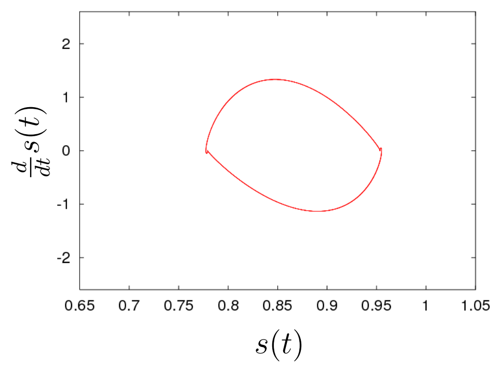
Figure 5.19: Duffing map: invariant measure of chaotic attractors
In (a) two coexisting chaotic attractors symmetric to each other are shown and in (c) the corresponding invariant or natural measures. In (b) a symmetric chaotic attractor is shown and in (d) its corresponding invariant or natural measure. Parameter setting: $b = 0.2$.

5.7 Frequency analysis

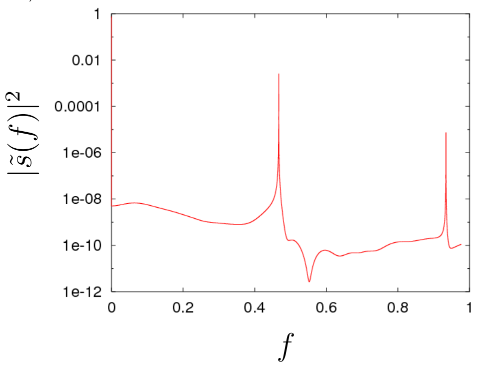
The ***frequency analysis*** or ***spectral analysis*** is based on the calculation of the ***Fast Fourier Transform*** (see for instance [21, 15, 173]) of the simulated orbit. For the input data $\underline{s}(t)$ the complex-valued discrete ***Fourier Transform*** (see [174]) $\tilde{\underline{s}}(\omega)$ is calculated using the external library for the Fast Fourier Transform `FFTW` [46], which is free software. This investigation method allows for instance the calculation of the real- and imaginary part of the Fourier Transform of each state component as well as the corresponding ***power spectrum*** and the ***autocorrelation function***. Some additional evaluation methods are available as well, like for instance the saving of the calculated Fourier coefficients. Using this extension, one can track the change of a limit cycle for instance after a Hopf bifurcation and can thus compare the numerical results with analytically calculated ones as it was done in [146] in order to validate the numerical and analytical results by a consistency check.

In figures 5.20 - 5.22 eight power spectra together with the corresponding orbits of the Mackey Glass system (see page 129 are presented. As one can easily read off from the spectra, a period doubling scenario takes place in this system. This period doubling scenario can also be easily detected by investigating the Lyapunov spectrum presented in figure 5.18(a).

$a = 26.5, b = 20.0$

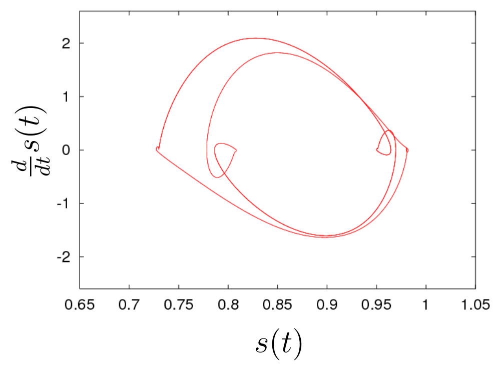


(a) period-1 orbit

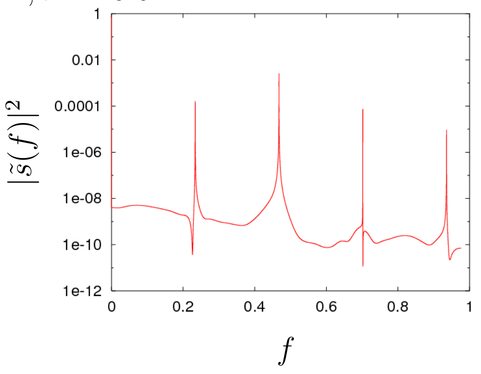


(b) power spectrum

$a = 27.2, b = 20.0$

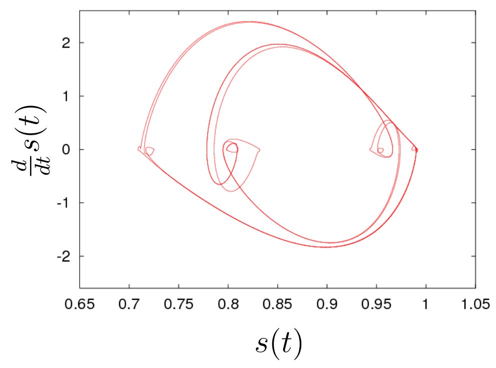


(c) period-2 orbit

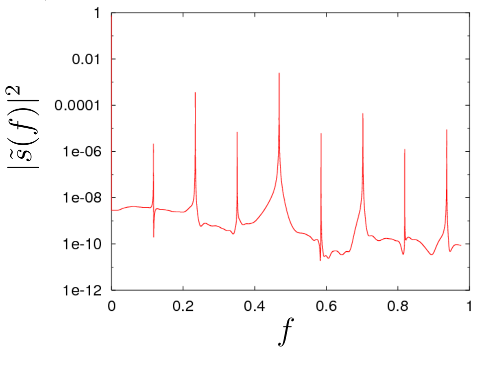


(d) power spectrum

$a = 27.46, b = 20.0$



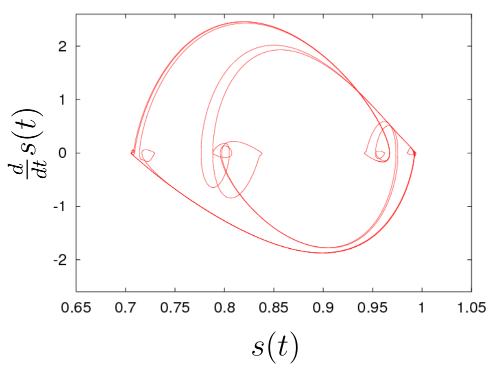
(e) period-4 orbit



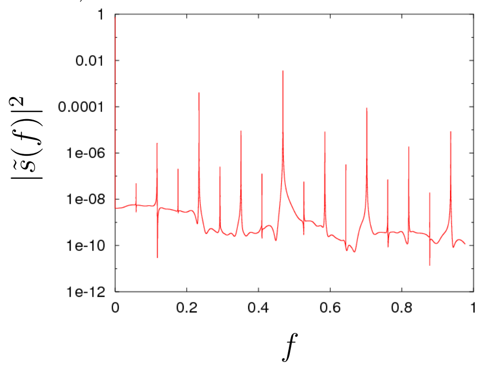
(f) power spectrum

Figure 5.20: Mackey Glass system: orbits and corresponding power spectra

$a = 27.5115, b = 20.0$

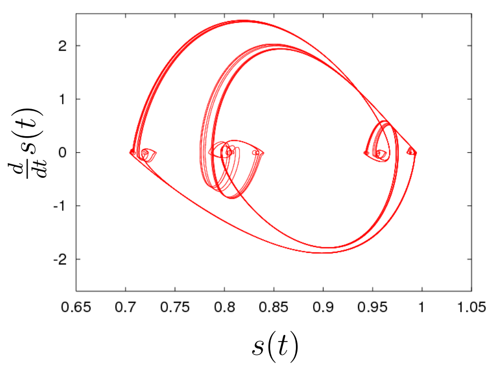


(a) period-8 orbit

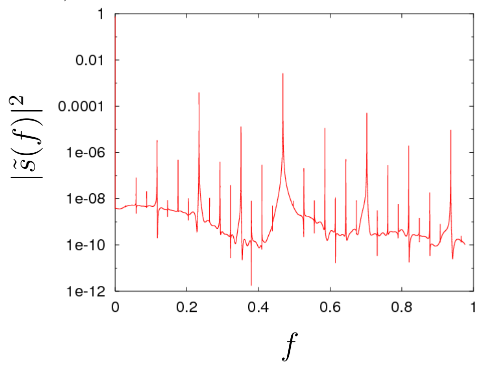


(b) power spectrum

$a = 27.528, b = 20.0$

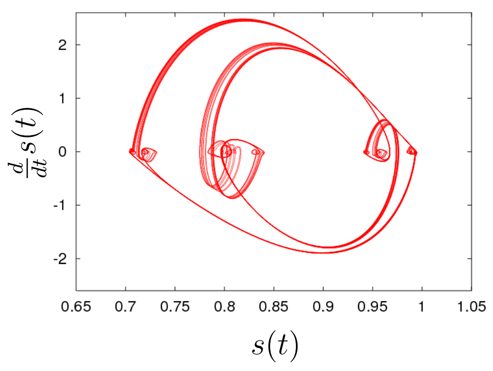


(c) period-16 orbit

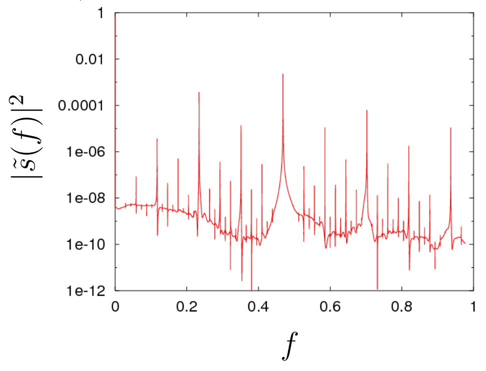


(d) power spectrum

$a = 27.5323, b = 20.0$

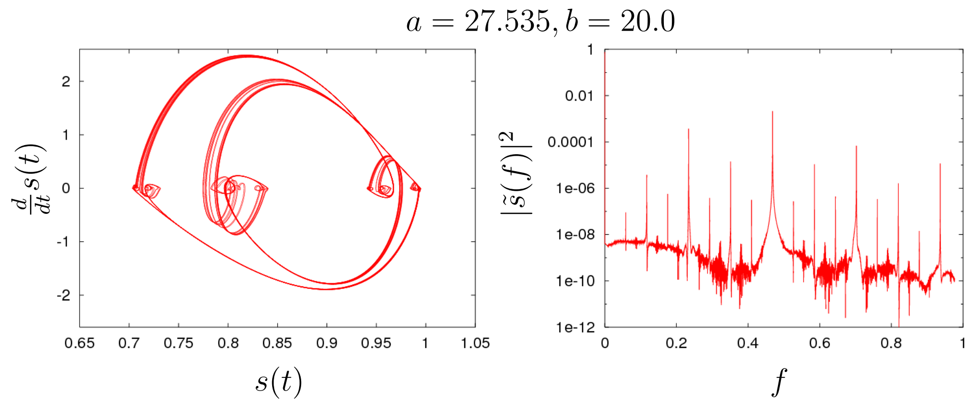


(e) period-32 orbit

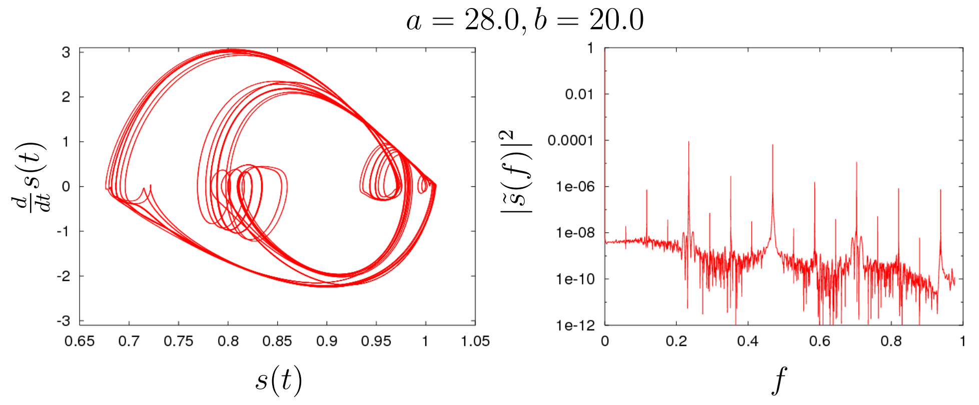


(f) power spectrum

Figure 5.21: Mackey Glass system: orbits and corresponding power spectra



(a) chaotic attractor (b) power spectrum



(c) chaotic attractor (d) power spectrum

Figure 5.22: Mackey Glass system: orbits and corresponding power spectra

5.8 Singular value analysis

This investigation method performs a ***singular value decomposition*** (SVD) [67] or ***principal component analysis (PCA)*** sometimes also denoted as ***Karhunen-Loève expansion*** after Karhunen [86] and Loève [96], who proposed this method independently from each other.

The method itself is known under different names in various research fields. Mathematically, the singular value analysis is a transformation which diagonalizes a given matrix $\underline{\underline{M}}$ and brings it to a canonical form $\underline{\underline{M}} = \underline{\underline{U \Lambda V}}$, where $\underline{\underline{\Lambda}}$ is a diagonal matrix. The roots of this method go back to the middle of the 19th century. A review of the early history of the method can be found in [157].

This method is a useful statistical technique that has found application in many fields such as face recognition or more general pattern recognition and image compression, and is a common technique for finding patterns in high dimensional data sets, where usual visualization techniques often fail or are not applicable at all.

The **AnT computation engine** uses the CLAPACK [18] subroutine `dgesvd` to calculate the eigenvalues and eigenvectors of the covariance matrix. As a consequence, a user have to install this f2c'ed version of the original Fortran LAPACK library to be able to use this investigation method.

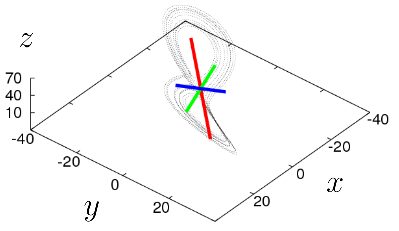
To demonstrate this method, it is applied to a simulated trajectory of the Lorenz system (see Sec. 3.4.1.1). The system was integrated numerically for 50 seconds using a fixed step size of 10^{-3}, whereby the first 40 seconds where omitted. Hence, the method is applied here to the remaining 10000 data points of the trajectory, which results in the following three eigenvalues

$$\lambda_1 = 45.35781, \quad \lambda_2 = 1.60651, \quad \lambda_3 = 0.87756 \tag{5.12}$$

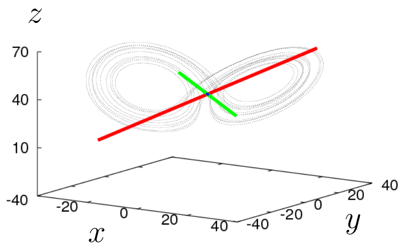
and the corresponding three eigenvectors

$$\begin{aligned} \underline{v}_1 &= (-0.57715, -0.57763, -0.57727) && (5.13) \\ \underline{v}_2 &= (0.62907, 0.13628, -0.76531) && (5.14) \\ \underline{v}_3 &= (0.52074, -0.80484, 0.28471) && (5.15) \end{aligned}$$

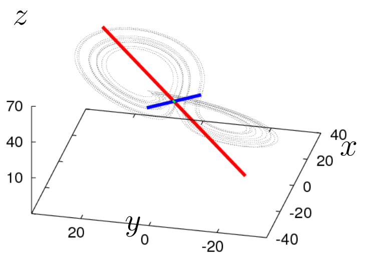
The magnitude of the eigenvalues indicate the importance of the corresponding eigenvectors or directions in the state space. To illustrate this, in figures 5.23(a) - 5.23(d) the trajectory is shown together with the three directions corresponding to the calculated eigenvectors. Hereby, the point of intersection represents the mean value of the attractor. The directions were weighted with the eigenvalues, but due to the large differences in their magnitude, the first eigenvalue was scaled by a factor of $\frac{1}{10}$. As one can see, the directions represent indeed the spreading of the data points and consequently of the underlying chaotic attractor.



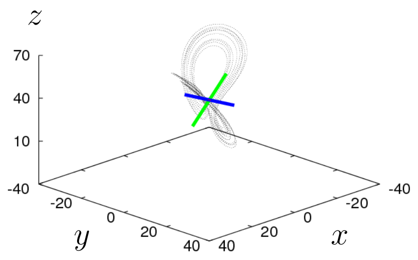
(a) $\underline{v}_1, \underline{v}_2, \underline{v}_3$



(b) $\underline{v}_1, \underline{v}_2$



(c) $\underline{v}_1, \underline{v}_3$



(d) $\underline{v}_2, \underline{v}_3$

Figure 5.23: Lorenz system: illustration of the singular value decomposition
Shown are the trajectory and the directions corresponding to the eigenvectors of the covariance matrix of the singular value decomposition. In (a) all three directions are shown, whereas in (b),(c) and (d) the viewpoint is along one of the directions causing this direction not to be seen. Parameter setting: $\sigma = 16.0, r = 45.92, b = 4.0$

5.9 Check for conditions

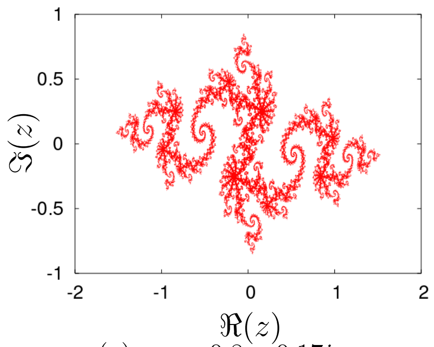
This method perform checks for some conditions and saves the time and the orbit length since the start of the simulation run, as soon as these conditions are fulfilled the first time. There are several pre-defined conditions, which correspond to some typical problems in the field of nonlinear dynamics. At the moment, the following conditions are implemented:

- **The orbit reaches a fixed point**
  This kind of condition check is suitable for instance within a scan for initial values in order to detect the basins of attraction for fixed point attractors. Another application example could be a scan for parameter values in order to detect the settings, for which the investigated dynamical system converges to a fixed point. The accuracy of the comparison, that is used in order to detect, whether the orbit reaches a fixed point or not (that means, that two subsequent states of the orbit are identical), can be set by the user (see remarks in section 5.3).

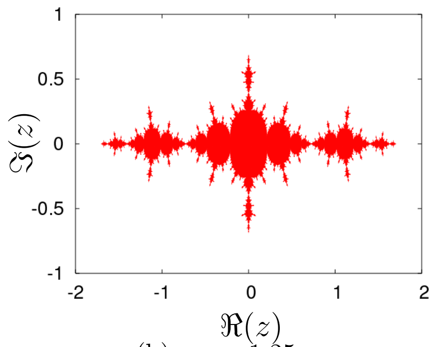
- **The orbit reaches a given point**
  This kind of condition is similar to the previous one. The user can define the state searched for. In contrast to the previous kind of condition, this point needs not to be a fixed point, but can be an arbitrary point, for instance a point on a limit cycle.

- **The orbit diverges from a given area**
  A typical situation in nonlinear dynamics is, that orbits started at some initial values converge to attractors, while orbits started at some other initial values diverge. In order to detect the initial values with convergent behavior one has to perform a corresponding state space scan. As an approximation of the convergent behavior, one can define a sufficiently large area assuming thereby, that orbits which do not leave this area, do also not diverge. In the current implementation the area in the state space can be define as a n-dimensional rectangle or as a n-dimensional circle.

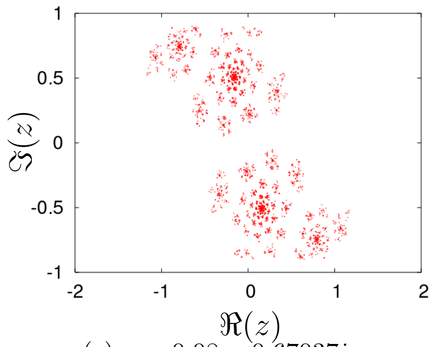
- **The orbit do not diverge from a given area**
  This kind of condition is complementary to the previous one. Accordingly, it has the same parameters and similar application areas. It can be used for instance for the calculation of generalized Julia sets,



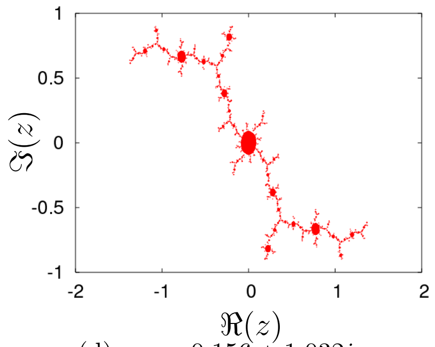
(a) $c = -0.8 - 0.17i$



(b) $c = -1.25$



(c) $c = 0.08 - 0.67037i$



(d) $c = -0.156 + 1.032i$

Figure 5.24: Julia sets: some examples

because these sets are non-wandering sets of some complex quadratic function. The function used for the figures 5.24(a) - 5.24(d) is the ***complex quadratic function***:

$$z_{n+1} = z_n^2 + c \tag{5.16}$$

with the corresponding real representation:

$$x_{n+1} = x_n^2 - y_n^2 + c_\Re \tag{5.17}$$

$$y_{n+1} = 2x_n y_n + c_\Im \tag{5.18}$$

whereby $\Re(z_n) = x_n$, $\Im(z_n) = y_n$, and $\Re(c) = c_\Re$ and $\Im(c) = c_\Im$

5.10 Symbolic sequence analysis

This method is based on the ideas from the field of the classical ***symbolic dynamics***. According to this ideas, the state space of the investigated dynamical system is split up in disjunct partitions or segments thereby defining a so-called ***covering*** of the state space. Then, a unique symbol from some alphabet Σ is assigned to each partition or segment. Consequently, each orbit of the investigated dynamical system discrete in time can be characterized by the symbolic sequence generated by this orbit. For dynamical systems continuous in time this can be done as well, whereby firstly a mapping of the continuous orbit to a discrete one has to be defined. This can be done using any kind of ***Poincaré mapping***, for instance a stroboscopic mapping. One possibility to define this stroboscopic mapping is to use the inherent discretization of the continuous trajectory caused by the integration step size. However, in most of the cases this possibility is not very efficient. The generated symbolic sequence have to be evaluated with respect to its periodicity, complexity, etc. Hence, the general approach on this field consists of to subsequent steps:

1. generation of symbolic sequences
2. evaluation of symbolic sequences

For these two steps the **AnT computation engine** supports several possibilities.

5.10.1 Generation of symbolic sequences

In most works known so far, the definition of the partitions in the state space depends on specific properties of the investigated dynamical system. It is often preferable to define these partitions depending on fixed points, manifolds or other specific features of the investigated system. Of course, the **AnT 4.669** package supports this approach and allows do define ***user-defined symbolic dynamics***. In this case a user have to implement the corresponding ***symbolic function*** in a similar way as the system function of the investigated dynamical system itself. The symbolic function has the same arguments as the system function itself and returns a string representing a symbol. Note, that this approach is most flexible and most general, compared with other techniques described below. However, it requires the user to define and implement the symbolic function.

Additionally, the **AnT 4.669** package provides two types of generic symbolic dynamics, namely the generalized $\mathcal{LR}$ ***symbolic dynamics*** and the generalized $\mathcal{PM}$ ***symbolic dynamics***. These approaches are known for one-dimensional maps and define the partitions of the one-dimensional state space according to the geometric shape of the system function. In the **AnT 4.669** software package this approaches are generalized for an arbitrary dimension of the state space. The generalization is made by applying the standard procedure for each component of the n-dimensional state space separately, so that a n-tuple of symbols is generated. Taking into account, that each n-tuple of symbols on an alphabet Σ can be interpreted as a single symbol on the alphabet Σ^n, a generalized symbolic sequence on this alphabet is created. This procedure is related to the ***band-reduction theorem***, well known from the field of ***Turing machines***.

When dealing with hybrid dynamical systems, the **AnT 4.669** package supports a generic ***symbolic dynamics for hybrid systems***. Hereby the fact is used, that the state space of hybrid systems consists per definition of several partitions. These partitions can be used for the creation of symbolic sequences as well.

5.10.2 Evaluation of symbolic sequences

Based on the symbolic sequences generated by a trajectory, several interesting quantities can be calculated. The most basic quantities hereby are the probabilities of the specific symbols, approximated by the frequency of occurrence of these symbols in the sequences. For growing length of the used sequence, the frequency of occurrence converges to the probability of the specific symbol. The probabilities approximated in this way can be used for the calculation of the entropy of the symbolic sequence. Note, that this approach can be generalized using symbolic subsequences with length larger than one instead of single symbols only. The length of the used subsequences is usually denoted as ***symbolic description level***. In practice, the complexity of the calculation grows exponentially with the symbolic description level, so that it is preferable to use this level not too large. An application example for the symbolic sequence analysis is presented in figure 5.26. For the Agnesi map (Eq. 5.5) symbolic sequences on the alphabet $\Sigma = \{\mathcal{L}, \mathcal{C}, \mathcal{R}\}$ are generated

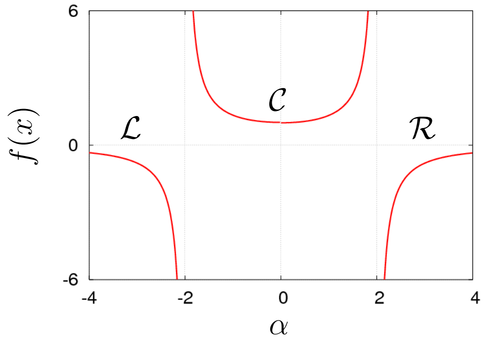
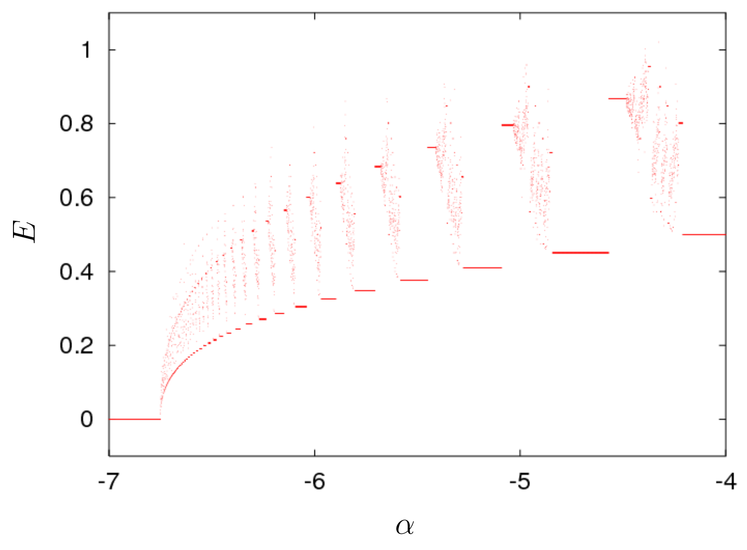


Figure 5.25: Agnesi map: system function at $\alpha = -4$ and the partitions of the state space marked with corresponding symbols

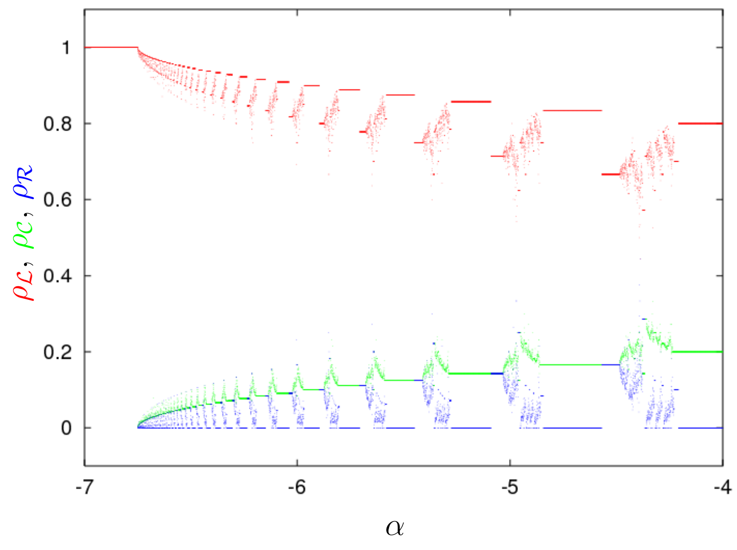
according to the following (user-defined) partition of the state space:

$$\sigma_n = \begin{cases} \mathcal{L} & \text{if} \quad s_n < -\sqrt{|a|} \\ \mathcal{C} & \text{if} \quad -\sqrt{|a|} \leq s_n \leq \sqrt{|a|} \\ \mathcal{R} & \text{if} \quad s_n < -\sqrt{|a|} \end{cases} \tag{5.19}$$

As one can see from figure 5.25, the partition is adjusted to the shape of the system function of the Agnesi map for $\alpha < 0$. Therefore the used symbolic dynamics is more suitable for the investigated system than the standard ($\mathcal{LR}$ or $\mathcal{PM}$) symbolic dynamics. In figure 5.26(a), the symbolic entropy of the Agnesi map is shown and in figure 5.26(b) the probabilities $\rho_{\mathcal{L}}$, $\rho_{\mathcal{C}}$, $\rho_{\mathcal{R}}$ of the specific symbols. The results correspond to the symbolic description level one.



(a) symbolic entropy



(b) symbolic probabilities

Figure 5.26: Agnesi map: symbolic sequence based analysis (blow-up)

5.11 Symbolic image analysis

This relatively new method is based on the ***symbolic image construction*** as described in [127, 129, 128]. The symbolic image analysis represents a unified framework for the investigation of the global structure of the vector field of dynamical systems both discrete and continuous in time. This method can be considered close to Cell-Mapping [80] and related to symbolic dynamics [1, 14, 95, 167]. Its main idea is the construction a directed graph which represents the structure of the state space for the investigated dynamical system. This graph is called the ***symbolic image*** of the system and can be considered as an approximation of the system flow. Distinct parts of this graph represent ***invariant sets*** of the flow. These invariant sets can be detected without any restrictions concerning the stability. Hence, the periodicity of stable and unstable limit cycles can be determined. A more sophisticated computational analysis of the symbolic image allows the location of the ***basins of attraction*** [130] and of the ***Morse Spectrum*** [125, 131, 126] as well.

The construction of the symbolic image for an area of the state space of the investigated dynamical system is an iterative procedure. In each step the investigated domain is divided in sub domains, so-called boxes. Each box corresponds then to a node of the directed graph to be constructed. The nodes of the graph are connected according to the vector field of the dynamical system.

Assume, that there exist two boxes I and J and the corresponding nodes i and j. For the box I its image under the vector flow $\underline{f}(I)$ is calculated. If the intersection of the box J with the image $\underline{f}(I)$ is not empty, then there exists an edge in the graph from node i to node j. Performing this procedure for all boxes of the investigated domain, one calculates in this way an approximation of the symbolic image. After that, the parts of the graph are detected, which have to be investigated more precisely. Especially the cycles of the graph represent a good choice to be investigated in more details, because they can correspond to stable or unstable periodic orbits. In the next step, the boxes corresponding to these nodes of the graph are subdivided again. The procedure is repeated with higher accuracy again, until the termination criterion is fulfilled. The termination creation is typically given either by the accuracy to be achieved or by the computer memory.

Note, that in the practice the investigated graph can contain a large number of nodes (for instance, 10^6 is a typical value in this context). Therefore, the analysis of the cycles in such graph may be a computation expensive task. For this reason, efficient algorithms, like Dijkstra algorithm for calculation of shortest paths in directed graphs [143] and Tarjan algorithms location of sets of strongly connected components [161].

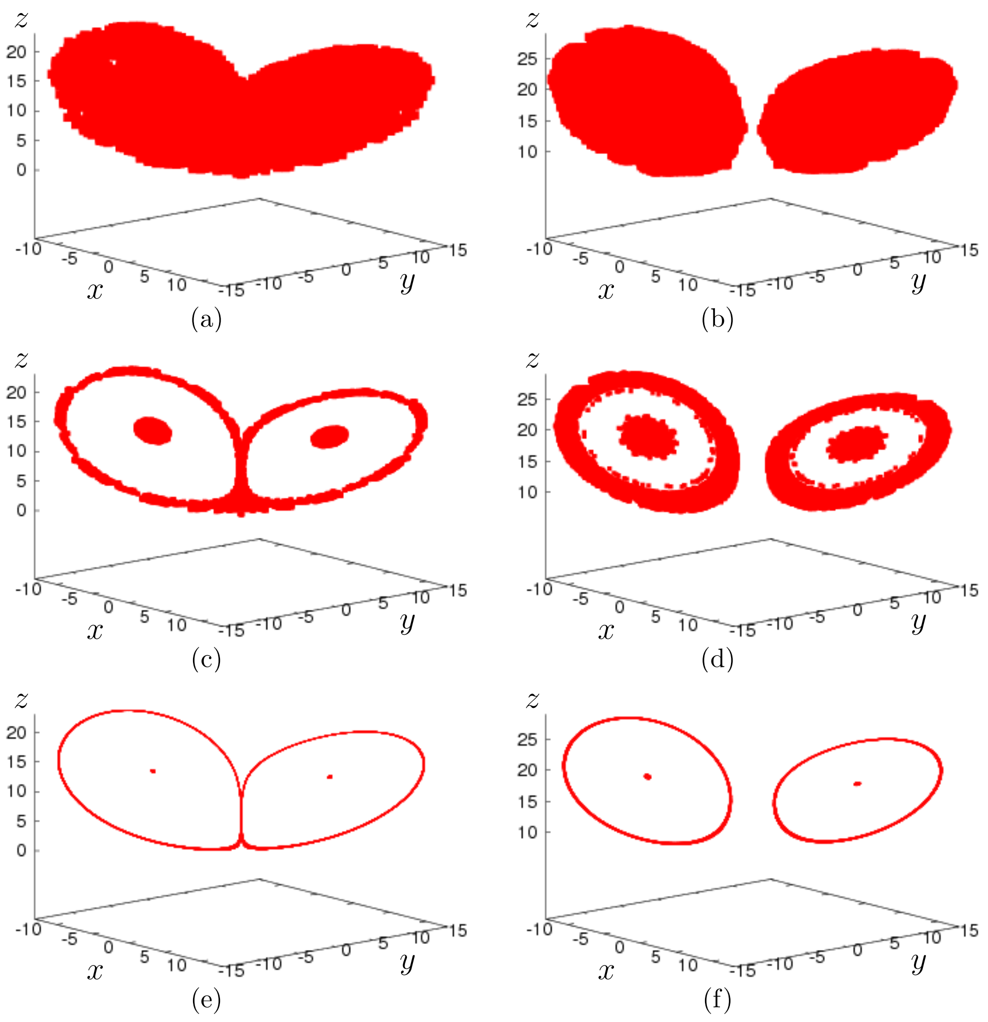


Figure 5.27: Lorenz system: symbolic image analysis

Shown are three subdivision steps of the numerically calculated symbolic image for the Lorenz system at $r_1 = 14.6$ *(left) and* $r_2 = 20$ *(right).*

5.12 Generalized Poincaré sections

The basic idea of ***Poincaré sections*** is based on the reduction of complexity of the dynamic behavior and can be considered as a procedure selecting some representative states from an orbit or trajectory. In the classical variants the orbit is assumed to be generated by a dynamical system continuous in time. The condition, selecting the states of the orbit, is typically given by the cross-section of the orbit with a given plane or in general a hyperplane. As one can see, the resulting states can be interpreted as an orbit of a dynamical system discrete in time denoted as ***Poincaré map***. Therefore the classical variant of Poincaré sections leads to the reduction of complexity by a reduction of the state space: instead of a continuous orbit in a n-dimensional state space one considers a discrete orbit in a $(n-1)$-dimensional state space. Hereby there exists no general approach, how to define the plane in such a way, that the properties of the Poincaré map are most similar to the properties of the dynamical system continuous in time and hence most representative for this system.

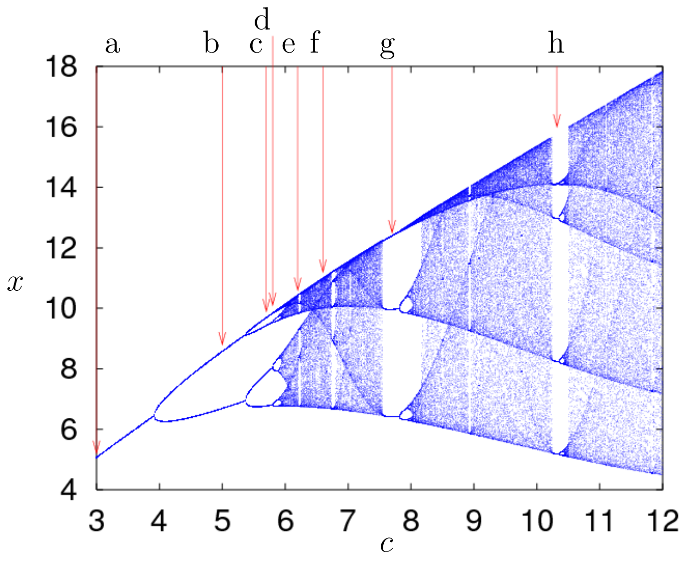
Considering the basic structure of Poincaré sections, the idea described above can be generalized. As one can see, the cross-sections of the orbit with a hyperplane represents only one possibility for the condition, which has to be fulfilled for the selection of states. In general, this condition can be arbitrary. The **AnT 4.669** simulator package supports the following types of ***Poincaré conditions***:

▶ **Cross-sections with a plane**
There are two variants for this classical type of Poincaré section:

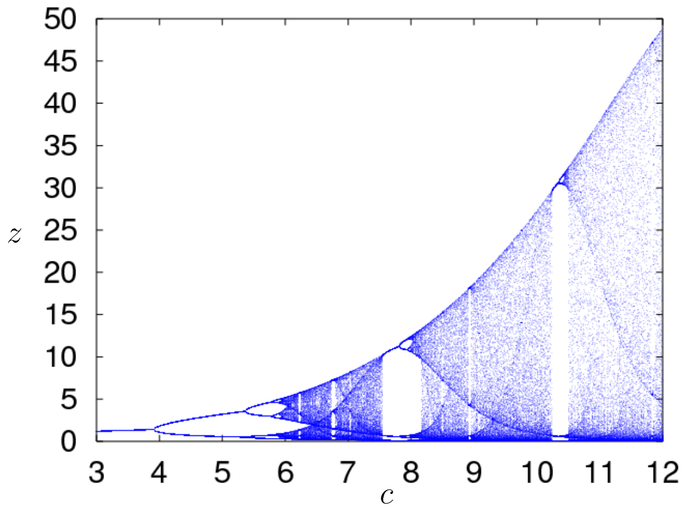
- The plane is **fixed**
- The plane is **parameter-dependent**

These two variants differ with respect to a scan run. In the first case, the plane is the same for all simulation runs. Therefore, it can be specified by its coefficients, which are fixed and have to be defined by the user. A well-known application example hereby is the Poincaré section of the Rössler system (see Sec. 11.5) with the plane $y = 0$ (see Fig. 5.28 and Figs. 5.29, 5.30).

In contrast to this, it is in many cases more suitable to define the plane dependent on some features of the investigated dynamical system, for

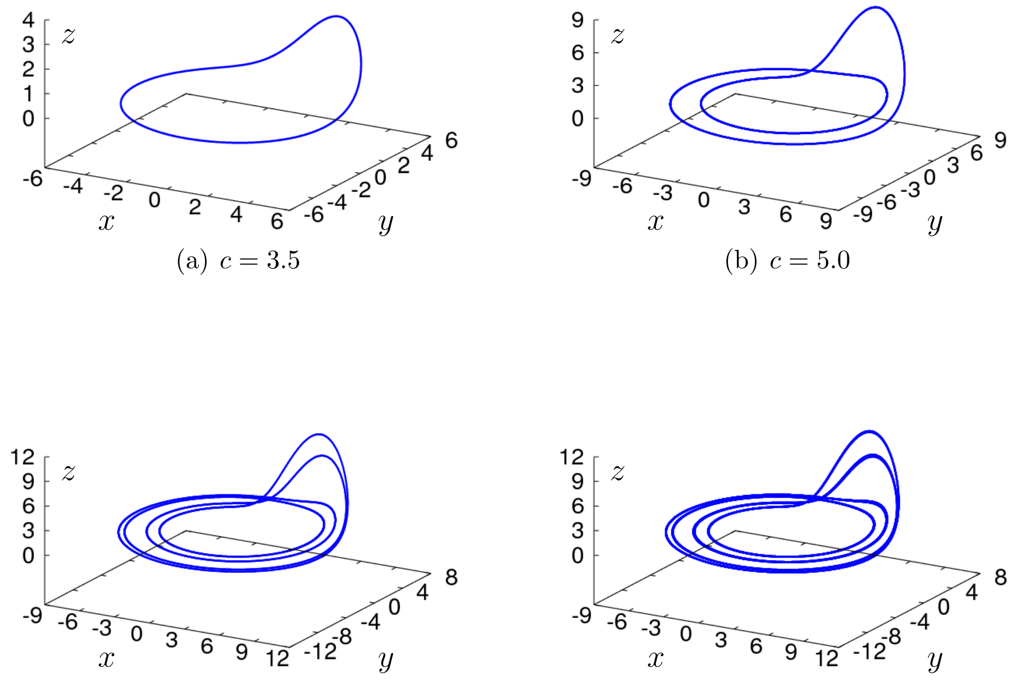


(a)



(b)

Figure 5.28: Rössler system: bifurcation diagram of the Poincaré map *Parameter setting: $a = 0.15$, $b = 0.2$. In (a), the points in the parameter space corresponding to Fig. 5.29 and Fig. 5.30 are marked.*

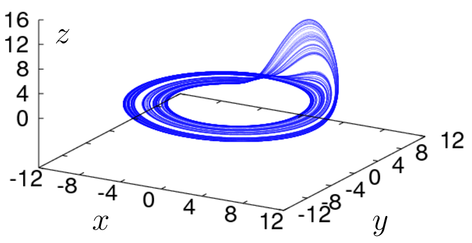


(a) $c = 3.5$ (b) $c = 5.0$

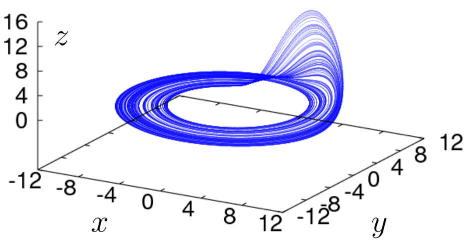
(c) $c = 5.7$ (d) $c = 5.8$

Figure 5.29: Rössler system: periodic attractors
Parameter setting: $a = 0.15$, $b = 0.2$.

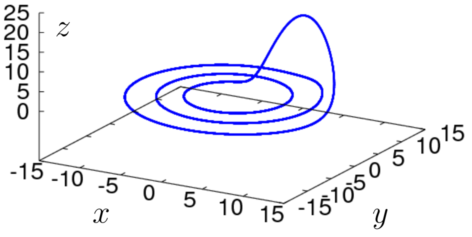
(a) *One-periodic limit cycle.*
(b) *Two-periodic limit cycle.*
(c) *Four-periodic limit cycle.*
(d) *Eight-periodic limit cycle.*



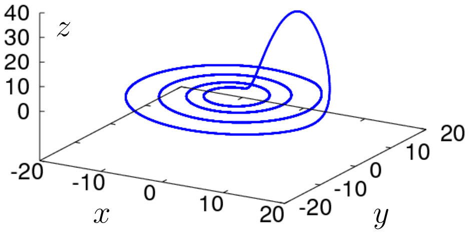
(a) $c = 6.2$



(b) $c = 6.6$



(c) $c = 7.7$



(d) $c = 10.32$

Figure 5.30: Rössler system: periodic and chaotic attractors
Parameter setting: $a = 0.15$, $b = 0.2$

(a) *Two-band chaotic attractor.*
(b) *One-band chaotic attractor.*
(c) *Three-periodic limit cycle.*
(d) *Four-periodic limit cycle.*

instance on its fixed points. These points may depend on the system parameters. Hence, within a scan run, the coefficients of the plain have to be adjusted for each simulation run separately. For instance, for the Lorenz system (see Sec. 3.4.1.1), that specific Poincaré section is representative, which is defined by the cross-section of orbits with a plane parallel to the plane $z = 0$ containing both non trivial fixed points of this system. The z-coordinate of these fixed points is given by $r - 1$. Hence, by variation of the parameter r, this plane is parameter-dependent.

- **States with local extreme values of a given state component**
This type of Poincaré section is sometimes denoted as $z_{\max}$-mapping, because it can be successful applied for the z variable of the Lorenz system. In the **AnT 4.669** simulation package there are three variants for this type of Poincaré section:
  - Using maximal values
  - Using minimal values
  - Using minimal and maximal values

- **Change of the discrete state**
For hybrid dynamical systems it is possible to define this generic type of Poincaré section. The state space of a hybrid system is divided into partitions, described by the discrete state of the system. It is well-known, that the dynamics of hybrid systems is dominated by transitions, which the system performs from one partition into another one. Therefore, the states on partition borders, where the transitions take place, are important and can be used for generating the discrete orbit.

- **User-defined Poincaré conditions**
This most general case allows the definition of arbitrary conditions. For instance, it can be preferable to define the condition not only dependent on the current state of the orbit, but also dependent on the current velocity. An application example for a dynamical systems with time delay, namely the PLL with time delay (see Sec. 3.4.2.1) is shown in Fig. 5.31.

Note, that concerning the architecture of the **AnT computation engine**, the generalized Poincaré section do not represent an investigation method in

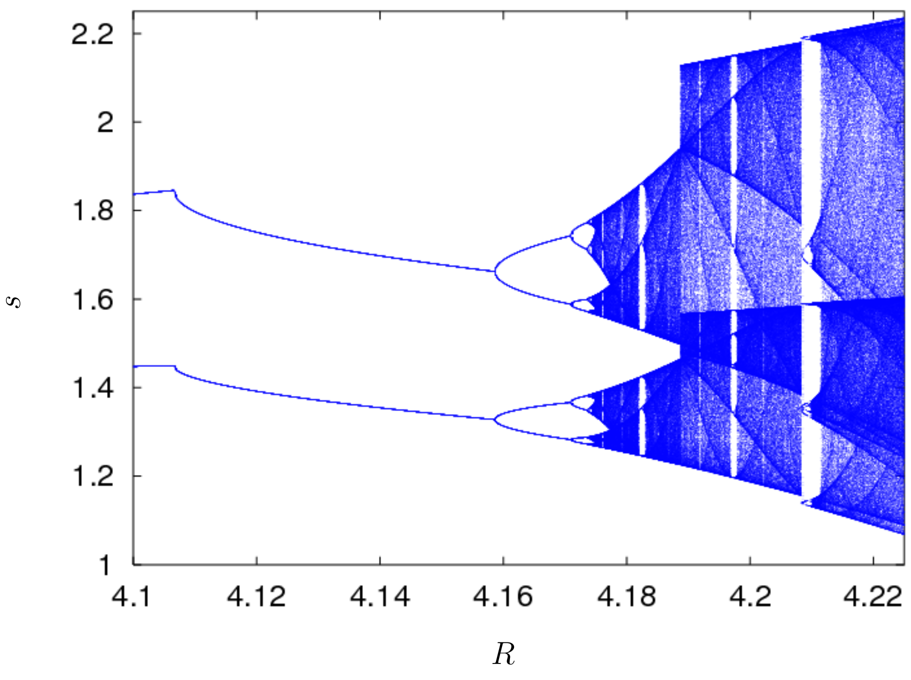


Figure 5.31: PLL with time delay: scan of a generalized Poincaré section
This bifurcation diagram was obtained using the generalized Poincaré section method, whereby the two conditions $\frac{d}{dt}s(t) = 0$ *and* $s(t) \in [1, 2]$ *define a point of the corresponding Poincaré map. A one-dimensional system parameter scan was performed and the cyclic and last 100 acyclic points of the Poincaré map are plotted. The two parabola at the bifurcation which occurs at the parameter value* $R \approx 4.103$ *indicate, that there was a symmetry-breaking bifurcation in this system at smaller values of the parameter and that in the considered parameter region two coexisting period-doubling scenarios take place. Only one of them is shown.*

the usual sense. Based on the concepts of an abstract iterator and a proxy, the simulation of the dynamical system inside the generalized Poincaré section can be sourced out from the iteration machine. A simulation step of the Poincaré map, containing the simulation of the dynamical system inside until the Poincaré condition is fulfilled, is performed within the proxy. Outside the proxy only an orbit discrete in time is observable. Therefore all investigation methods, which are applicable for standard maps, can be applied for Poincaré maps as well.

Chapter 6

Simulating and investigating dynamical systems

6.1 Motivation

In this chapter, the basic architectural principles and concepts of the **AnT 4.669** software package and especially the **AnT computation engine** will be presented. These principles and concepts allow not only the support of the broad spectrum of system classes presented in chapter 3, but are the reason for the great flexibility of the software as well. The aim here is not to explain the architecture in full detail, but to provide an overview and to point out the generic characteristics of some of the concepts, which turned out to be so general, that they are not specific for the **AnT 4.669** software package and the simulation of dynamical systems, but can adapted to other simulation problems or tasks as well.

As already mentioned in chapter 1 in section 1.2.1, it is inherent to computer-based simulations that there exist a data model and an execution model corresponding to the specific simulation problem or task. Of course, this holds also for the simulation of dynamical systems, which can be performed by the **AnT computation engine**. As it was the goal of the software, the data and execution model are specifically designed not only for the simulation of dynamical systems, but for their investigation as well.

6.2 The data model

As the notion data model implies, it is the part containing all the relevant data and information which is necessary to perform the possible and supported simulation and investigation tasks. There are two important aspects which should be mentioned here.

1. Distributed and centralized data
   Because the information contained in the data model is required in different parts of the software and at different phases of the simulation it is not located at a single position in some sort of a data container, but is distributed over large parts of the software. One can think here for instance about the specific information which is needed for the integration of dynamical systems continuous in time according to Runge-Kutta type integration methods or schemes. These integration schemes require coefficients, which are either stored in specific arrays, denoted as Butcher arrays, or are even hard-coded. However, this information is so specific, that it makes no sense to store it together with other data in some sort of general data container. Nevertheless, whenever there is no need to distribute the information, it should be stored in a general data container or module at one location. Both, the distribution and centralization of data and hence the data model is not typical for the **AnT 4.669** software package, but is quite common in many software applications and is part of a suitable software structure or architecture which helps to keep large software projects maintainable.

2. Dynamic and static data
   The data model has dynamic and static components as well. This is in some sense obvious because on the one hand there is static information required for the specification and the control of the complete simulation task, which must not change during the simulation run. On the other hand, there is information or data which will be produced or created during the simulation itself and is of course dynamic.

Most of the information in the data model is provided by the initialization file (see Sec. 8.2.1). Note, that the graphical user interface (see Chap. 8) guides and supports a user of the system to create or manipulate initialization files in a convenient way. Hence, the initialization of the **AnT computation engine** still depends on an initialization file, because each simulation task has to be specified by a corresponding initialization file. Additionally, as already mentioned, there exists also information which is not user provided but hard coded. The abovementioned coefficients of the implemented Runge-Kutta type integration methods for dynamical systems continuous in time for instance, are a typical example of that kind of hard coded information or data. According to the intention of the **AnT 4.669** software, to support the simulation and the investigation of dynamical systems, the data model is split into four main parts, which correspond in some sense to the four main sections of the initialization file (see Chap. 8, Secs. 8.2.1.1 - 8.2.1.4). These four parts of the data model are: the dynamical system data, the iterator data, the investigation methods data, and the visualization data.

Currently, the data model realized in the **AnT computation engine** is slightly different and consists of the three main parts `DynSysData`, `IterData` and `ScanData`. However, this distribution has historical reasons and will be changed in one of the next releases of the software package. Therefore, the new and more suitable structure will be presented in the next sections 6.2.1 - 6.2.4.

6.2.1 The dynamical system data

The dynamical system data itself consists of two parts: the general part and the specific part containing information corresponding to the different supported classes of dynamical systems. The general part holds information about the state of a dynamical system and the orbit which is implemented as a cyclic array of states, so that a part of the trajectory can be stored in the orbit. During the simulation or iteration, the orbit is updated in a cyclic manner, so that it contains always some of the already calculated states of the dynamical system up to a certain time in the past. In the general part of the dynamical system data, the initial states and system parameters are stored as well. Note, that in the case of initial state or system parameter scans some of the initial states or system parameters are part of the iterator data according to their change during iteration.
The specific part of the dynamical system data contains for instance information about the step size and the method for the numerical integration of dynamical systems continuous in time, the size and granularity of the grid for indexable systems, or the delay time respectively the recurrence level for systems with memory.

6.2.2 The iterator data

This part of the data model contains data or information, which is not only necessary to perform an iteration step, that means the transition from the current state to the next one, but the complete simulation as well. Here, the number of iterations, and the scan specific information is stored. It is a direct consequence of the architecture, that a single iteration step, a complete iteration, i.e., an iteration-run and even a complete scan, i.e., a scan-run is treated in an almost identical manner (see Secs. 6.3.4, 6.3.5.1, 6.3.5.2 and Figs. 6.2, 6.3, 6.4). Therefore, the information necessary to perform an iteration step, an iteration-run or a scan-run is contained in the iterator data part of the data model.

6.2.3 The investigation methods data

In this part of the data model information about the investigation methods is included. Firstly, of course this is information about the investigation methods activated in the current simulation. Additionally, each investigation method needs further specifications which is contained in the investigation method data as well. Two examples may illustrate this:

1. Lyapunov exponents
2. Box counting based methods

6.2.4 The visualization data

This part of the data model is in principle part of the investigation methods data presented in the previous section 6.2.3, because the visualization of trajectories and hence solution curves of a simulated dynamical system is considered to be some sort of investigation. However, due to the possibilities of the OpenGL based visualization, a lot of control information has to be specified in the data model. Consequently, this large part of the data model has been encapsulated in a separate part, the visualization data.

The visualization data contains information about the number of graphic windows and the type of visualization which could be a two-dimensional or a three dimensional representation. Furthermore, the visualization data contains information about the specific type of representation (time-series, phase-portrait, ...) together with all the basic graphics controlling information about additional objects, axes, mesh, color and so on.

6.3 The execution model

6.3.1 The transition concept

The concept of a transition already mentioned in Sec. 3.2 plays an essential role in the definition of a dynamical system in this work but it is important also in many design parts of the **AnT computation engine**. Therefore it will be discussed in more detail. The concept is realized in the software package by the most important abstract class denoted as `Abstract Transition`. Many subclasses are derived from this abstract class, among them the subclasses which are used for a single iteration step (`IterTransition`), a complete iteration run (`IterMachine`) and a complete scan run (`ScanMachine`).

6.3.2 Abstract transition

The concept of an abstract transition is introduced in order to describe an entity, which transforms one state into another. Therefore, the only functionality, which has to be supported by each instance of this abstract transition, is a method (called `execute`), which performs this transformation. As one can see, the basic aim of an abstract transition is formulated without any relation to a specific task. Hence, abstract transitions can be considered as a basic component for the execution model of any simulation tool.

For the creation of an execution model for a specific application, there are two types of abstract transitions:

- **generic** or **structure-defining** transitions
- **non-generic** or **task-specific** transitions

The corresponding inheritance hierarchy is shown in figure 6.1. Typical examples for transitions of both types are discussed in the following sections.

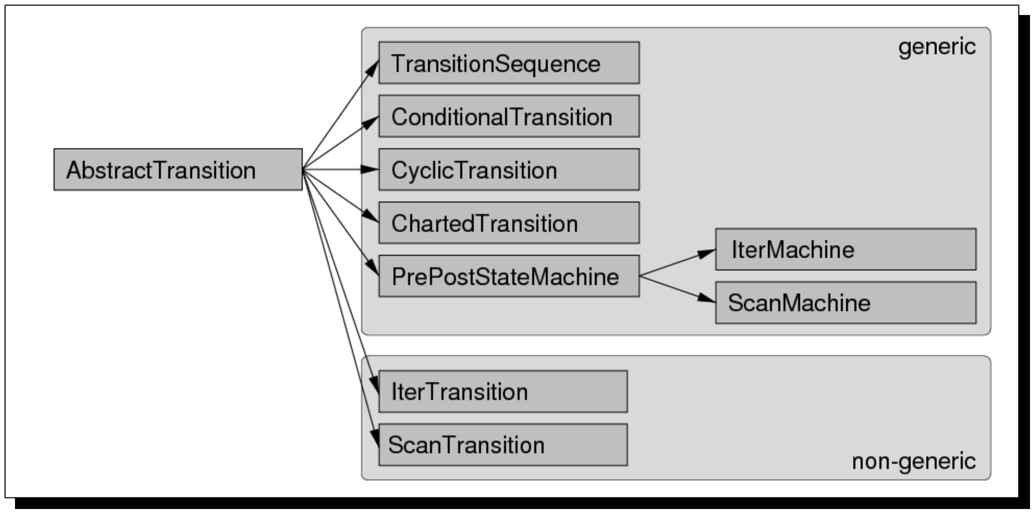


Figure 6.1: Inheritance hierarchy of the class `AbstractTransition`

6.3.3 Generic transitions

Within the design phase of the **AnT 4.669** software package it turned out, that the following generic transitions represent some common ***execution patterns*** on the field of numerical simulation:

- ***Transition sequence***
  This kind of transition represents a sequence of abstract transitions. The execution of the sequence is a generic operation, consisting of the subsequent execution of the transitions within the sequence.

- ***Conditional transition***
  The execution of conditional transitions is performed depending on some condition.

- ***Cyclic repeated transition***
  The execution of these transition is performed repeatedly as long as some ***termination criterion*** is fulfilled.

- ***Charted transition***
  The execution of these transition is performed depending on a state chart.

Note, that the tasks performed by the described transitions are not specific for a simulation tool for dynamical systems. Using these generic transitions, a general framework for the execution model can be created. This framework can be considered as a skeleton of the execution model which has to be completed with task-specific transitions.

6.3.4 The iterator concept

The main task of an iterator, is to calculate the next state of a dynamical system based on the current state. For dynamical systems with memory, also previous states are involved. Hence, an iteration step represents a transition from the current state to the next one and an iterator represents an abstract transition in the sense of section 6.3.2.

During the design phase of the **AnT 4.669** simulation package it was aimed to support many different classes of dynamical systems. Especially, dynamical systems continuous and discrete in time, with and without memory, spatial homogeneous and spatial inhomogeneous, etc, are supported by **AnT computation engine**. Therefore, several iterators and integrators are required, because several system classes require several approaches for the calculation of the next state. For instance, the same integration scheme should be implemented differently for ODEs, DDEs and FDEs, due to the different number of states involved into the calculation. Note, that a general implementation for all these system classes is possible but would be very inefficient.

It turned out, that a large number of system classes can be iterated / integrated using the following classes of iterators:

- ***map iterator***
  Because the iteration scheme is very simple in this case, there is exactly one implementation of this iterator. The next state is determined directly by the evaluation of the system function. The following classes of dynamical systems can be iterated using the map iterator:

  - ordinary maps
  - recurrent maps
  - hybrid maps

- stochastic maps
- Poincaré maps
- CMLs
- external data

▶ ***ODE integrators***
There are a large number of different ODE integrators, which implement several integration schemes. There are one-step and multi-step method, explicit and implicit methods, predictor-corrector methods, etc. All these methods can be used with fixed or adjustable integration step size. The following classes of dynamical systems can be iterated using ODE integrators:

- ODEs
- hybrid ODEs
- CODELs
- PDEs

▶ ***DDE integrators***
The **AnT computation engine** contains also several DDE integrators. At the moment, they are used for the following two system classes:

- DDEs
- hybrid DDEs

In future expansions it is planned to realize such system classes as CDDELs and PDDEs. As soon as implemented, these system classes will be integrated with the available DDE integrators as well.

▶ ***FDE integrators***
This class of integrators is currently used only for one class of dynamical systems, namely:

- FDEs

Future expansions of the **AnT computation engine** with respect to related system classes, using the same integrators, are planned.

▶ ***Stochastic ODE integrators***
Due to the inherent stochastic process, the integration schemes for

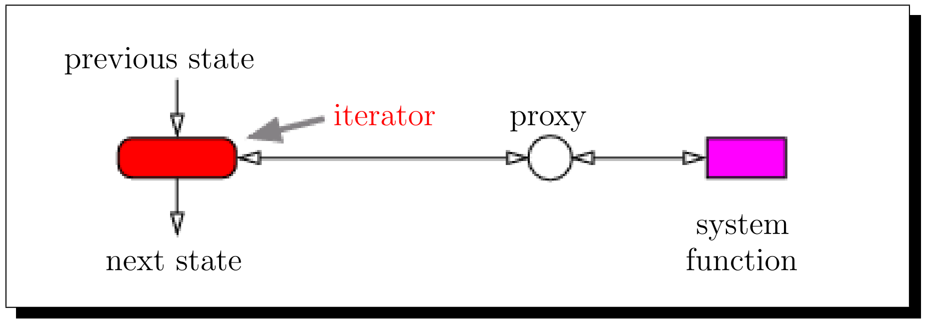


Figure 6.2: The iterator: schematic representation

ODEs have to be adjusted. This might lead to the reimplementation of the corresponding integrator.

As one can see, the same iterator or integrator will be used for several classes of dynamical systems, which can be considered as a family of dynamical system classes. For instance, maps, Poincaré maps and CMLs belong to the family of dynamical systems, which can be iterated with a map iterator. Obviously, DDEs and FDEs do not belong to this family. Note, that this was requirement during the design of the **AnT computation engine** because the implementation of the specific integrators for each class of dynamical systems would lead to an enormous code duplication and had to be avoided. However, from the mathematical point of view, all classes within one family are defined in different ways, and especially the number and types of arguments of the system function differs as well. For instance, a state of a map is a simple real-valued vector, and a state of a CML is a complex cellular structure.
From the software engineering point of view, this correspond to different interfaces of system functions, which have to be mapped onto the same interface within an integrator. In order to cope with this problem, special adaptors (called proxies) are required, as shown in figure 6.2.

6.3.5 The machine concept

Machines can be seen as basic execution patterns specific for simulation of dynamical processes and especially dynamical systems. They consist of generic transitions and represent therefore an abstract transition as well. The basic idea used here is, that several activities on the field of simulation consist of three phases, namely:

1. initialization of a task
2. processing of this task as long as necessary
3. finishing this task

A direct realization of this scheme by generic transitions leads to a transition sequence, consisting of three transitions, whereby the second one is typically a cyclic repeated transition. This general execution pattern is realized within the **AnT 4.669** project as an abstract class ***PrePostStateMachine***. In the following the three parts of the transition sequence mentioned above are denoted as **pre**-, **during**- and **post**-part of the corresponding machine. Note, that each of these parts may perform several activities and therefore may represent a transition sequence. Two realizations of the PrePostStateMachine, namely the Iter- and the ScanMachines, are described in the following sections 6.3.5.1 and 6.3.5.2.
Note, that several parts of the investigation methods have to be executed at several phases mentioned above. Therefore, each investigation method has to be implemented as a collection of transitions. If required, these typical non-generic transitions can be added to the current instantiation of the execution model in such a way, that they will be automatically executed at correct times. An example for the splitting of the investigation methods into single transitions is discussed in section 6.3.5.3.

6.3.5.1 The IterMachine

The goal of the ***IterMachine*** is to perform a simulation run, i.e., the simulation of a dynamical system with fixed settings.

In the **pre**-part of the IterMachine several initialization activities have to be performed. Especially the trajectory has to be initialized with the initial values. For classes of dynamical systems, where these values are not given explicitly, they have to be calculated. Typical examples here are the classes of dynamical systems continuous in time with memory like delay differential equations. The timer (a data entity, containing the information about the current simulation time) has to be initialized as well. Depending on the currently activated investigation methods the data of these methods have to be initialized as well.

The main task performed in the **during**-part of the IterMachine is the calculation of the trajectory. Therefore, one transition always located here, is the iterator. After the iterator performs an iteration step, the time must be incremented. This is done by an update of the timer. Additionally some parts of the investigation methods, which have to be executed in each iteration step, are executed in the during-part of the IterMachine as well. A typical **termination criterion** of the IterMachine is given by the timer itself. In most cases the simulation has to be performed for a given time, i.e. a certain number of iteration steps. However, in some cases the simulation can be terminated ahead of time, for instance if the orbit diverges.

In the **post**-part of the IterMachine typically transitions of the investigation methods are executed, which evaluate the calculated orbit and write the results of the evaluation. As typical examples here one can consider the corresponding parts of the period analysis or of the spectral analysis.

A schematic representation of the IterMachine is shown in Fig. 6.3. Note, that this complex structured object remains an instance of the abstract transition and therefore can be used as an abstract transition within an arbitrary other execution pattern.

6.3.5.2 The ScanMachine

The goal of the ***ScanMachine*** is to perform a series of simulation runs of a given dynamical system with various settings.

In the **pre**-part of the ScanMachine some initialization activities are performed, which have to be done before all following simulation runs. Most of them are related to the initialization parts of the investigation methods.

The main task performed in the **during**-part of the ScanMachine is a single simulation run. Therefore, one transition, which is always located here, is the IterMachine. Additionally, some parts of the investigation methods, which have to be executed after each simulation run, are executed in the during-part of the ScanMachine as well. Note, that the execution of this parts here is semantically equivalent to their execution in the post-part of

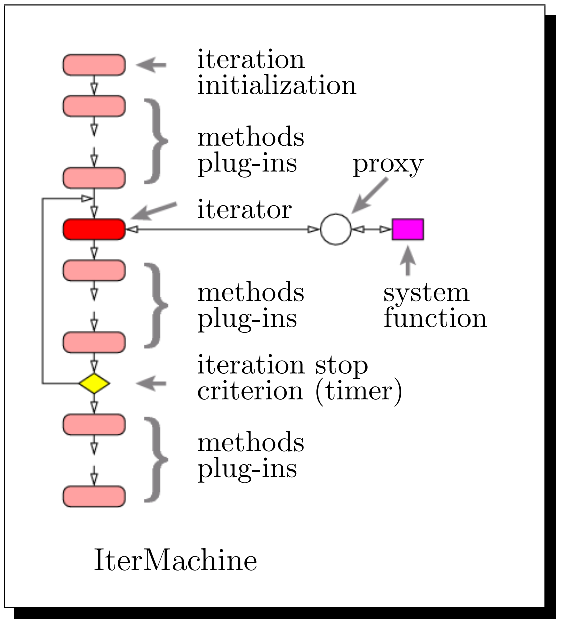


Figure 6.3: The IterMachine: schematic representation

the IterMachine. The termination criterion of the ScanMachine is given by the required number of simulation runs.

The activities performed in the **post**-part of the ScanMachine are mostly related to investigation methods, which are based not only on a single simulation run, but on results of several simulation runs. An example for such an investigation method is the region analysis. Figure. 6.3 shows a schematic representation of the ScanMachine. Like the IterMachine, the ScanMachine is still an instance of the abstract transition and can in principle be used as an abstract transition within an arbitrary other execution pattern.

6.3.5.3 Splitting of investigation methods

As an illustrative example for the splitting of investigation methods one can consider the calculation of the mean value for a given sequence of numbers. Mathematically, one have to sum up all values and to divide the result by the total number of values. In any software realization, one has to use a summation variable, which has to be set to the value zero. This represents

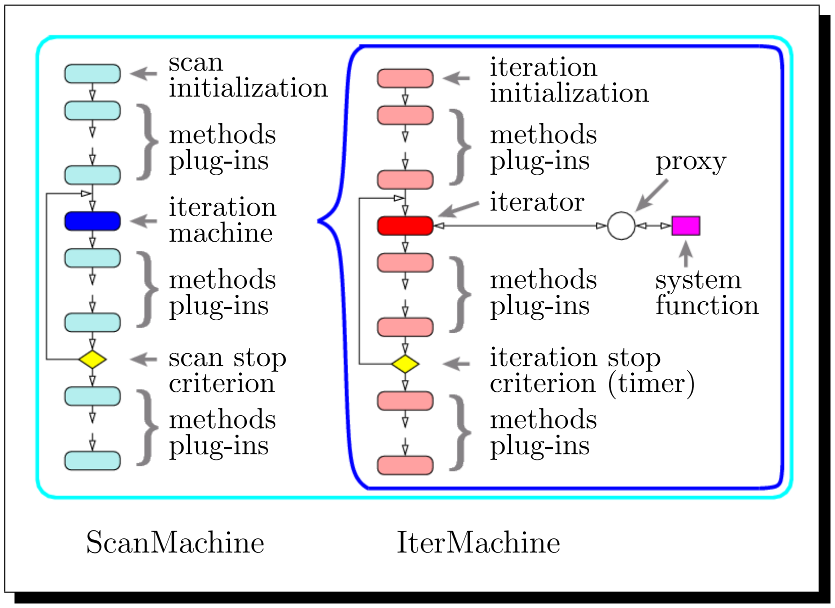


Figure 6.4: The ScanMachine: schematic representation

the initialization part of the calculation. After that, the summation is done in the cyclic repeated part of the calculation. In the final part of the calculation, the division has to be performed. In order to integrate this algorithm into the execution model of the **AnT computation engine**three corresponding transitions have to be implemented. Accordingly, the first transition should be added to the pre-part of the IterMachine, the second one to the during-part, and the last one to the post-part. In this way the calculation of temporal averaged mean values for trajectories is implemented.

6.3.6 Conclusions

The presented execution model, used for the implementation of the **AnT computation engine** turned out to be generic for many applications on the field of numerical simulation. The concept of transitions represent a generic framework and can be used in many application areas of numerical simulation. This framework can be considered as an analog to a programming language for the creation of execution models for several applications in this field. The table 6.1 shows the relationship between several generic transitions and corresponding structures of common programming languages.

transition	statement
transition sequence	sequence of statements
conditional transition	`if` statement
cyclic repeated transition	`loop` statement
charted transition	`switch` statements

Table 6.1: Relationship between transitions and instructions

The concepts of the Iter- and ScanMachine represent a general execution pattern for several applications related to the simulation of arbitrary dynamical processes. With respect to the analogy to programming language, the machines represent a standard template.

6.4 Integration of dynamical systems continuous in time

The numerical solution of an initial value problem of dynamical systems continuous in time is done by calculating an approximation to the exact solution from the initial value $\underline{s}(t_0)$, representing the start state to an end state $\underline{s}(t_1)$ at a time $t_1 > t_0$. The exact solution is defined by the trajectory or orbit

$$\underline{p}(t),\ t \in [t_0, t_1] \quad \text{with} \quad \underline{p}(t_0) = \underline{s}(t_0) \tag{6.1}$$

which can be considered as a path in the state space of the dynamical system. There are two reasons, why the numerical solution is only an approximation to this exact path. The first reason is due to the discretization, which is necessary for the calculations to be done in a digital computer system. Due to this discretization, the numerical solution of a dynamical system continuous in time is in fact represented by a system discrete in time. Hence, while the exact solution is usually a smooth path in the state space (exceptions are grazing, sliding, stick-slip, impact, corner collision, . . .), the approximation is not.

6.5 Simulation of generalized Poincaré sections

As an interesting example for the usage of the software architecture described above one can consider the implementation of the generalized Poincaré sections. Like standard maps, Poincaré maps represent dynamical systems discrete in time. However, in order to calculate the next state of the trajectory for these systems, one has to integrate another dynamical system, typically continuous in time, until the Poincaré condition is fulfilled (see section 5.12). Therefore, this dynamical system continuous in time is denoted as dynamical system inside the Poincaré sections. As one can see, the software realization of the Poincaré maps has to combine the integration of the dynamical system inside with the iteration of the map discrete in time. Also the data models of these two dynamical systems are different and have to be combined in an appropriate way. The **AnT computation engine** supports this as follows:

- The data model of the Poincaré map is identical with the data model of standard maps. If required by some investigation method, the data model can be extended by a cyclic buffer containing the time intervals, which the system inside is integrated between two subsequent points of the Poincaré map. However, for the iteration itself this extension is not necessary.

- The standard map iterator used for the iteration of the Poincaré map is connected with a proxy, which is responsible for the integration of the dynamical system inside. In order to perform this task, this proxy (called Poincaré map proxy) contains a complete data model of the dynamical system inside. Additionally, the proxy contains a partial instantiated execution model of the dynamical system inside as well, namely the IterMachine for this system as shown schematically in figure 6.5.

- The termination criterion of the IterMachine inside the Poincaré map proxy is given by the Poincaré condition. As already mentioned in section 5.12, this condition can be not only the cross-section with a hyper-plane, as in the case of classical Poincaré sections. An arbitrary condition, which selects some specific states from a continuous orbit, can be used as well.

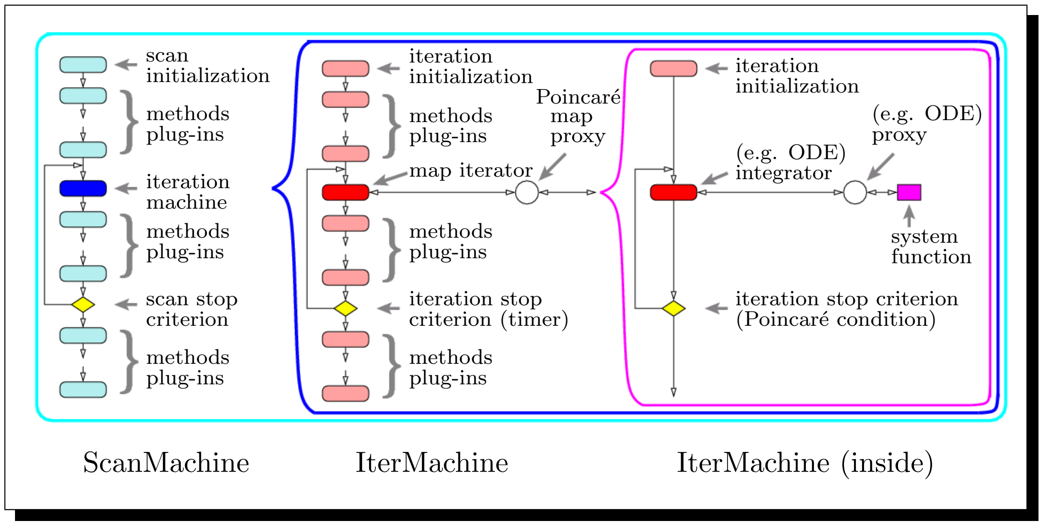


Figure 6.5: ScanMachine for the simulation of a Poincaré map
The system function of a Poincaré map is given by a complete IterMachine containing a dynamical system inside. The Poincaré condition defines the termination criterion of the IterMachine inside.

- The system function of the dynamical system inside is connected to the corresponding proxy. The fact, that this proxy is connected to the iterator used in the IterMachine within the Poincaré map proxy is transparent for the user.

The described architecture is shown in Fig. 6.5. Note, that this architecture contains no restrictions about the type of the dynamical system inside. This system can be not only continuous but also discrete in time. In principle it can be even a Poincaré map, although there is no known application field for Poincaré maps within Poincaré maps until now.

Chapter 7

Distributed computing

7.1 Why distributed computing?

In the last few years, ***distributed computing*** or ***grid computing*** has become a new paradigm in the field of ***supercomputing*** or ***high-performance computing***, which is illustrated in the statistics of the ***TOP 500 list*** [163] presented in table 7.1.

The main reasons for this trend are:

- development of fast network technology
- cheap hardware
- enhanced single processor performance
- development of 64 Bit architectures
- widespread availability of computing sites
- enhanced robustness and reliability of the hardware

Distributed computing is a powerful and cheap alternative to other supercomputing architectures and allows high-performance computing at many places in contrast to the sparse supercomputing sites. Either because relatively cheap clusters are available or because existing workstations or personal computers are connected using fast networks and can be used for grid computing. As a consequence of this hardware based trend, the development

Architecture	1998	1999	2000	2001	2002	2003
Cluster	0.4	1.4	5.6	8.6	18.8	41.6
Constellations	3.4	13.2	23.0	28.6	40.6	25.4
MPP	45.2	51.6	69.2	51.4	40.6	33
SMP	51.0	33.9	2.2	11.4	-	-

Table 7.1: Statistics of the TOP 500 list of supercomputing sites
The shares are given in %. Clearly one can see, how the importance of the cluster sites increases compared with the MMP and especially the SMP architectures.
MMP: massively parallel multiprocessor machines
SMP: shared memory multiprocessor machines

of software which can be distributed on many computing nodes is increasing rapidly.

As mentioned in chapter 4, it often occurs during the analysis of the properties and characteristics of dynamical systems, that one is interested in scans, that is the investigation of the influence of some quantities, for instance the system parameters, on the considered characteristic properties. These scans may be one-, two- or even multi-dimensional and therefore may be very time consuming. Especially when investigating complex systems, the execution of such scans required high-performance computing power. It is a great advantage of scans, that they are mostly inherently parallel. This is due to the fact, that the simulation runs at a certain parameter value are independent from simulation runs at other parameter values. As a consequence, such problems can be solved ideally on a cluster or even a heterogeneous environment with for instance many different workstations as computing nodes.

This is the reason, for the development of a network variant of the **AnT computation engine**. The details about the ***client/server*** based architecture are presented in the next sections.

7.2 The client/server architecture

7.2.1 Overview of the client/server extension

7.2.1.1 Modes of operation

In this section it will be described how the **AnT computation engine** is extended in order to make distributed simulation of dynamical systems possible. ***Distributed simulation*** means that several nodes on a network (but also several processors that are part of one multiprocessor system) collaboratively work on one simulation task. Without this extension, the **AnT computation engine** is a program that always runs on a single node, independent of all other nodes on the network. Subsequently this non-network-aware ***mode of operation*** will be called ***standalone mode***.

In order to achieve distributed computing, there has to be a different run-mode, that is in some way aware of what other nodes are doing. If there was only one type of network-capable run-mode for the **AnT computation engine**, this would imply a ***peer-to-peer*** solution, whereby all instances that are running would be of the same kind. Sometimes such architectures have great advantages, but almost always a pure peer-to-peer topology leads to the problem that coordination between instances is very difficult. Therefore the possibility of a peer-to-peer variant of the **AnT computation engine** was not pursued. Instead a client/server solution was chosen - this means that there are two different variants of the **AnT computation engine** that are network-aware: the ***network client*** denoted as ***AnT client*** and the ***network server*** denoted as ***AnT server***.

This means that altogether there are now three different modes of operation or run-modes for the **AnT computation engine**:

- standalone
- client (AnT client)
- server (AnT server)

In the following, the network-aware run-modes client and server will be discussed in greater detail.

7.2.1.2 How client/server operation works

The client/server mode of the **AnT computation engine** can be used only for scans (see Chap 4). Because the need to distribute work arises mostly due to the huge amount of computation necessary for scan-runs, this is no big constraint.

In networked operation, the run-mode server has to be used on exactly one node - this ***AnT network server*** coordinates only the work that has to be done, it does not perform calculations for scan-points, but assigns scan-points to the clients. After a client has completed its share of work, the server collects the results that the client reports (see Fig. 7.1). However, besides the distribution and collection of scan-points, the AnT server performs the calculations that are located in the ***ScanMachine.post transition***. These calculations usually occur, when investigation methods require a ***post-scan processing***. A typical example here is the region analysis investigation method. Hereby, the region analysis itself is a post-scan processing task which is done in a final step after all the periods are detected.

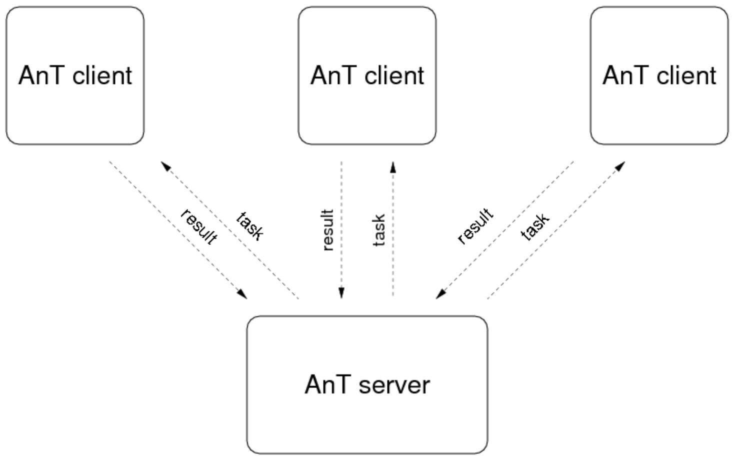


Figure 7.1: Client/server operation of AnT

7.2.2 Flow of control and distribution of work

7.2.2.1 Modification of the scanNext transition

The `ScanMachine` (see Sec. 6.3.5.2) of the **AnT computation engine** has a central transition denoted as `scanNext`. In standalone mode, this transition selects the next scan-point to investigate. It seems logical to modify the `scanNext` transition for networked operation: the server has to choose the next scan-points for the clients and send them the corresponding information, but does not actually work on them itself. On the other hand, a client cannot determine by itself which scan-point is next - it has to query the server for the next scan-point. To this end, there are now three different variants of the `scanNext` transition - one for each run-mode. The standalone variant remains unchanged, the client and server variants will be described in the following sections.

7.2.2.2 Network communication on the client

On client-side, the `scanNext` transition only queries the server for new scan-points to work on as illustrated in figure 7.2.

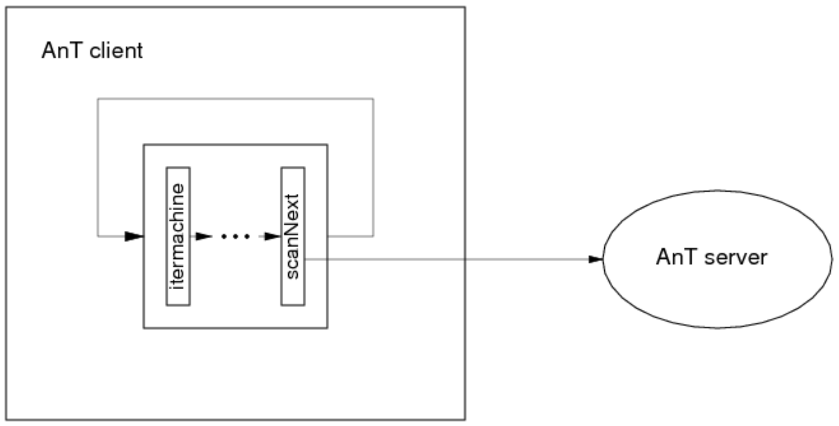


Figure 7.2: The `scanNext` transition on the client

7.2.2.3 Network communication on the server

On the server-side, the necessary changes are more profound: not only is there the need to change the `scanNext` transition, but also the entire flow of control needs to be changed. The main reason for this is that the server needs to handle scan-points in a very different way from how they are assigned by the **AnT computation engine**. For example on the server it can be necessary to assign a scan-point more than once (reassigning a scan-point) when no result gets reported (for example when the client that worked on this scan-point has crashed). So on the server the `scanNext` transition gets executed just once and handles the entire scan internally. This `scanNext` transition only does management of scan-points and handles communication with the clients. This means that there is no `IterMachine` on the AnT server (only the `scanNext` transition gets executed once in the during part of the server's `ScanMachine`). However, `ScanMachine.pre` and `ScanMachine.post` exist only on the server and get executed there (for example the regions analysis method described in section 5.4 is located in `ScanMachine.post` of the AnT server).

7.2.3 Handling output from investigation methods

Apart from the communication that is necessary between client and server, there is another form of communication that has to be changed within the **AnT computation engine** in order to be able to run in client/server mode: So far, investigation methods opened files themselves and just wrote their output into these files. While this works in standalone mode, it clearly will not work in client/server operation. There is the need to handle the output of investigation methods differently depending on the run-mode. On the other hand, the implementation of investigation methods needs to be as simple and clean as possible.

Therefore an abstraction layer for the output of investigation methods was created: the class `IOStreamFactory` provides a common interface to be used by investigation methods, with which those can output their data independently of the run-mode. In order to implement the two different kinds of output, two subclasses `LocalIOStreamFactory` and `NetIOStreamFactory` exist - depending on the run-mode, the right one of these classes is used by the investigation methods (which are not aware of the current run-mode themselves).

The main operations of the interface of `IOStreamFactory` are:

- `getOStream(fileName)`:
  This method opens a stream for output of data and returns an `ostream` object. This `ostream` can be used to output data in the standard C++ way.

- `commit()`:
  All data that has been written to an `ostream` that was created using `getOStream` gets flushed (this is only meaningful in networked operation, but can be called safely in standalone mode). This has to be called after the data for one scan-point has been written to the respective `ostreams`.

7.2.3.1 Output in standalone mode

In standalone mode, the class `LocalIOStreamFactory` gets used in the investigation methods. This class behaves just as a wrapper for the ordinary C++ file operations (using the class `ofstream`). Calling `getOStream(fileName)` produces a pointer to an `ofstream` object that is associated with the file `filename` on the local filesystem. This `ofstream` object can be used in the usual way, for example by applying the `<<` operator. Calling the `commit()` method of `LocalIOStreamFactory` has no effect - here this method is just provided so that no distinction between the run-modes has to be made when using the `IOStreamFactory`.

7.2.3.2 Output in client/server mode

When running in client mode, the class `NetIOStreamFactory` gets used in the investigation methods. This class behaves as an abstraction layer for communicating with the AnT server: calling `getOStream(fileName)` produces a pointer to an `ostrstream` object, this `ostrstream` can also be used in the standard way. This object is also kept in the `NetIOStreamFactory` internally - when `commit()` is called, the contents of all open `ostrstreams` get moved into the `anpClient` communication object (see below). From there they will be sent to the server after the results for all scan-points assigned to this client have been calculated.

7.2.4 Managing the network communication

Both the AnT server and the AnT client contain a component that handles communication as illustrated in figure 7.3. These components are implementing the server- and the client-side of the ***AnT Network Protocol*** (ANP) respectively. ANP defines in which way the communication between server and client has to be performed. A detailed description of the AnT Network Protocol will be given in section 7.3.2.

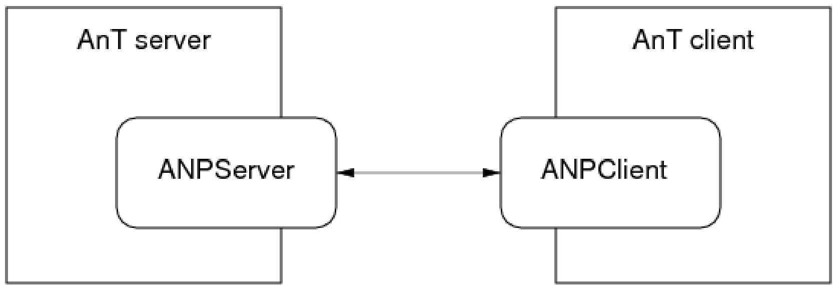


Figure 7.3: The ANP communication objects on the server and the client

7.2.4.1 Communication on the client

The `anpClient` object on the AnT client (see Fig. 7.4) implements the client-side part of the AnT Network Protocol.

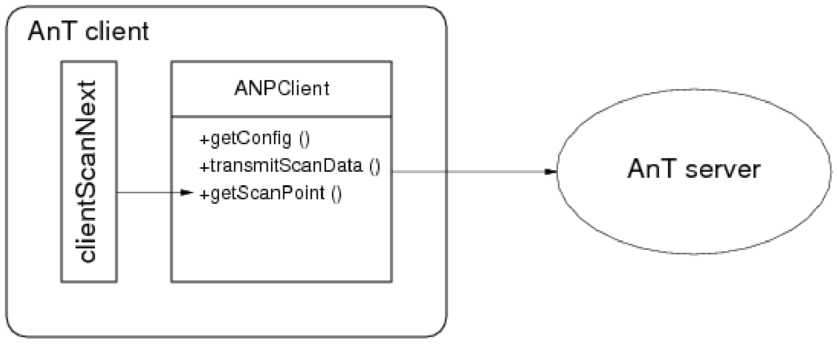


Figure 7.4: The `anpClient` object on the AnT client

There is one instance of the `ANPClient` class on the AnT client that manages the communication with the server. This object provides calls to the **AnT**

computation engine for each operation that the client needs to do relating to the network:

- `getConfig()`:
  get the configuration data for the scan from the server (this is done once, when the client is started)
- `getScanPoint()`:
  get the next scan-point (from the communication object, which itself caches scan-points according to the current block-size in order to increase the performance)
- `transmitScanData()`:
  gives the results for one scan-point to the communication object (this call also caches scan-points, until all results for the current block of scan-points are finished, in order to save communication)

Internally, the `ANPClient` class fetches and transmits data in blocks that consist of multiple scan-points (the main reason for this is that opening network connections is a relatively expensive operation, so the number of network communications has to be kept low).
The interface of `ANPClient` hides this caching, so from the point of view of the **AnT computation engine** on the client, calling `getScanPoint()` always just fetches a single scan-point.

7.2.4.2 Communication on the server

The `anpServer` object on the AnT server (see Fig. 7.5) implements the server side part of the AnT Network Protocol.
On the server the `scanNext` transition gets executed just once and handles the entire scan internally.
The `scanNext` transition of the AnT server calls the `communicationLoop()` method of the communication object (the `anpServer` object), which handles all communication on the server (see Fig. 7.5). The communication object contains three handlers for the different types of requests from clients that are called by the communication loop (see Fig. 7.6):

- `handleGetConfig()`:
  sends the configuration data for the current scan to a client

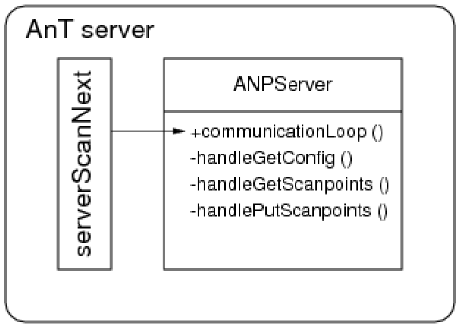


Figure 7.5: The `anpServer` object on the AnT server

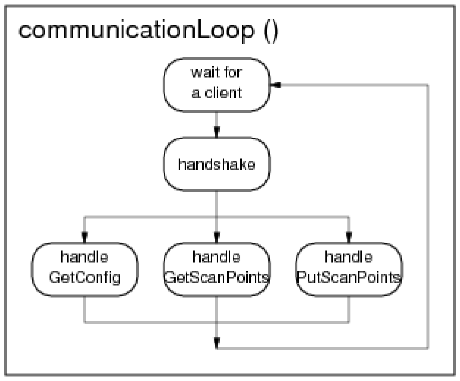


Figure 7.6: The `communicationLoop` method

- `handleGetScanPoints()`:
  sends a block of scan-points to a client
- `handlePutScanPoints()`:
  accepts the results for a block of scan-points from a client

7.2.5 Scan-point management on the server

As has already been mentioned previously, the server does more than just get the next scan-point from the AnT-engine. It needs to keep track of scan-points that have been assigned and has to remember these, in case a scan-point needs to be reassigned. The capability to reassign scan-points is necessary in order to be able to handle malfunction of client nodes gracefully. The relevant functionality for this is implemented in the class `ScanPointManagement`.
In the following, the mechanisms for scan-point management in the server are described.

7.2.5.1 Normal operation

From the point of view of the server, there are two types of scan-points that are not of concern:

- Those that are still to come, which will be announced by the **AnT computation engine** in the future. There is no information about these scan-points in the `anpServer` or `ScanPointManagement` yet.
- Those which are entirely done - they have been assigned to a client, the client has reported results and the results have been written to disk. There is no longer the need to keep any information regarding these in the `anpServer` or `ScanPointManagement`.

However, the server needs to keep track of two different sets of information:

- scan points that are currently in progress:
  These are scan-points that have been assigned to a client, but the results have not yet been reported back. The server needs to keep the information about these scan-points in case the need arises to assign them to another client (for example when the original client never sends a result because of a malfunction). The order in which the scan-points

have been assigned to clients is also stored so that the “oldest” scan-points can be reassigned first.

- unsaved results:
  This is data that clients have reported but which has not yet been written to disk. Results need to be saved in the correct order - the same order in which the **AnT computation engine** traverses the scan-space (this is also the order in which the scan-points get assigned to the `anpServer` by the **AnT computation engine**). So if the clients report results in another order, some results need to be cached in memory until all results that have to be written out prior to them are reported.

7.2.5.2 Reassigning scan-points

There are two possible reasons for reassigning scan-points:

- All scan-points have already been assigned to clients, but a new client is asking for a task. In this case it might be a good strategy to reassign some “old” scan-points to this client. The reason why no results for this scan-point have been reported might be that the client that was originally working on it has crashed. Or the client might be very slow and unable to report a result within reasonable time.

  The duration of the entire scan can not become longer because of re-assigning scan-points in this manner (however, the amount of total CPU-cycles spent can!).

- There are a lot of unsaved results and the server is running out of memory because of this. In this case, it is not entirely clear what to do:

  The missing results (after which the server could write out some of its unsaved results) might be reported very soon - in this case reassigning the scan-points would be a bad strategy.

  On the other hand, the responsible client might have crashed. In this case it would be fine to reassign the scan-point again now instead of at the end of the entire run.

The current implementation only does the first type of reassignment. When implementing the second type of reassignment, there is the need to make some “guess” about when it is a good time to reassign a scan-point. This will always be a trade-off between overall speed of the scan and the memory usage of the AnT server.

7.2.6 Fetching more than one scan-point at a time

Opening a TCP network connection (see Sec. 7.3.1.1) is a very time-consuming operation. Because of this, it is often more efficient to open fewer connections of which each transmits more data instead of more connections of which each transmits smaller amounts of data.
In order to minimize the number of connections to the server that AnT clients open and close during a scan, more than one scan-point can be handled during one communication cycle.

7.2.6.1 Strategies for fetching on the client

In the current design the decision of how many scan-points should be fetched in one block (the results are always reported in the same blocks as the work is assigned) is done by the client. The reasoning behind this is that the server does not know about the clients at all - the server makes no assumptions concerning clients. Thus only the client itself can decide what amount of scan-points is reasonable to work on considering its own speed.
The goal of choosing an amount of scan-points that are transmitted as a block is to find a good balance between

- minimizing the number of communications (establishing a network socket connection is an expensive operation),
- not transmitting an undue amount of data as a result (output of investigation methods) in one block (the transfer time for the assigned scan-points and for the result that is sent back should not be overly long, as the server is blocked and can’t process requests from other clients during that time) and
- not having too much reassignment of scan-points happening close to the end of a scan. The extreme case of this problem is when each client fetches all available scan-points and each client just works on the same set of scan-points. To some extent this effect always happens close to the end of a scan (so a possible optimization would be to minimize it).

These aims contradict each other, so it is not obvious which block-size is good for a given simulation run. As outlined above, there are a number of influences on the optimal block-size.
The current implementation provides two ways to choose the size of the blocks that get transferred:

- Select a certain fixed number of scan-points that will always be transferred at a time.
- Select a certain amount of time that the clients should be busy between two interactions with the server. The number of scan-points per block changes over the course of a scan according to the time a block takes on the client to be completed - the block-size adapts to the workload on the node that a client is running on, different complexity of the dynamical system in different parts of scan-space and to all other influences on the client's speed of processing the assigned scan-points.

7.2.7 Limitations in client/server mode

In contrast to many problems in computer science, simulation of dynamical systems as done by the **AnT computation engine** is rather well suited for being distributed on many nodes. The entire functionality can be used in client/server mode.

The only limitation that is known at the time of this work is, that the state of the simulator at the end of the previous scan-point cannot be easily used as the initial state at the beginning of a following scan-point. This is not easily possible because the order in which the scan-points are processed does not guarantee that the 'previous' scan-point has already been completed at the time - in fact it is very unlikely that it has been in the current realization. Most subsequent scan-points will be processed in parallel, or even out of order.

Reusing the previous state in this manner can be an advantage in order to keep the transient part of orbits small and thus to save iterations. However, this is not really a limitation to functionality, but rather an optimization that is not easily applicable in the parallelized version of the **AnT computation engine**.

7.3 Network Communication

This section describes how AnT clients communicate with the AnT server. Therefore first the ***TCP/IP protocol suite*** and ***Internet sockets*** will be introduced. After that, the ***AnT Network Protocol*** will be discussed. This communication protocol defines the actual communication between AnT clients and the AnT server. The section then concludes with some thoughts on security in the context of the ***AnT Network Protocol***.

7.3.1 The communication layer

7.3.1.1 TCP/IP

The Internet as it exists today is built on the ***TCP/IP protocol suite*** that was introduced in the year 1983. The constant growth of the Internet - it doubles in size about every year - brought massive popularity to TCP/IP, which is now in widespread use in both scientific and corporate context.
At the time of this work, competing network protocols, such as IPX, SNA or UUCP only have any significance in small niche markets. For this reason, the communication protocol of the distributed version of the **AnT computation engine** is based on TCP/IP.
The TCP/IP protocol suite gets its name from the two most important protocols that belong to it: the ***Transmission Control Protocol*** and the ***Internet Protocol***, although it includes many other protocols too. A more detailed discussion of TCP and IP can be found in the respective RFCs, [138] and [139].
The architecture of TCP/IP can be divided into four layers [81] (see Fig. 7.7):

- ▶ The lowest layer is the ***Network Access Layer***. It abstracts the technical details of the underlying network. For each physical network standard (such as Token-Ring or Ethernet) a ***Network Access Protocol*** has to be developed, that, among other things, handles the encapsulation of IP datagrams into the frames transmitted by the network and takes care of the mapping of IP addresses to physical network addresses.

- ▶ Above the Network Access Layer lies the ***Internet Layer***. This layer implements IP, the ***Internet Protocol***. All higher layers use IP to transport data. The responsibilities of the Internet Layer include segmenting data into datagrams and routing datagrams to remote hosts.

- The next layer is the ***Host-to-Host Transport Layer***. The ***User Datagram Protocol*** (UDP) and the ***Transmission Control Protocol*** (TCP) reside in this layer. Most applications use these protocols to transport data.

  UDP provides a low-overhead, connection-less datagram delivery service. UDP is an unreliable protocol. It does not guarantee that datagrams are delivered in the right order, or at all. If, however, a datagram is delivered, the data is guaranteed to be error-free.

  TCP, in contrast, is a reliable, connection-oriented protocol. All datagrams are guaranteed to be delivered and the sequence of the datagrams is preserved. As a drawback, TCP has a higher overhead compared to UDP. Because reliability is crucial for AnT, TCP is used as communication protocol.

- The highest layer is the so-called ***Application Layer***. This layer includes well-known ***application protocols*** such as telnet, FTP, SMTP or HTTP. The distributed version of the **AnT computation engine** resides in this layer as well.

7.3.1.2 Internet sockets

The application protocols that are implemented at the Application Layer of TCP/IP described earlier, use so-called ***Internet sockets*** to exchange data over a TCP/IP network[1].
There are basically two kinds of Internet sockets, ***Datagram sockets*** and ***stream sockets***[2] [76].
Datagram sockets use the ***User Datagram Protocol*** (UDP) to send messages. Because of the properties of UDP discussed in section 7.3.1.1, Datagram sockets are connection-less and unreliable. Datagram sockets are used in application protocols like tftp and bootp.

[1]There also exist so-called ***UNIX Domain sockets***, that are used for interprocess communication. In this context only ***Internet sockets*** are of interest.
[2]Sometimes also called ***TCP sockets***.

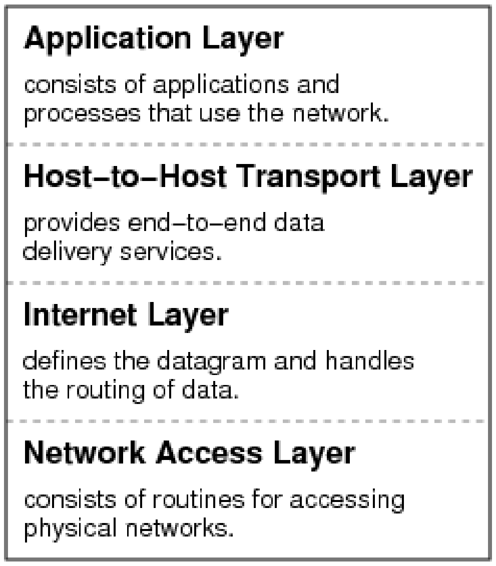

Application Layer
consists of applications and processes that use the network.
Host–to–Host Transport Layer
provides end–to–end data delivery services.
Internet Layer
defines the datagram and handles the routing of data.
Network Access Layer
consists of routines for accessing physical networks.

Figure 7.7: Layers in the TCP/IP protocol suite

stream sockets in contrast use the ***Transmission Control Protocol*** (TCP). TCP guarantees the reliable delivery of data. TCP is a ***reliable***, ***connection-oriented***, ***byte-stream*** protocol. Therefore stream sockets are reliable, two-way connected communication streams. stream sockets are in widespread use, for example in application protocols like telnet or HTTP.

TCP achieves reliability by a mechanism called ***Positive Acknowledgment with Re-transmission*** (PAR). The sender retransmits a message, if it doesn't receive an acknowledge message from the remote system in time. Each message contains a checksum, so the recipient can check if the data is undamaged. Messages also contain a sequence number that can be used to identify duplicated messages and to preserve the sequence of the messages.

Connection-oriented means that a connection between two hosts has to be established first, before any data can be transmitted. TCP uses a so-called ***three-way handshake*** to establish a connection[3]. Once the connection is established, data can be sent in both directions.

TCP handles the data it sends as a continuous stream of bytes. To an application, stream sockets look just like sequential files. The system calls to

[3]This handshake is not to be confused with the ANP handshake described in section 7.3.2.2.

read from and write to stream sockets are the same as the system calls that are used to read and write files.
As has been seen, stream sockets are a reliable and comfortable means of communication between remote hosts. The application protocol that is used in the distributed version of the **AnT computation engine** uses stream sockets to exchange commands and data between the clients and the server. The term ***socket*** from now on always addresses ***stream sockets*** in this document.

7.3.1.3 Client/server programming with sockets

The ***client/server*** model is very popular in network programming. Almost all well-known network services use this model. Telnet for instance consists of a telnet server running as a daemon (telnetd) and a telnet client that is called by the user (the telnet program itself). There are many other client/server pairs, such as ftp/ftpd, ssh/sshd or web browser/httpd.

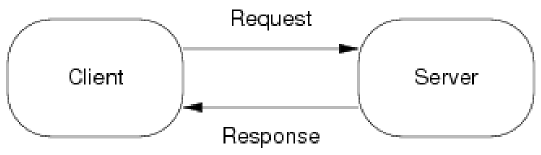


Figure 7.8: The client/server model

Figure 7.8 summarizes the interactions between the client and the server. The communication is always initiated by the client by sending a request to the server. The server then sends a response to the client and the communication is ended. Often more than one client can talk with a server.
The client/server model can be implemented in many different ways - using Datagram sockets, stream sockets or any other communication protocol. In the **AnT computation engine**, the client/server model is implemented using Stream sockets. Figure 7.9 shows the system calls that take place in a simple client/server example using stream sockets [76].

- ▶ At first the server is started. The server creates a new socket with the `socket()` system call.
- ▶ It then calls `bind()` to assign a port number to the socket.

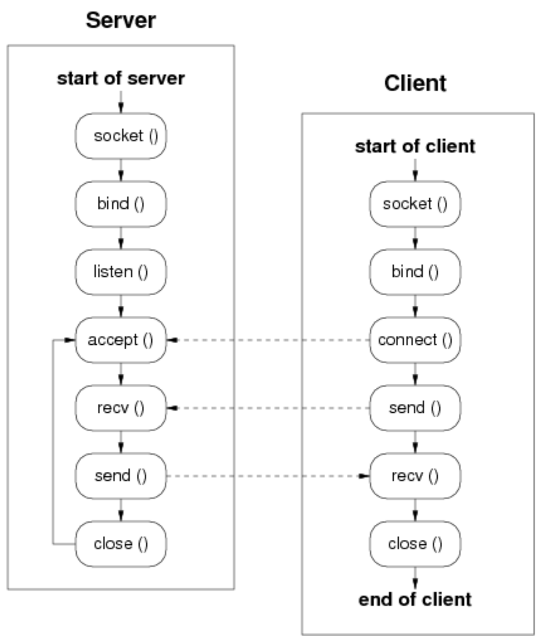


Figure 7.9: Implementing the client/server model with stream sockets

- ▶ The `listen()` system call prepares the server to accept incoming connections.
- ▶ The server then waits for a client to connect calling `accept()`.
- ▶ Now a client can be started. The client also has to create a socket first using the `socket()` call.
- ▶ Then the client has to `bind()` the socket to a local port number.
- ▶ The client then can establish a socket connection to the server using the `connect()` system call.
- ▶ Once the connection is established both server and client can send and receive data with the `send()` and `recv()` system calls.
- ▶ When all communication is done, both client and server call `close()` to close the socket connection.

The server then loops back to the `accept()` call to handle further client requests. If more than one client starts a request at the same time, the

`connect()` requests are stored in a queue and handled by the server one by one.
It is also possible that the server creates a new thread (or forks) after the `accept()` call and the client is handled by a different thread (or process). The server then immediately returns to the `accept()` call. In this case, the server can handle several clients at the same time. This approach can be used to improve the performance of the server, but makes it generally more difficult to program the server, in particular when the server threads need to be synchronized. The AnT server as it is implemented now does handle the clients in one single-threaded task.
Although a C++ wrapper class is used in the **AnT computation engine** to program sockets instead of the basic system calls, the same principles for the communication between the AnT server and the AnT clients as the ones explained here apply.

7.3.2 The AnT Network Protocol

7.3.2.1 Introduction

As discussed earlier, the client/server model is used in the network version of the **AnT computation engine**. Therefore it is divided into one AnT server and one or many AnT clients. The ***AnT Network Protocol*** (ANP) defines the communication events that can take place between an AnT client and an AnT server. These events are so-called ***ANP commands***.
ANP uses stream sockets as described in section 7.3.1.2. Therefore the participating nodes have to be connected by a TCP/IP network in such a way that each client can establish a TCP connection with the server at any time.
It is always the AnT client that initiates communication with the AnT server. The server thus has to be started prior to any client and is always waiting for a client to send an ANP command. Figure 7.10 shows the life-cycle of an AnT client. The figure also shows all available ANP commands.
The `GET CONFIG` command is called once during the startup of a client to fetch the configuration data from the server. During calculation the `GET SCANPOINTS` and the `PUT SCANPOINTS` commands are called several times to get new scan-points from the server and to report the results back to the server. The subsequent sections describe the individual ANP commands in detail.
ANP as described in this document has the version `ANP/1.0`. As changes

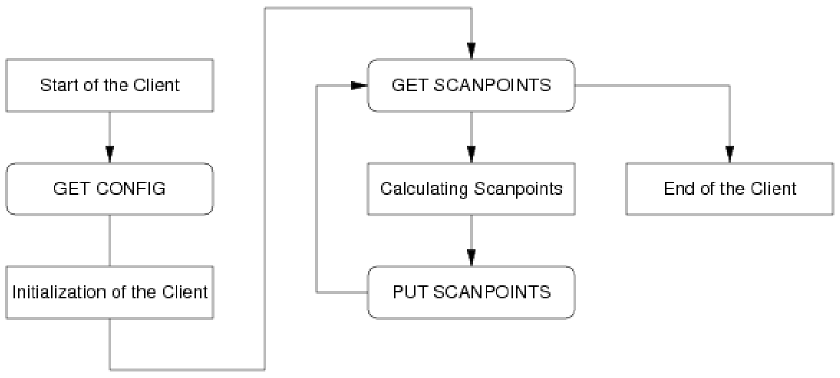


Figure 7.10: Life-cycle of an AnT client

to the protocol become necessary, a new ANP version number has to be assigned to the new protocol. This is intended to avoid problems when incompatible versions of an ANP server and an ANP client try to communicate (the handshake discussed in the following section is designed to detect such problems).

7.3.2.2 Establishing a connection

Every communication between a client and the server is preceded by a handshake: the client sends its ANP version and the name of the active dynamical system to the server. The client then waits for the server to reply.

```
Client sends:

ANP/1.0
lorenz63
```

The server checks if the client has loaded the same dynamical system as the server (by comparing the transmitted name of the system) and if the ANP version of the client is compatible with the ANP version of the server. If the client meets both requirements, the server sends a reply containing the ANP version of the server.
Receiving this message, the client knows that the connection has been successfully established. The server now expects the client to send an ANP command (like `GET CONFIG` or `PUT SCANPOINTS`, which will be described in the following sections).

```
Server sends:

ANP/1.0
```

If, on the other hand, the client has not loaded the same dynamical system as the server, or the ANP version of the client is not compatible with the ANP version of the server, the server tells the client that the connection failed. This `CONNECT FAILED` message is followed by an error message explaining what went wrong. The client exits with a respective error message when receiving such a reply from the server.

```
Server sends:

CONNECT FAILED
<error message>
```

This handshake takes place prior to every single ANP command a client sends to the server. The intention of this handshake is to prevent mistakes of the user - it is not sufficient to prevent a malicious client or server from exploiting the communication protocol. See section 7.3.3 for a more detailed discussion of security in regard to ANP.

7.3.2.3 Initialization of a client

After a client has been started, the first thing the client does is to fetch an initialization file (see Sec. 8.2.1) from the server. The client initializes itself according to the information provided in the initialization file. Therefore the client first establishes a connection using the handshake described in section 7.3.2.2. It then sends the ANP command `GET CONFIG` to the server and waits for the server to send the initialization file:

```
Client sends:

GET CONFIG
```

The server answers to this command by sending the content of the initialization file followed by an `EOF` character and closes the connection afterwards.

```
Server sends:

<contents of the initialization file>
EOF
```

7.3.2.4 Requesting scan-points

Using the ANP command `GET SCANPOINTS` the client can request new scan-points from the server. A client usually does this right after it has initialized itself (see Sec. 7.3.2.3) or after it has reported the result for all previously fetched scan-points (see section 7.3.2.5). Once again the client first has to perform a handshake (as described in section 7.3.2.2) before the `GET SCANPOINTS` command can be sent.
The client tells the server how many scan-points it would like to fetch. The user of each client specifies, how this amount is to be determined. Either the amount is fixed, or the client tries to fetch as many scan-points as needed to keep it busy for a certain amount of time.

```
Client sends:

GET SCANPOINTS
<number of scan-points>
```

- `<number of scan-points>` is the number of scan-points the client would like to get from the server.

If there are any scan-points available, the server answers as follows:

```
Server sends:

<number of scan-points>
<scan-point>*
```

- `<number of scan-points>` is the number of scan-points that the server is about to send to the client. This may be up to as many scan-points as the client requested from the server (However, the server might send fewer scan-points as requested, in particular near the end of the calculation).
- `<scan-point>*` are the individual scan-points. Each `<scan-point>` is transmitted as follows:

  ```
  Server sends:
  ```

```
<sequence number>
<length of scanpoint description>
<scanpoint description>
```

- `<sequence number>` is a consecutive number that is assigned to the scan-point by the server. The server uses this number to be able to associate results reported by the clients with a scan-point. Therefore the server expects this number to be reported by the client along with the results for the scan-point.
- `<length of scan-point description>` is the length of the subsequent description of a scan-point in bytes.
- `<scan-point description>` is the ASCII representation of a scan-point.

If, on the other hand, the calculation is complete and the server has no more scan-points to assign to the client, the server replies a `0` to the client (0 scan-points available) and closes the connection. The client here-upon exits (see Sec. 7.3.2.6).

```
Server sends:

0
```

7.3.2.5 Reporting results

If a client has calculated the results for all scan-points assigned to it, it reports the results back to the server. To initiate this, the client sends the ANP command `PUT SCANPOINTS` to the server (after establishing a connection with the server as described in section 7.3.2.2).
The `PUT SCANPOINTS` command is followed by the number of scan-points for which results are to be reported. Then the individual scan-point results are transmitted.

```
Client sends:

PUT SCANPOINTS
<number of scan-points>
<scan-result>*
```

- `<number of scan-points>` is the number of scan-points that the client is going to report results for.
- `<scan-result>*` are the individual results for the scan-points. Each `<scan-result>` consists of the following three parts:

  ```
  Client sends:

  <sequence number>
  <number of files>
  <file>*
  ```

  - `<sequence number>` is the sequence number that the server transmitted along with the scan-point initially (see Sec. 7.3.2.4).
  - `<number of files>` is the number of files that the result for this scan-point consists of. The individual files are transmitted afterwards[4].
  - `<file>*` are the files that contain the results. Each `<file>` consists of three parts which are constructed as follows:

    ```
    Client sends:

    <filename>
    <length of file>
    <file contents>
    ```

    - `<filename>` is the name of the file.
    - `<length of file>` is the length of the following file contents in bytes.
    - `<file contents>` are the contents of the file as plain ASCII text.

When the last scan-point is transmitted the client ends the communication. The server doesn't send any acknowledgment.

[4]number of files can also be zero

7.3.2.6 Terminating the server and the clients

The server's during transition exits (and thus the ANPServer object gets destructed) as soon as results have been reported for every scan-point by the clients (after that, the server executes its post transition, which can for example contain transitions that belong to an investigation method). Because the server doesn't manage a list of participating clients, it can't tell every client to quit. Instead, the clients exit if they can't establish a network connection to the server anymore (which they can't after the ANPServer object has been destructed).
As an exception, the first client that can't fetch any new scan-points exists after receiving 0 for the number of available scan-points as response for the ANP command `GET SCANPOINTS` (see Sec. 7.3.2.4).
If one wants to stop a simulation run, only the server needs to be stopped. The clients will thereafter exit automatically (after their next attempt to connect to the server).

7.3.2.7 Restarting an AnT server

When stopping the AnT server on a given machine and starting it again (after a short period of time), it can happen that the TCP port that the AnT server tries to bind to has not yet been released by the kernel.
This problem is not related to the AnT server in particular (see also [108]) - under certain circumstances the socket that the server used while it was running goes into a state called `TIME_WAIT` and stays there for a certain time. During this time, it is not possible to `bind()` to this port. This is not a bug, but rather a feature of TCP/IP.
One noticeable effect of this is that after an AnT server has stopped, it is impossible to start it again during a certain period of time. The length of this period of time depends on the operating system. On Solaris it is 4 minutes while on many other systems (including GNU/Linux) it is 1 minute.
In case of this failure, the AnT server exits with an error message of the form

```
Could not bind to address '<hostname>:<port>'
```

7.3.3 Security

This section will discuss security issues connected with the networked operation of the **AnT computation engine** and the AnT Network Protocol.

7.3.3.1 Basic assumptions

In computer networks two generally different sorts of attacks have to be considered: attacks that only listen to communication on the network (eavesdropping) and attacks that change existing communication or produce communication by themselves.
In this work, the basic assumption is that eavesdropping on the communication between an AnT server and an AnT client is no risk by itself - because the calculations that are made are not of a confident nature. If this assumption is not true in some application of the **AnT 4.669** software package, then either the entire concept of ANP has to be changed accordingly, or the network needs to be secured in an appropriate way (fire-walling, control, encryption, . . .). In universities and similar environments, such secrecy should usually not be any concern.
In contrast to this, active changes of the communication between AnT client and server have to be rated differently. Attacks to computers that are connected to the Internet are part of life on the net - every service that is running on a system that is connected to the Internet is a potential security risk. This also applies to the AnT server.
But as the **AnT computation engine** and in extension ANP will in all likelihood never be spread very widely, it seems safe to assume that attacks to ANP will not happen on a large scale - if anything, attackers will most likely be not overly motivated.
The most likely type of attack will probably have the form of a “prank”, or a “practical joke” in a context where the **AnT 4.669**software will be used (for example within a university). ANP is designed to withstand such attacks to a certain degree (the main obstacle is to challenge the client to announce the name of the dynamical system which is currently being simulated).

7.3.3.2 Filenames for output-files

Having the server store results in an arbitrarily named file that can be chosen by any AnT client poses an enormous risk. A malicious client could try to change important initialization files or corrupt data this way (editing `$HOME/.rhosts` on the machine the AnT server runs on is not something that the AnT client should be allowed to do).
To minimize the possibilities of such an attack, the server only accepts filenames of a certain form. It is not allowed to change to a directory above the

current one (that the AnT server was started from) - in particular "..", "∼" and a "/" at the beginning of the filename are not allowed.
Legal filenames are defined by the following regular expression:

```
<alphanum> = [a-zA-Z0-9_]
<slash>    = /
<dot>      = .

<filename> = <alphanum>+ ( <slash> <alphanum>+ )*
             <dot> <alphanum> <alphanum> <alphanum>
```

Filenames which can't be produced by the regular expression `<filename>` get ignored by the server, the corresponding data is lost and a warning of the form

```
ANPServer: Invalid filename '<filename>'
```

is printed on the server's console.

7.3.3.3 Attacks of the communication between client and server

Eavesdropping on the communication As already stated above, in the design of ANP it was assumed that there is no danger in eavesdropping itself. Therefore if an attacker has the means to eavesdrop on the net (for this being root on a machine on the same network is usually needed), ANP can do nothing to stop him.
In general, if no encryption of the entire communication is done, there is no protection against eavesdropping (just as with most other Internet protocols like http, ftp, electronic mail, nfs, . . .).

Attacks by a malicious client The AnT Network Protocol provides basic protection against an attack by a malicious client (which could be a program written from scratch, designed to disrupt the operation of a simulation performed by the **AnT computation engine** or a modified version of the AnT client itself) by demanding the complete ANP handshake before the actual communication takes place - a client that is not able to fully perform the

handshake will not be able to interfere with the calculation itself (to complete the handshake, knowledge of the name of the system that is being simulated is necessary[5]).
At best, a malicious client that does not know the right system name can block the AnT server by connecting to it without communicating at all (this would amount to a ***denial-of-service*** attack).
A malicious client that knows the right system name on the other hand can interfere with the calculation in very disruptive ways, worst of all sending the server wrong results.
To successfully attack the AnT server in this way the attacker needs either insider knowledge or the ability to eavesdrop on the network (or a list of names of dynamical systems to conduct a brute-force ***word-list attack***).

Manipulation of communication In this case, manipulation of communication amounts to the same as the combination of eavesdropping and a malicious client - it is possible to pollute the results of the simulation by manipulating results as they are being sent to the server.
Another possible attack would be to make the clients use up all their CPU time without actually generating any results (by changing the initialization file that is sent from server to client, for example) - this would also be a denial-of-service attack.

7.3.3.4 Attacks on the lower layers

So far, the security of the ANP layer was considered. But there is also the risk that any of the layers below the application layer might be vulnerable to some sort of attack - the underlying socket-libraries of the operating system may have bugs, the standard C++ library, the implementation of TCP/IP in the operating system. Many security holes that get exploited are not flaws of the attacked program itself, but problems of a specific library that this program uses (in particular ***buffer overflows*** are regularly provoked this way). Such attacks could ultimately lead to a break-in into the account of the user in which the AnT server is running.

[5] An attacker with the ability to eavesdrop on the communication can of course trivially obtain the system-name. However, eavesdropping on a network is typically harder than only communicating with a server.

7.3.3.5 Minimizing the risk

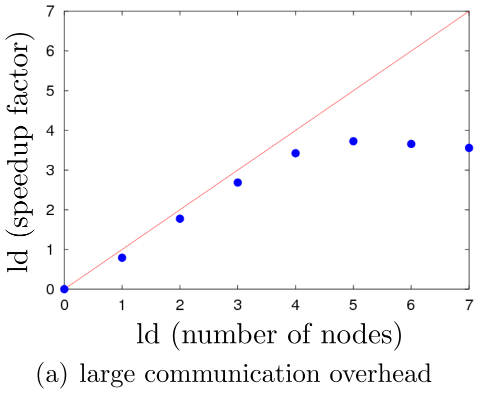
Running the AnT server from a privileged user account has to be considered a risk, even if no attack is known at this time. If there is the need to take as little risk as possible while using the AnT server, it might be best to create a user account that does nothing but run the AnT server. This way, the possible damage by a successful attack on the AnT server can be kept within certain limits.

7.4 Scaling behavior of distributed scans

When dealing with grid computing and distributed applications on computer networks or clusters, one of the important questions is about the ***scaling behavior*** of the application, that is roughly speaking the ***speedup factor*** with respect to the total number of nodes involved in the computation. Of course, the ideal speedup factor would be the number of nodes itself and is in principle the theoretical limit to the speedup factor. In some extraordinary cases, so-called ***speedup anomalies*** could be observed, which may result in a ***superlinear speedup*** slightly larger than the number of involved processors. These speedup anomalies are mainly caused by memory or ***caching effects*** and have no real meaning.

In almost all cases, the scaling behavior depends on the number of involved computing nodes and of course on the specific problem under consideration. If a problem can be generically parallelized, which is the case when the problem can be divided into subproblems which are totally independent from each other, then the speedup factor grows first linearly, before it leaves the bisecting line. For a certain number of involved nodes, the speedup factor has a maximum value and slows down if the number of nodes is increased further, due to ***communication overhead*** which increases more and more with the number of involved nodes.

Because the scan problems described in chapter 4 can be generically parallelized, the scaling behavior of the client/server mode of the **AnT computation engine** is exactly of that type described above. This scaling behavior of some typical scan problems is illustrated in figure 7.11. Hereby a specific scan problem was solved on a Linux cluster with 128 identical



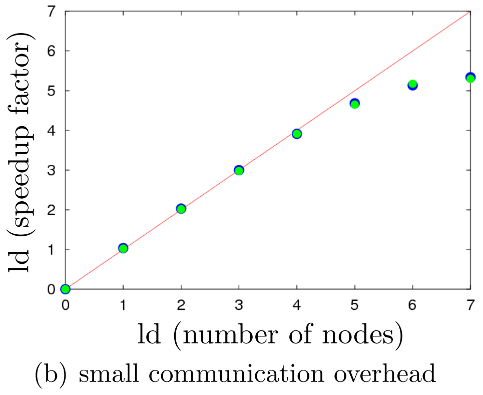


Figure 7.11: Typical scaling behavior of distributed scans
Shown are the speedup factors (blue and green circles) of three different scan-runs in dependence on the number of involved computation nodes varied from 1 to 128 in powers of two in a doubly logarithmic representation. The red bisectoring line marks the theoretical limit of the speedup factor. Clearly one can see the divergence of the real speedup factor from the theoretical limit at a certain number of involved nodes, caused by communication overhead. In (a), the scan problem is ill-posed which results in a large communication overhead and a reduced speedup factor (note the logarithmic scale), whereas the two cases in (b) are well-posed scan problems with small communication overheads resulting in relatively good speedup factors.

Intel Xeon 3.06GHz CPUs. The number of involved nodes was varied from 1 to 128 in powers of 2. The speedup factors were calculated as the ratio of the total computation time using only one node to the total computation time needed by the corresponding number of computing nodes. As one can see, the client/server architecture of the **AnT computation engine** scales very well for the considered problem.

Remark:
An appropriate choice of the command line arguments presented in section 2.4.2, would lead in many cases to a suitable scaling behavior like the one presented in figure 7.11(b). However, this appropriate setting has to be done by hand and requires some experience.

Chapter 8

AnT-gui: the graphical user interface

8.1 Motivation

In this chapter, the graphical user interface **AnT-gui** of the **AnT 4.669** software package is presented. Because it was designed as a non-interactive simulation system (see Sec. 1.2.3.2), the initialization phase of the **AnT computation engine** may be a challenging task. The degree of complexity of the initialization and how time-consuming it is, depends not only on the knowledge about dynamical systems and on the experience with the **AnT 4.669** software package, but also on the specific dynamical system to be investigated.

1. Because the specification of initial values or system parameters depend linearly on the state space dimension N_s or the parameter space dimension N_p, it is obvious, that the initialization is more time-consuming for dynamical systems with many degrees of freedom or many system parameters.

2. The initialization of DDEs (see Sec. 3.4.2.1) and FDEs (see Sec. 3.4.2.3) requires an initial function on the memory interval for each state space variable, which usually have some additional parameters to be specified. Similarly, the PDEs (see Sec. 3.4.1.3) require the specification of a spatial initial function for each state variable.

3. The usage of the scan capabilities of the **AnT 4.669** software package (see Chap. 4), requires an additional initialization of the scan to be executed, that is the specification of the individual scan items contained in the scan item sequence.

4. Usually each investigation method needs specification parameters and settings, which have to be supplied by the user.

5. Due to the many possible graphical representations of trajectories, the usage of the OpenGL based visualization requires a large number of settings.

The abovementioned examples of some initialization steps demonstrate, that the development of a graphical user interface was necessary, to support and guide a user of the software through the complex initialization phase. Unfortunately, it is not possible to demonstrate all the capabilities of the **AnT-gui** in detail, instead a rough overview concerning the main functionalities is given.

8.2 Initializing the AnT computation engine

The initialization of the **AnT computation engine** is done by an ***initialization file***, which contains all necessary information mainly by the specification of corresponding ***key-value pairs*** or ***key-value assignments***. Hereby the notion of value has to be interpreted in that sense, that to a given key also an array or another data structure could be assigned to. The initialization file initializes and controls the **AnT computation engine**, so that the engine can perform the simulation and investigation of the specified dynamical system according to the ***user instructions*** given by the key-value pairs. The initialization file can be created by hand or even simpler and much more convenient with the graphical user interface **AnT-gui**. As already mentioned, the user interface guides the user through the initialization phase in a way, which should be as intuitive as possible. After the initialization is completed, the user interface allows the saving of the defined initialization in an initialization file, so that it can be used later by the **AnT computation engine**. Alternatively the **AnT computation engine** could be launched directly from the user interface and the simulation and investigation is carried out immediately.

The advantages of the usage of key-value pairs or key-value assignments are:

- ***shared data***
  All possible key-value assignments are defined in the configuration file `GlobalKeys.cfg`, which will be installed together with the software itself. On the one hand, the **AnT computation engine** uses this configuration file to verify, whether a user supplied initialization file, that is the set of key-value assignments, is syntactically correct. On the other hand, the graphical user interface **AnT-gui** uses the same configuration file, which controls in this case the possible entries or selections in the several windows of the graphical user interface.
- ***flexibility***
  The names of the keys could be changed and this has to be done at only one place, namely within the configuration file `GlobalKeys.cfg`. Once a change is made and of course after the compilation, the graphical user interface is automatically "aware" of this change, because it gets its information from the configuration file.
- ***extendibility***
  When extensions to the **AnT computation engine** are made, for instance a new integration method for the ODE class (see Sec. 3.4.1.1) was added, then after the compilation, the list of integration methods in the graphical user interface (see Fig. 8.6) contains the new method automatically, without any further intervention of the developer or maintainer.
- multi language support
  Although at the moment, the **AnT 4.669** software package has no multi language support, it could be implemented without much effort, because only the file with the definition of the key-value assignments have to be adapted to the new language.

8.2.1 A simple initialization file

In the following a simple - in fact really simple - initialization file is presented, which was used to produce the cobweb diagram (see Fig. 3.2(c)) in Chap. 3 on page 52.

```
# Generated by AnT 4.669, Release 3a, (c) 1999-2004

dynamical_system = {
  type = map,
  name = "Logistic map",
  parameter_space_dimension = 1,
  parameters = {
    parameter[0] = {
      value = 3.83,
      name = "alpha"
    }
  },
  state_space_dimension = 1,
  initial_state = (0.504666487408413),
  reset_initial_states_from_orbit = false,
  number_of_iterations = 4
},
scan = {
  mode = 0
},
investigation_methods = {
  general_trajectory_evaluations = {
    is_active = true,
    transient = 0,
    saving = {
      is_active = true,
      type = time_oriented,
      points_step = 1,
      trajectory = false,
      trajectory_file = "orbit.tna",
      cobweb = true,
      cobweb_file = "cobweb.tna",
      velocity = false,
      current_velocity_file = "orbital_velocity.tna",
      phase_portrait = false,
      phase_portrait_file = "phase_portrait.tna",
      initial_states = false,
      initial_states_file = "initial_states.tna",
      save_only_specific_area = false
    }
  }
},
visualization = {
  is_active = false,
  transient = 0,
  points_step = 1,
  update_step = 1,
  buffer_size = 1,
  number_of_windows = 0
}
```

As one can see, the initialization file consists of four main sections, namely the dynamical system section, the scan section, the investigation section and the visualization section. Each of this main sections specifies a part of the initialization.

8.2.1.1 The dynamical system section

In this section of the initialization file, general information about the dynamical system to be investigated should be provided. If one takes into account, that in the case of a user supplied system, the user has provided a C++-file of the implementation of the system, one could think, that some part of this information is redundant. In fact it is, but it is used for consistency checks. For instance, if a user provides an implementation of a dynamical system of the ODE class and specifies in the initialization file a map, the **AnT computation engine** bails out with the following error message:

```
loading the system... system loaded successfully
starting scanMachine...
ProxyException: Map system function is not defined!

Error::Exit: abnormal program termination!
Bye!
```

Or in the case where one specifies, that the system has a state space dimension of 2, but the implemented system has in fact three state variables, then the **AnT computation engine** bails out with the error message:

```
Array assignment failure.
Trying to put an entity list of length 3
into an array of length 2.

Error::Exit: abnormal program termination!
Bye!
```

The main part of the information given here provides the **AnT computation engine** with general information about the dynamical system such as

- the parameter space dimension N_p
- the values of the parameters
- the state space dimension N_s
- the initial states or initial functions
- the number of iterations
- the numerical integration method if necessary

8.2.1.2 The scan section

In this section of the initialization file, the scan mode, the scan item sequence and the individual scan items are provided. In the example above, the scan mode is 0, and hence no scan is performed. This is obvious, because the cobweb diagrams are defined for fixed parameters only.

8.2.1.3 The investigation method section

In this section of the initialization file, the investigation methods which should be applied during or after the simulation of the considered dynamical system are provided. Depending on the complexity of the investigation method, the user may provide here several specifications. See for instance the relatively large number of specification entries in figure 8.5, for the frequency analysis (see Sec. 5.7), the Lyapunov exponents analysis (see Sec. 5.5) and the period analysis (see Sec. 5.3). In the presented example initialization file, only the `general trajectory evaluation` method is selected with an activated saving option and the cobweb representation as only selected saving method. Together with the number of four iterations and the initial value $x_0 = 0.504666487408413$, which was chosen in such a way, that it already lies on the three-periodic limit cycle, a run of the **AnT computation engine** would lead to the saving of the data needed for the cobweb diagram in figure 3.2(c)).

8.2.1.4 The visualization section

In this section of the initialization file, the OpenGL based visualization is specified. According to the many features and possibilities of the visualization, this part is the most substantial part of the initialization. As one can see, in the presented example, the visualization is switched off.

8.3 The graphical user interface

As already mentioned, the graphical user interface of the **AnT computation engine** guides a user through the initialization phase of the simulation system. Hereby, the user interface supports the user of the system by a help system, giving hints via pop-ups and detects unresolved dependencies or conflicts. In figure 8.1, the `main` window of the user interface is shown.
From this one can start from scratch creating a new initialization file or load an existing one. Because the initialization procedure of the **AnT-gui** is based on the structure of the initialization file, its `root window` has also the four main sections or parts described above (see Fig. 8.2).
In figure 8.3 the windows corresponding to these four main parts are shown. After the editing of the file is completed, a user can save it and start the **AnT computation engine** later or launch the **AnT computation engine** directly using the run button of the `main` window. After that, a small window asks for the run-mode and the system to be simulated. The system can be chosen via a selector, which allows the selection of the corresponding shared library file (see Fig. 8.4).
In figure 8.5 screen shots of some investigation methods are presented,

Chapter 8
AnT-gui: the graphical user interface

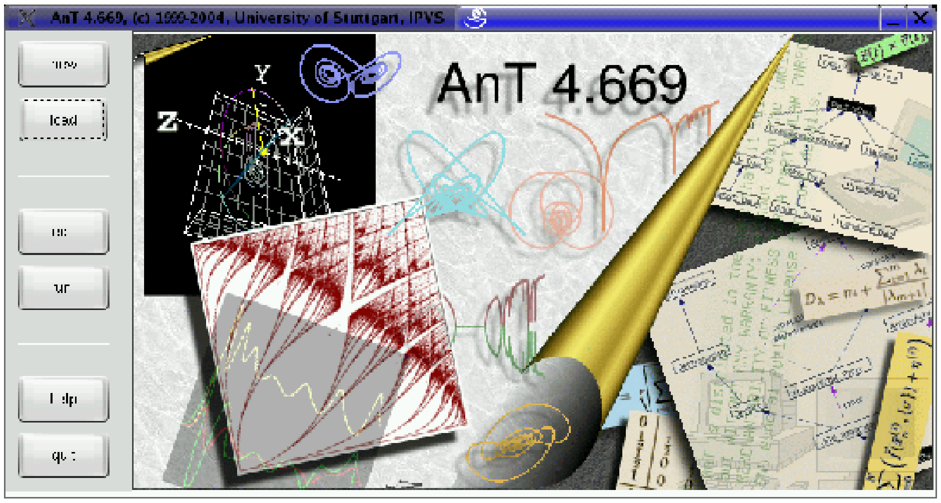


Figure 8.1: AnT-gui: main window

The `main` *window of the graphical user interface. One can start totally from scratch with a new initialization file or load an already existing one. After editing the file one can save it and start the computation.*

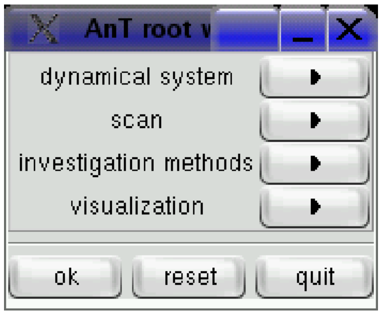
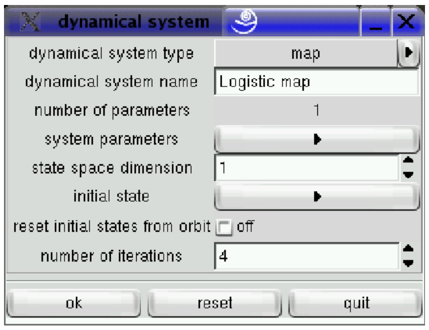


Figure 8.2: AnT-gui: AnT root window

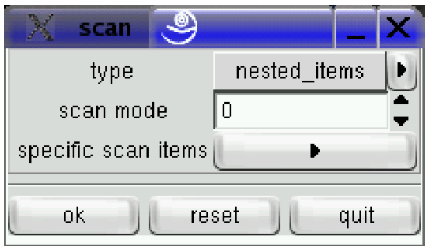
Like the initialization file itself, the `AnT root` *window consist of the four main parts. The dynamical system part, the scan part, the investigation methods part and the visualization part. These parts have to be specified, prior to a simulation and analysis run.*


dynamical system

dynamical system type	map
dynamical system name	Logistic map
number of parameters	1
system parameters	▸
state space dimension	1
initial state	▸
reset initial states from orbit	☐ off
number of iterations	4

ok reset quit

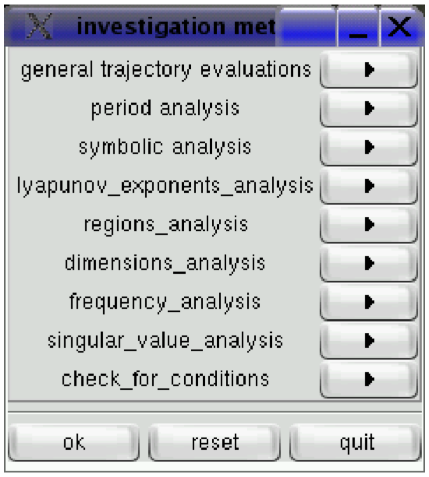
(a) dynamical system part


scan

type	nested_items
scan mode	0
specific scan items	▸

ok reset quit

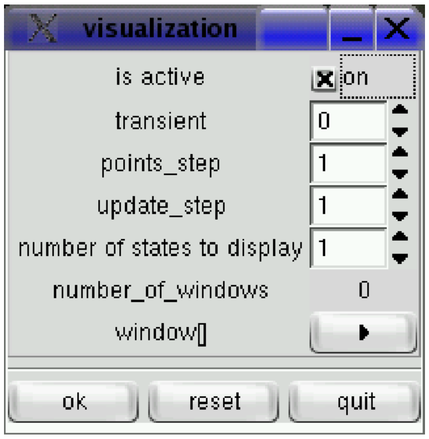
(b) scan part


investigation met

general trajectory evaluations ▸
period analysis ▸
symbolic analysis ▸
lyapunov_exponents_analysis ▸
regions_analysis ▸
dimensions_analysis ▸
frequency_analysis ▸
singular_value_analysis ▸
check_for_conditions ▸

ok reset quit

(c) investigation methods part


visualization

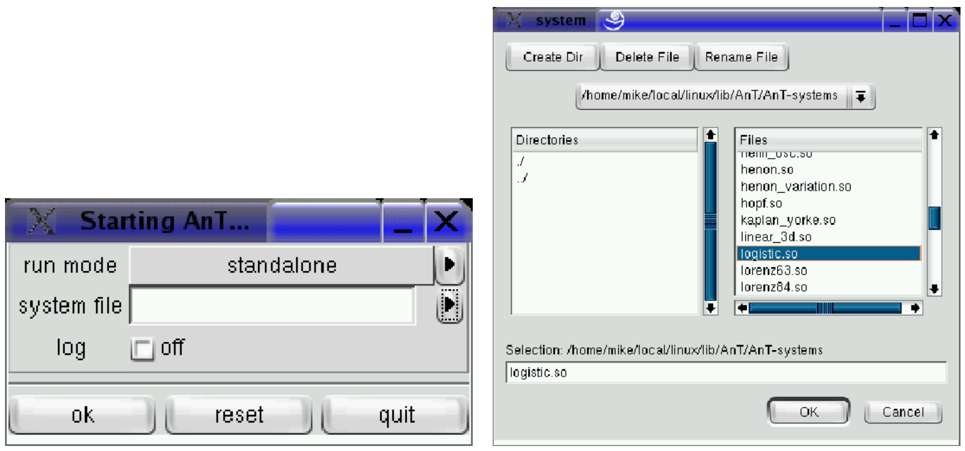
is active	☒ on
transient	0
points_step	1
update_step	1
number of states to display	1
number_of_windows	0
window[]	▸

ok reset quit

(d) visualization part

Figure 8.3: AnT-gui: main parts of the initialization

(a) *Screen shot of the* `dynamical system` *window*
(b) *Screen shot of the* `scan` *window*
(c) *Screen shot of the* `investigation methods` *window*
(d) *Screen shot of the* `visualization` *window*



run mode standalone
system file
log off

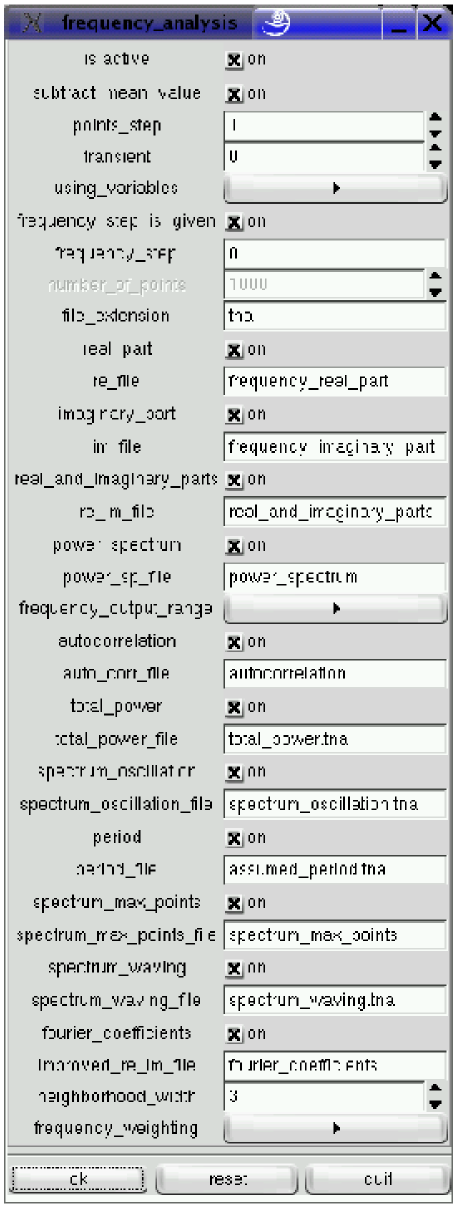
/home/mike/local/linux/lib/AnT/AnT-systems

Selection: /home/mike/local/linux/lib/AnT/AnT-systems
logistic.so

(a) choosing the run-mode (b) selecting the system function

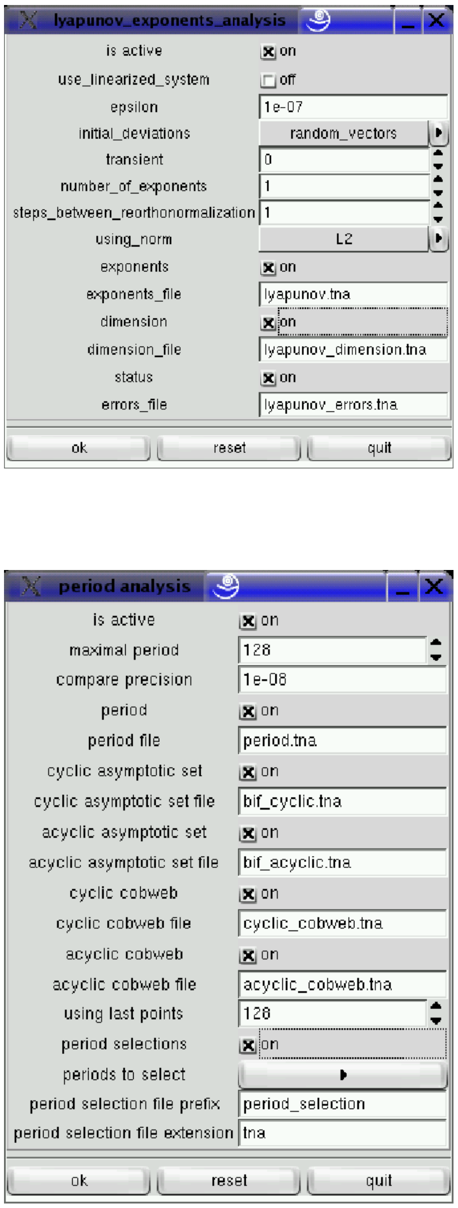
Figure 8.4: AnT-gui: running a system
In (a) a screen shot of the `Starting AnT` *window, where one can choose the run-mode, is presented. In (b) the* `system selector` *is shown, which allows the selection of a shared library file corresponding to the system function of the dynamical system to be simulated.*

whereas this chapter will be closed with figure 8.6, which shows a screen shot with a part of all implemented integration methods for dynamical systems continuous in time (see Sec. 3.4).



is active	on
subtract mean value	on
points_step	1
transient	0
using_variables	
frequency step is given	on
frequency_step	0
number_of_points	1000
file_extension	tna
real part	on
re_file	frequency_real_part
imaginary_part	on
im_file	frequency_imaginary_part
real_and_imaginary_parts	on
re_im_file	real_and_imaginary_parts
power spectrum	on
power_sp_file	power_spectrum
frequency_output_range	
autocorrelation	on
auto_corr_file	autocorrelation
total_power	on
total_power_file	total_power.tna
spectrum_oscillator	on
spectrum_oscillation_file	spectrum_oscillation.tna
period	on
period_file	assumed_period.tna
spectrum_max_points	on
spectrum_max_points_file	spectrum_max_points
spectrum_waving	on
spectrum_waving_file	spectrum_waving.tna
fourier_coefficients	on
improved_re_im_file	fourier_coefficients
neighborhood_width	3
frequency_weighting	

(a) frequency analysis window



is active	on
use_linearized_system	off
epsilon	1e-07
initial_deviations	random_vectors
transient	0
number_of_exponents	1
steps_between_reorthonormalization	1
using_norm	L2
exponents	on
exponents_file	lyapunov.tna
dimension	on
dimension_file	lyapunov_dimension.tna
status	on
errors_file	lyapunov_errors.tna

is active	on
maximal period	128
compare precision	1e-08
period	on
period file	period.tna
cyclic asymptotic set	on
cyclic asymptotic set file	bif_cyclic.tna
acyclic asymptotic set	on
acyclic asymptotic set file	bif_acyclic.tna
cyclic cobweb	on
cyclic cobweb file	cyclic_cobweb.tna
acyclic cobweb	on
acyclic cobweb file	acyclic_cobweb.tna
using last points	128
period selections	on
periods to select	
period selection file prefix	period_selection
period selection file extension	tna

(b) Lyapunov exponents and period analysis windows

Figure 8.5: AnT-gui: some investigation methods

(a) *Screen shot of the* `Lyapunov analysis` *window*
(b) *Screen shot of the* `period analysis` *window*
(c) *Screen shot of the* `frequency analysis` *window*

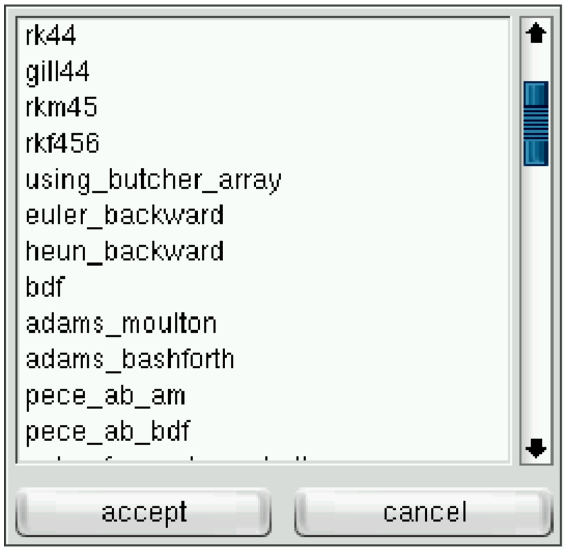

rk44
gill44
rkm45
rkf456
using_butcher_array
euler_backward
heun_backward
bdf
adams_moulton
adams_bashforth
pece_ab_am
pece_ab_bdf

Figure 8.6: AnT-gui: integration methods
This figure shows a screen shot of the selector, from which the integration method for the following time continuous classes: ODEs, CODELs, DDEs, CDDELs, FDEs and PDEs can be selected. The scrollbar on the right illustrates, that the **AnT computation engine** *provides a large number of integration methods.*

Chapter 9

Visualization of dynamical systems

9.1 Motivation

The importance of scientific visualization grows steadily and rapidly in the last years. The fast development of high performance graphics cards accelerates the development and the usage of visualization tools. One well-known example of such a visualization tool is the OpenGL cross-platform standard for 3D rendering and 3D hardware acceleration [124, 107]. This standard was used also for the visualization part of the **AnT 4.669** software package.

The main advantages of this visualization are on the one hand, that it will be done in the course of the simulation and on the other hand, that one can translate, rotate or scale an image, for instance of an attractor, in an interactive manner. To see, how the sometimes quite complex structure of a chaotic attractor emerges in the course of time and the possibility to change the view on it in different ways, is very useful for the understanding of this complex geometric structures. Although this two advantages cannot be illustrated here, in the next section some examples of the visualization are given.

9.2 The visualization capabilities of AnT 4.669

9.2.1 Visualizing trajectories

The aim of the current status was the possibility to visualize trajectories, that means the state of a dynamical system or some of its components in the course of time. The following two representations are supported:

1. Representation in the extended state space
2. Representation in the state space

Hereby the ***extended state space*** is defined as the Cartesian product of the state space with the time component. For instance, a time series of one state component, that is the plot of this component versus time, is the extended state space representation of this single component. Of course this can be done also with two components. The state space representation allows also two- or three-dimensional versions. Here the state components are plotted versus each other.

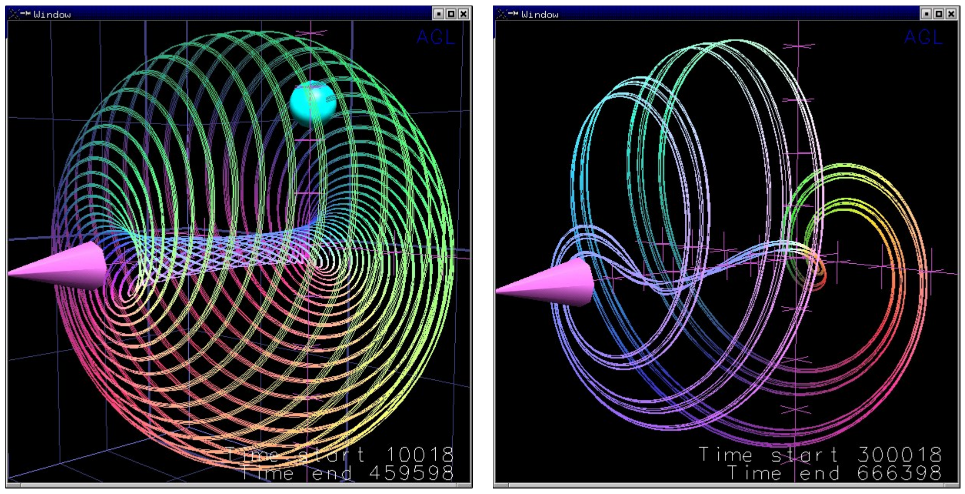
9.2.1.1 State space representation

In figure 9.1 the state space representation of two trajectories of the Aizawa system (see Sec. 11.4) for two different parameter settings are shown.

9.2.1.2 Extended state space representation

In figure 9.5 three time series of the Rössler system (see Sec. 11.5) are presented, which show the evolution in time of the three state variables or components of this system. The parameter setting is the same as in figure 9.3(a). As already mentioned, the behavior is in this case periodic.
As demonstrated, the **AnT 4.669** software package allows not only the saving of data produced by the simulation and the application of some investigation methods, but also the animated visualization of the trajectories of the considered dynamical system. This type of visualization is useful, because the animated representation of trajectories in the course of time helps to discover basic principles of the underlying dynamics.

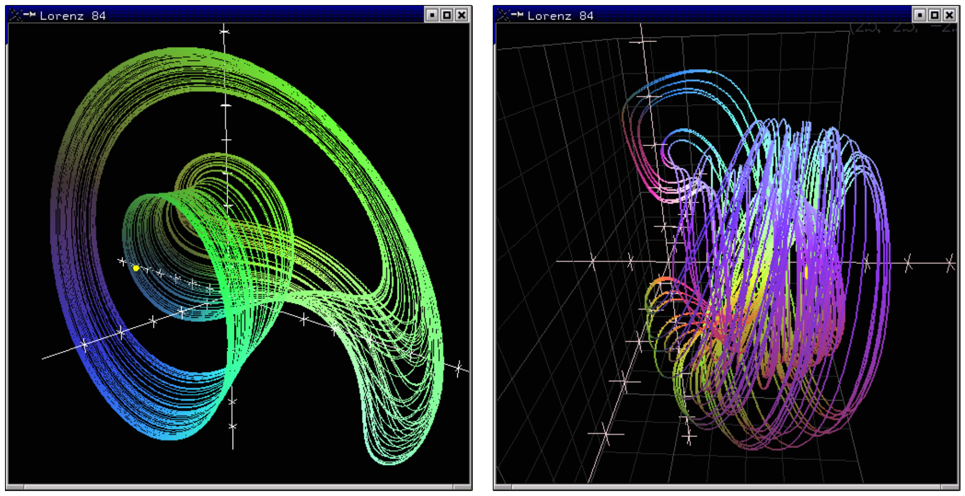
Chapter 9
Visualization of dynamical systems



(a) Aizawa system: chaotic attractor (b) Aizawa system: chaotic attractor

Figure 9.1: Aizawa system: example attractors

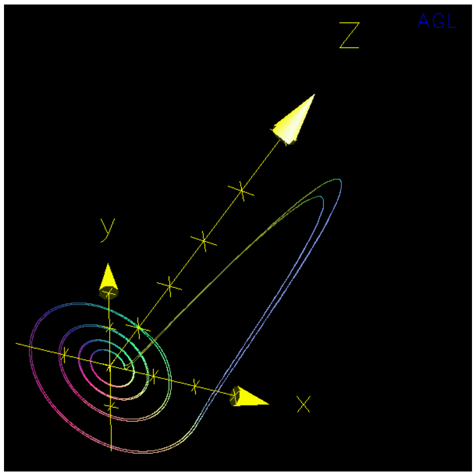
In (a), a very picturesque chaotic attractor of system (11.4) is shown, whereas in (b) a chaotic attractor near the end of a period-doubling scenario is presented.



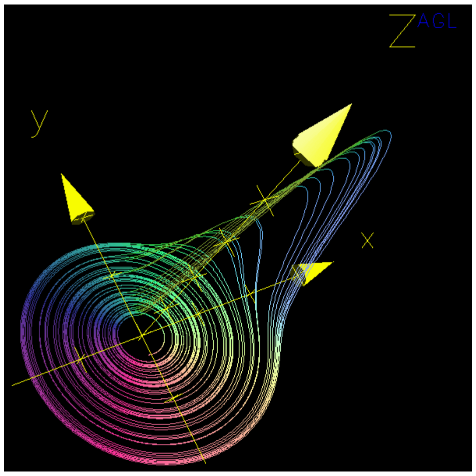
(a) Lorenz84 system: chaotic attractor (b) Lorenz84 system: chaotic attractor

Figure 9.2: Lorenz84 system: example attractors

Two different chaotic attractors of the Lorenz84 system (see Sec. 11.6) are shown.



(a) Rössler system: limit cycle

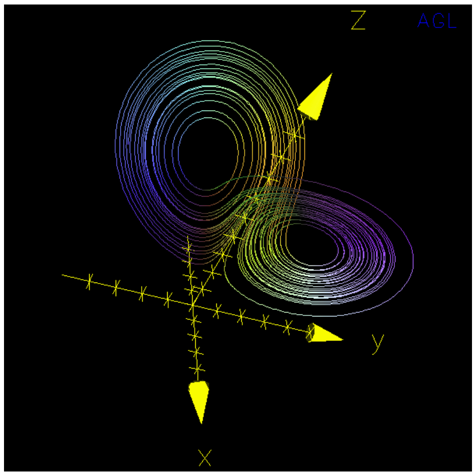


(b) Rössler system: chaotic attractor

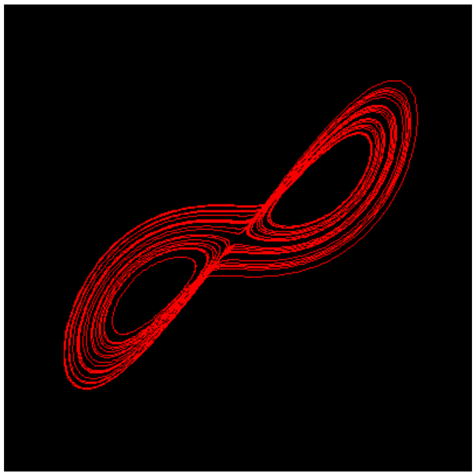
Figure 9.3: Rössler system: example attractors

In (a), a limit cycle with period two occurring in a window of the chaotic regime is shown (compare the Lyapunov spectrum in Fig. 11.14). In (b) a chaotic attractor of this system is presented. Parameter settings:

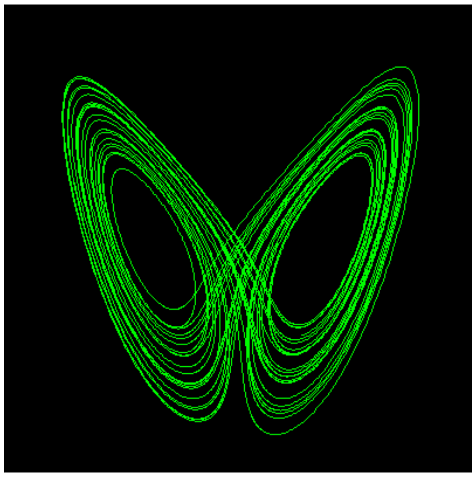
(a) $a = 0.15, b = 0.2, c = 18.35$
(b) $a = 0.15, b = 0.2, c = 12.0$



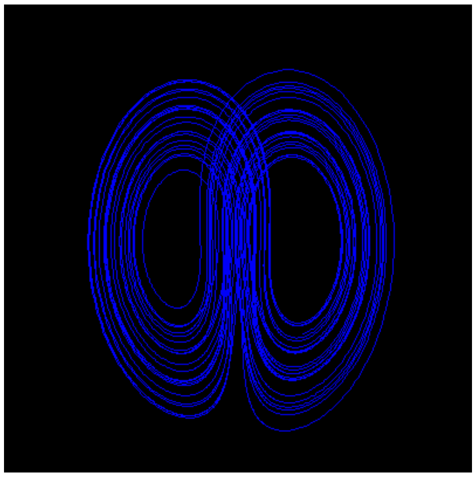
(a) Lorenz system: chaotic attractor



(b) Lorenz system: x versus y

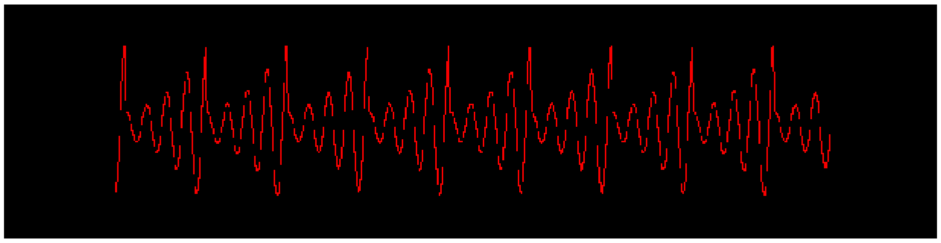


(c) Lorenz system: x versus z

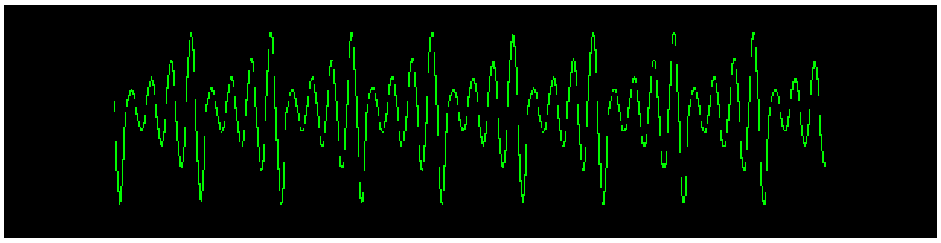


(d) Lorenz system: y versus z

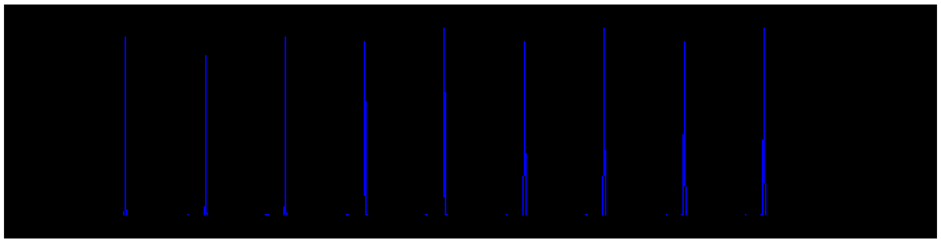
Figure 9.4: Lorenz system: chaotic attractor in 3D and 2D projections
(a) chaotic attractor at the parameter setting $\sigma = 16.0, r = 45.92, b = 4.0$
(b) xy projection
(c) xz projection
(d) yz projection



(a) Rössler system: x component



(b) Rössler system: y component



(c) Rössler system: z component

Figure 9.5: Rössler system: time series

Three periodic times series or extended state space representations of the state components of the Rössler system (11.5) are shown. The parameter setting is the same as in figure 9.3(a).

Chapter 10

Numerical aspects of simulation

10.1 General remarks about numerics

Whenever one is dealing with numeric computations at least three important aspects should be clear in one's mind:

- ▶ The first aspect is concerned with the available computational resources. Often the computer memory and performance are limiting factors of numerical computations. These limitations are in many cases generic or problem inherent and can't be avoided by simply using faster and more powerful computation sites. It is the numerical complexity of these problems which make them hard or even impossible to tract exactly. A well-known example of such kind of problems is finding an exact solution to the three index assignment problem (see Sec. 3.4.1.2 on page 75) at least for large problem sizes. This subclass of ***combinatorial optimization*** problems is ***NP-hard***, which means that there is no known deterministic algorithm, capable to solve these problems in polynomial computation time. However, such problems are then often treated by computing only approximative solutions which are in some sense precise or good enough.
- ▶ The second aspect is concerned with the reliability of the obtained results. It is by no means guaranteed that results obtained by numerical computations are exact or precise. There are well-known numerical phenomena like accumulation of small errors, which cause numerical

computations to fail. If one is aware of these difficulties, one is often able to get rid of them by using more accurate or precise calculation techniques. But this mostly requires more computational resources. One demonstrative example here is for instance the interval arithmetics [114, 90, 55, 56, 115]. The idea of this approach is not to perform calculations on the basis of single machine numbers, but instead with whole intervals. The advantage of this approach is, that at every step of the performed calculations it is guaranteed, that the exact result lies in the determined interval. However, the approach has also significant disadvantages: Obviously, the computations carried out in this way are much more time and memory consuming compared with computations based on single machine numbers. Furthermore, there is often an increase of the length of the intervals, making the results of large computations at least difficult to interpret. Hence, in most applications where numerical computations are involved, one has to make a suitable compromise between the precision and as a consequence the reliability and the usage of precise or very accurate computation techniques. On the one hand one wants to obtain the results as soon as possible and with a minimum amount of resources and on the other hand one wants the numerical results as precise as possible to meet the required reliability. Moreover in some cases even this strategy may fail, due to specific problem inherent characteristic properties. For instance, if one wants to calculate the trajectory of a chaotic attractor, this calculation can never be exact due to the sensitivity of the chaotic dynamics on small perturbations. Although it can be shown using the *shadowing theorem*, that there exists a real trajectory with a slightly different initial condition that stays near the numerically computed one, one has to be careful when interpreting the numerical computations.

- The third aspect is concerned with the fact, that in all numerical computations one is dealing with machine numbers. This may cause problems which will be discussed in more detail in the next sections.

10.2 Machine numbers

When using computers to solve a mathematical problem numerically, it is a matter of fact that all numbers one is dealing with have to be mapped to the computer memory. Hence, an internal representation of the numbers is required. It is also a matter of fact, that the set of real numbers and even an open or closed subset of real numbers is uncountable. Therefore, it is clear that not all real numbers in a certain range or a certain interval can be represented by a set of machine numbers, which of course is always a countable set. Of course there exist several internal representations of real numbers. Here only the representation of floating point numbers according to the IEEE 754 standard is considered.

10.2.1 The IEEE 754 standard for the representation of floating point numbers

A normalized floating point number according to the IEEE 754 standard is defined by the triple (s, m, e):

$$\begin{aligned}(s,m,e) &= (-1)^s\, m\, 2^e \qquad \text{with} \\ m &= 1.m_{-1}m_{-2}\dots m_{-k} = 1 + \sum_{i=-1}^{-k} m_i 2^i \qquad \text{and} \\ e &= \sum_{i=0}^{n-1} e_i 2^i - b\end{aligned} \tag{10.1}$$

Hereby normalized means, that the mantissa has an *invisible* or *hidden* leading bit and all bits of the mantissa field are interpreted as decimal bits of the dual number representation of the mantissa. The bit s determines the sign of the number f, n is the number of bits reserved for the representation of the exponent and k is the number of bits reserved for the normalized representation of the mantissa. All bits of the exponent field represent an unsigned integer value. Hence to allow also negative exponents in the representation of floating point numbers the bias value b is used.
Properties of the IEEE 754 standard:

- The normalized representation is unique
- Existence of range limiting values, that is a smallest and a largest representable floating point number

bit number	$k+n$	$k+n-1$	$\ldots$	k	$k-1$	$\ldots$	0
denotation	sign	exponent e			mantissa m		
	s	e_{n-1}	$\ldots$	e_0	m_{-1}	$\ldots$	m_{-k}

Table 10.1: The IEEE 754 standard for the representation of floating point numbers

- ▶ Not all numbers in the range between the limiting values are representable
- ▶ The representable floating point numbers are not uniformly distributed within the range of the limiting values
- ▶ The range where the representable floating point numbers are most dense is around the value zero
- ▶ Arithmetic operations are not closed, that is the result of an arithmetic operation of two representable floating point numbers may by an unrepresentable floating point number
- ▶ The mathematical laws of associativity and distributivity are not valid

10.2.2 Single, double and extended precision floating point numbers

For the single (double) precision floating point representation, the number of bits in the exponent field is $n = 8$ ($n = 11$) and the number of bits representing the mantissa is $k = 23$ ($k = 52$) so that in total 32 (64) bits, that is 4 (8) bytes are required. The value of the bias is given by $b = 2^{n-1} - 1$, that is 127 for the single and 1023 for the double precision floating point representation.

10.2.3 Arbitrary precision

Using computer algebra systems like Maple [105] or Mathematica [106], which allow symbolic manipulations, one can perform computations with in principle arbitrary precision. However, the precision is confined due to limited resources like computer memory and performance.

10.3 Reliability aspects of numeric computations

To demonstrate that encountering numerical problems when dealing with machine numbers is not at all an exceptional case the following examples are considered:

1. Addition of floating point numbers (see also chapter B): Using single floating point precision according to the IEEE 754 standard the simple addition of the two numbers 16777216 and 1 leads to the result 16777216 which is obviously wrong.

   Although both numbers are representable floating point numbers the result of the addition is not a representable floating point number. In this case, the two results are mapped on the same machine number. The floating point numbers to be added are 2^{24} and 2^0. Their machine representations according to the IEEE 754 standard are shown in Table 10.2.

bit	31	30	...	23	22	...	0
denotation	sign	exponent e			mantissa m		
2^0	0	01111111			00000000000000000000000		
2^{24}	0	10010111			00000000000000000000000		

Table 10.2: The floating point representations of 2^0 and 2^{24} according to the IEEE 754 standard

   As one can see, they differ only in their exponents. The addition procedure now works in three steps. In the first step the smaller exponent is equalized to the larger exponent whereby the mantissa has to be adjusted. In the second step the mantissas are added and in the third step the result is rounded if necessary and normalized. Here the exponents of the two numbers differ by the value 24. Hence to adjust the mantissa of the smaller number the *hidden* bit has to be shifted to the right 24 times. The first shift leads to the appearance of the *hidden* bit in the mantissa but 23 shift operations remain. Because there are only 22 places in the mantissa left, the last shift operation shifts the former *hidden* bit out of the mantissa and hence the number zero is added to the larger number leading to the wrong result.

2. Pure numerical determination of eigenvalues:

   Consider the eigenvalue problem of the following matrix:

$$\underline{\underline{M}} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 2 & 5 & 6 \end{bmatrix} \qquad (10.2)$$

The characteristic polynomial of this matrix is

$$p(\lambda) = -\lambda^3 + 9\lambda^2 + 3\lambda\,, \qquad (10.3)$$

from which one can calculate analytically the eigenvalues

$$\begin{aligned} \lambda_1^a &= \frac{1}{2}(9-\sqrt{93}) \approx -0.3218253805, \\ \lambda_2^a &= 0, \\ \lambda_3^a &= \frac{1}{2}(9+\sqrt{93}) \approx 9.321825380\,. \end{aligned} \qquad (10.4)$$

Using the program *Maple 8* with an arbitrarily chosen accuracy of 10 digits for floating point arithmetics, one encounters the following strange results for the real-valued matrix

$$\underline{\underline{M}} = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 2.0 & 5.0 & 6.0 \end{bmatrix} \qquad (10.5)$$

Calculating only the eigenvalues using the *Maple* built-in function *eigenvals* one obtains:

$$\begin{aligned} \lambda_1^n &= -0.3218253805 \\ \lambda_2^n &= 7.700622123\ 10^{-16} \\ \lambda_3^n &= 9.321825380 \end{aligned} \qquad (10.6)$$

which is in acceptable coincidence with the exact result (10.4). Whereas one gets the following results when calculating the eigenvalues and eigenvectors using the *Maple* built-in function *eigenvects*:

$$\begin{aligned}
\tilde{\lambda}_1^n &= -0.3218253804, \\
\underline{v}_1 &= [0.5567547066, 0.5567547150, -0.6164806444] \\
\tilde{\lambda}_2^n &= 4.28431003\ 10^{-9}, \\
\underline{v}_2 &= \left[0.9486832980, -4.625747414\ 10^{-10}, -0.3162277652\right] \\
\tilde{\lambda}_3^n &= 9.321825393, \\
\underline{v}_3 &= [-0.3940365890, -0.3940365890, -0.8303435024]
\end{aligned}$$

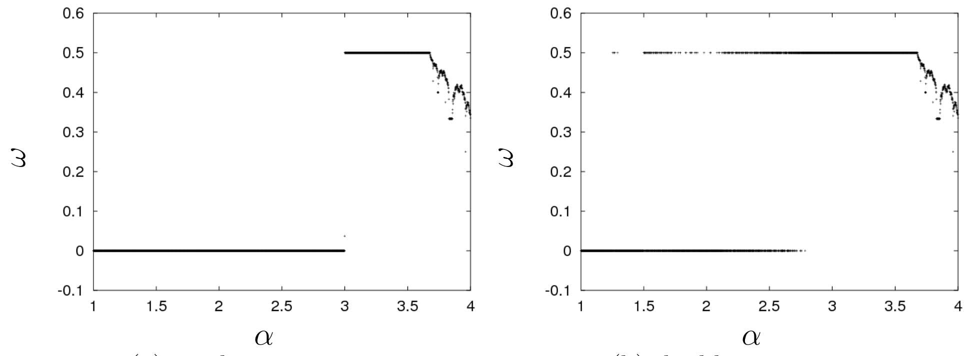
Especially the calculation of zero eigenvalues is numerically sometimes very difficult. In the considered case also the product of the matrix $\underline{\underline{M}}$ with the eigenvector $\underline{v}_2$ differs significantly from the product of the eigenvalue $\tilde{\lambda}_2^n$ with the eigenvector $\underline{v}_2$ although the results should be the same.

3. The calculation of wave numbers

   To demonstrate the fact, that using higher arithmetic precision leads not necessarily and in any case to more accurate results, the following example is considered. In this specific case the results of double precision calculations are incorrect, whereas the results of the single precision is not. Of course, this is not caused by the used precision itself but by the used algorithm. However, it is a demonstration, that one has always to interpret the results in context with the used precision and the used algorithm.

   If one compares the single and double precision results of the ***wave number*** calculation (see Sec. 5.2.3) of the logistic map (see Sec. 3.3.1.1), one observes (see Fig. 10.1), that the result of the more accurate double precision calculation is at some parameter values totally incorrect, whereas the result of the single precision calculation is not.

   This strange result is caused by numerically induced oscillations when using the double precision floating point representation, although it can be shown analytically, that for $\alpha \leq 3$ the only stable attractor of this system is a fixed point (see Fig. 3.1). The oscillations take place in the last digit of the double precision floating point representation causing



(a) single precision (b) double precision

Figure 10.1: Logistic map: wavenumber ω

In (a) the calculation of the wavenumber ω of the logistic map was done with single precision which leads to the correct result, whereas in (b) the calculation of the wavenumber was done in double precision which leads to the incorrect result.

> the wrong calculation of the wave number. Of course this definitely wrong result could be avoided when using a more sophisticated method to calculate the wave number, but it demonstrates nevertheless that one has to be very careful when interpreting numerical results, because one gets the correct result when using only the single precision floating point representation and the same algorithm.

As one can see, one has to be very carful when relying naively on numerical results. Especially results obtained by excessive numerical computations have to be interpreted thoroughly and not to be taken as a fact. Whenever one have the possibility to proof some results analytically one should do this in order to justify and validate the applied numerical algorithm.

Chapter 11

Examples

To demonstrate and illustrate the various and numerous capabilities of the **AnT 4.669** software package, examples with different applied investigation methods (see Chap. 5) are presented in this chapter as well as an example of an extensively investigated system. The examples may be the result of single simulation runs, one-dimensional scan-runs as well as multi-dimensional scan-runs. The results were obtained either by using the standalone mode on a single computation node or by using the client/server mode on several nodes of a heterogeneous environment - for instance different workstations with different operating systems - or of a homogeneous environment for instance a cluster.

11.1 Logistic map

The first example in this section belongs to the class of ordinary maps (see Sec. 3.3.1.1)). Figure 11.1 shows the period doubling scenario of the logistic map already introduced in section 3.3.1.1, when varying the single system parameter α from $\alpha_1 = 3$ to $\alpha_\infty \approx 3.569945672$. In this doubly logarithmic representation, the first eight bifurcations of this period doubling scenario together with the corresponding periods can be shown. This figure shows clearly the self-similarities connected with the famous scaling properties of this dynamical system.

In figure 11.2 a comparison of the numerically calculated value of the Lyapunov exponent at the parameter value $\alpha = 4$ with the analytically cal-

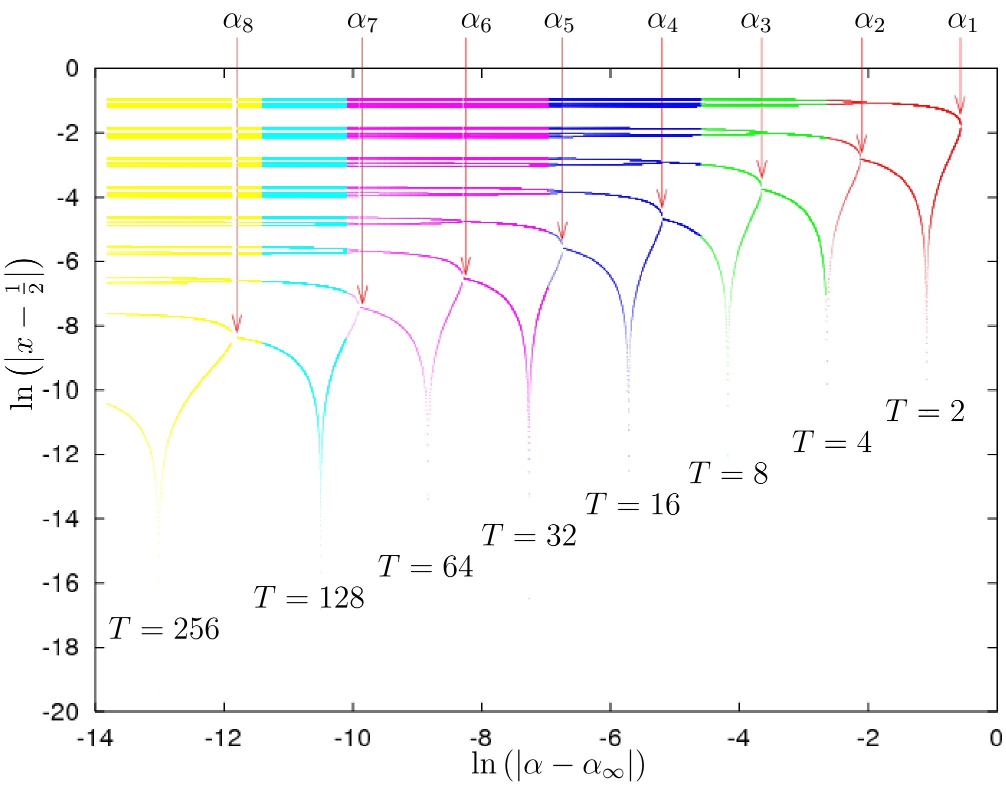


Figure 11.1: Logistic map: period doubling scenario

The doubly logarithmic representation of the period doubling scenario allows the presentation of the first eight period doubling bifurcation points at α_1, α_2, $\ldots \alpha_8$, together with the corresponding periods of the stable periodic behavior between two bifurcation points.

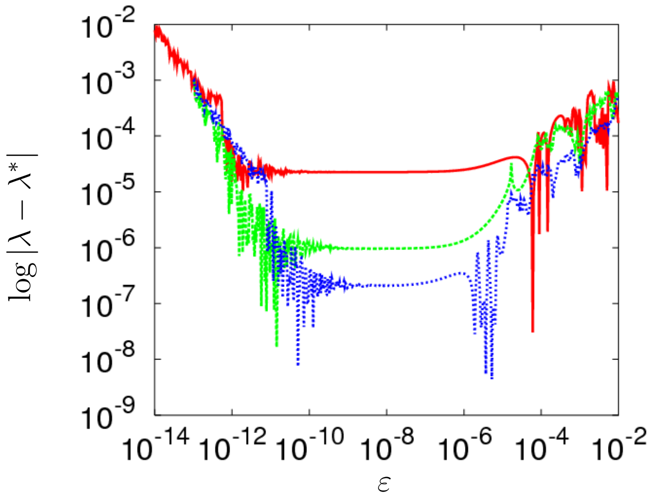


Figure 11.2: Comparison of analytic and numeric results at $\alpha = 4$
The figure shows a comparison of analytic and numeric results of the Lyapunov exponent calculation for the logistic map at $\alpha = 4$*. The presented values are obtained within* 25000 *(red line),* 250000 *(green line) and* 2500000 *(blue line) iteration steps.*

culated value is shown. Presented are three one-dimensional investigation method parameter scans for three different number of iterations whereby in each scan the difference to the exact value of $\lambda^* = \log 2$ in dependence of the investigation method parameter ε is plotted. For the numerical calculation of the Lyapunov exponent, the method without using the linearized system function is applied, whereby instead an adjacent trajectory with initial distance ε is tracked and the divergence rates are averaged in the course of time. After each iteration, the distance is re-scaled to the initial distance ε. The mathematical definition of the Lyapunov exponents requires in principle the limit $\varepsilon \to 0$ to be taken. Figure 11.2 demonstrates drastically, that there exist in fact upper and lower bounds for the investigation method parameter ε. If ε is chosen too small, the algorithm runs into numerical problems and an ε value chosen too large leads to a bad approximation of the Lyapunov exponent by definition. Hence, there exists a suitable range for this investigation method parameter which one is able to determine with the **AnT 4.669** package by such a specific scan.

11.2 The Gingerbreadman map

In this example the ***Gingerbreadman map*** [175] is investigated with the period analysis (see Sec. 5.3)

This two-dimensional dynamical system discrete in time is defined by:

$$x_{n+1} = 1 - y_n + |x_n| \tag{11.1}$$

$$y_{n+1} = x_n \tag{11.2}$$

Accordingly, it has the state space dimension $N_s = 2$ and the parameter space dimension $N_p = 0$. In figure 11.3, the result of the period analysis applied to a two-dimensional initial value scan is shown, whereby periods up to 246 were detected. In figure 11.3(a), the initial values were varied from -20 to 40 using 481 scan points, whereas in figure 11.3(b), the same range was used with only 480 scan points. The drastic difference in the figures is caused by the properties of the system where the period detection is very sensitive with respect to the initial values. In figure 11.3(a) the scan point increments are exact powers of 2. Because the initial scan points are also sums of powers of 2, all the scan points are sums of powers of 2 and hence pure machine numbers. In figure 11.3(b) the scan point increments are not powers of 2 and hence, the scan points aren't sums of powers of 2 and may be even not representable as machine numbers. This leads to the different detection of periodic orbits in this system.

11.3 Quadratic map CML

The following coupled map lattice is based on the well-known quadratic map, which is connected via a diffeomorphism with the logistic map presented in section 3.3.1.1. This system is used in [85] to illustrate the universality classes of the dynamic behavior of coupled map lattices. This universality classes are shown here together with the corresponding Lyapunov spectrum and with some spatial return maps. The system is defined by:

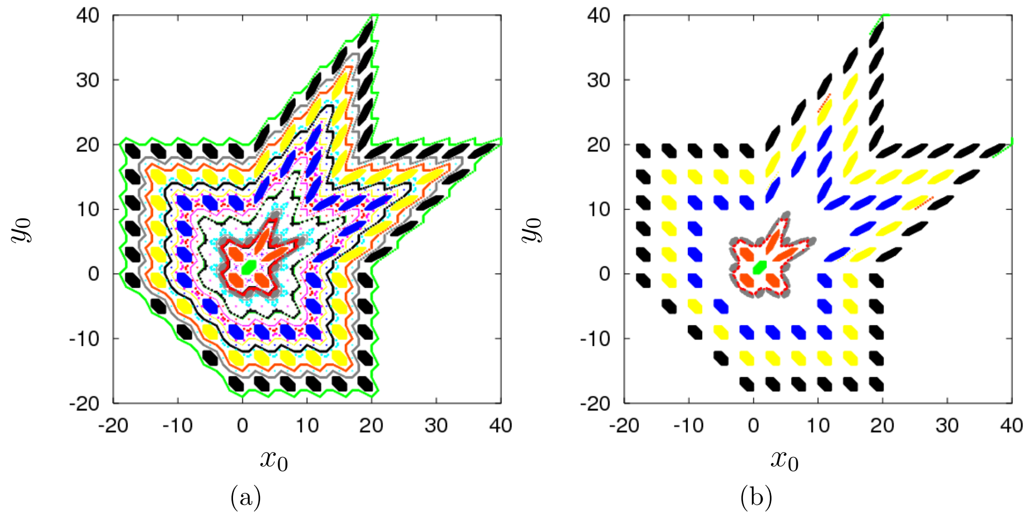


Figure 11.3: Period analysis of the gingerbread man map.
In (a) the initial values were varied from -20 to 40 using 481 scan points, whereas in (b) the same range was used with 480 scan points. For a detailed description of this strange phenomenon see text.

Definition: ***quadratic map CML***

$$\begin{aligned} x^i_{n+1} = \; & (1-\gamma)f(x^i_n,\alpha) + \\ & \frac{\gamma}{2}\left(f(x_n^{1+((i-2) \bmod M)},\alpha) + f(x_n^{1+(i \bmod M)},\alpha)\right) \end{aligned} \tag{11.3}$$

with

$$f(x,\alpha) = 1 - \alpha x^2\,, \quad i = 1,\ldots,M\,, \quad \underline{p} = (\gamma,\alpha) \in \mathbb{R}^2$$

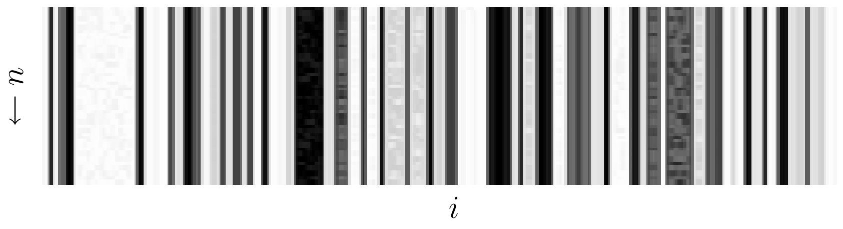
■

This system has three parameters, namely the coupling parameter γ, the parameter α of the single cells and finally the number M of cells.

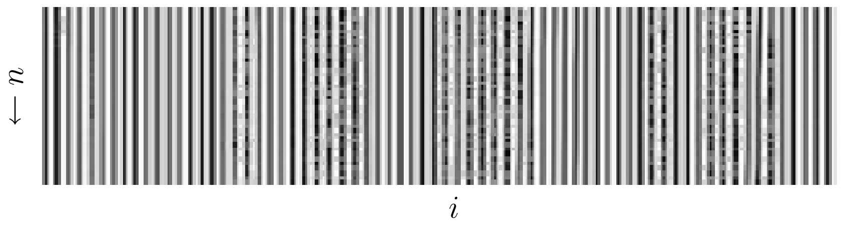
11.3.1 Universality classes of CMLs

In this section single simulation runs with fixed parameters, illustrating the different universality classes of the spatio-temporal asymptotic dynamics of coupled map lattices are presented (see Figs. 11.4 and 11.5). The figures are grey scale representations whereby the discrete time n runs, as indicated, from the top to the bottom. From the left ($i = 1$) to the right ($i = M$) runs the cell index i of the CML. The values of the states of the single cells x_n^i define the corresponding grey scale value.

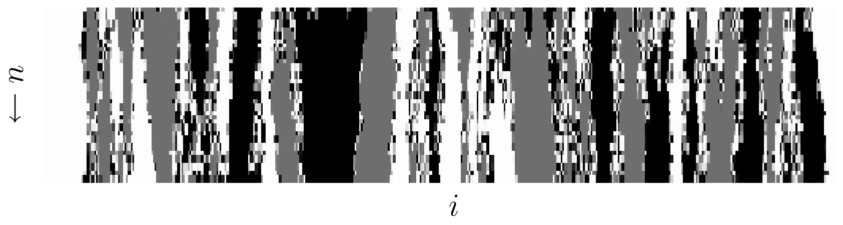
- **Frozen Random Pattern**
  See figure 11.6
- **Pattern Selection**
  See figure 11.7
- **Spatiotemporal Intermittency Type I**
  See figure 11.8
- **Spatiotemporal Intermittency Type II**
  See figure 11.9
- **Zigzag Pattern**
  See figure 11.10
- **Traveling Wave**
  See figure 11.11
- **Fully Developed Chaos**
  See figure 11.12



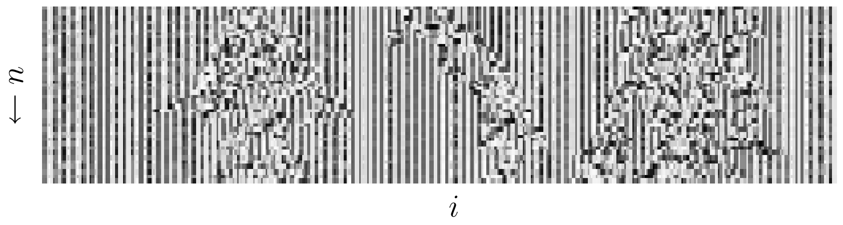
(a) Frozen random



(b) Pattern selection

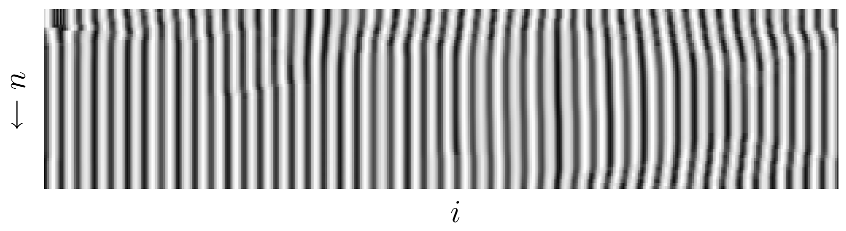


(c) Spatiotemporal intermittency (type I)

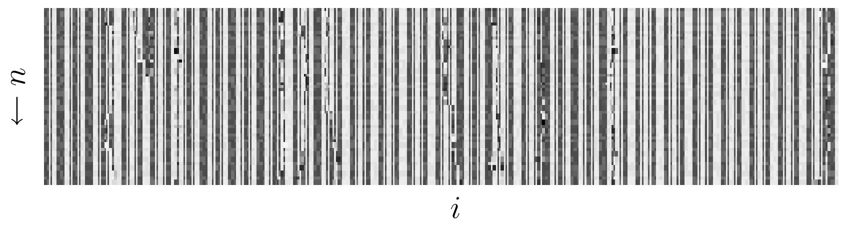


(d) Spatiotemporal intermittency (type II)

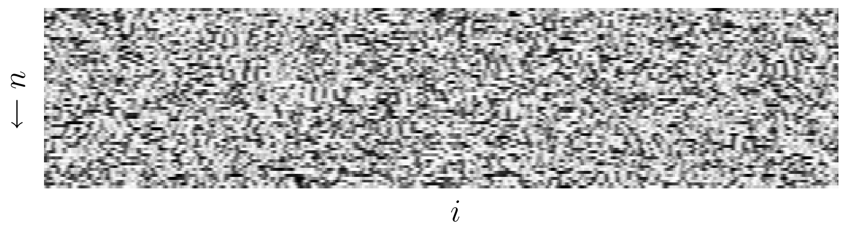
Figure 11.4: CML: universality classes I



(a) Traveling wave

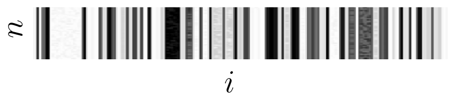


(b) Zigzag

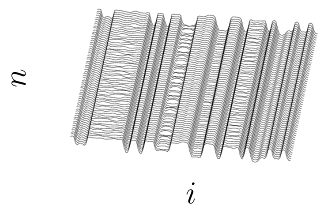


(c) Fully developed spatio-temporal chaos

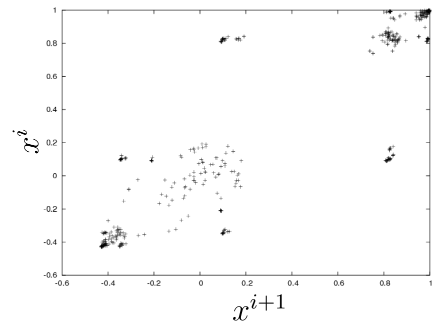
Figure 11.5: CML: universality classes II



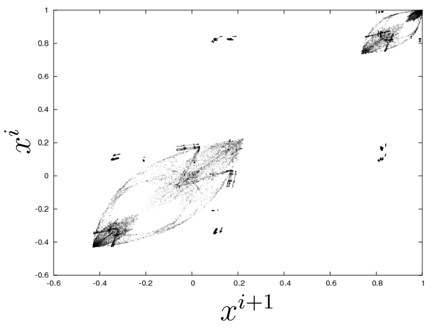
(a) Space-time grey scale plot: $M = 500$, starting at the top with iteration 3000, every 32nd time step is shown



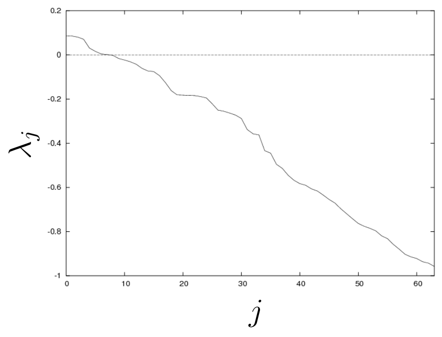
(b) Space-time plot with lines: 128 sites of the run of figure 11.6(a) are shown



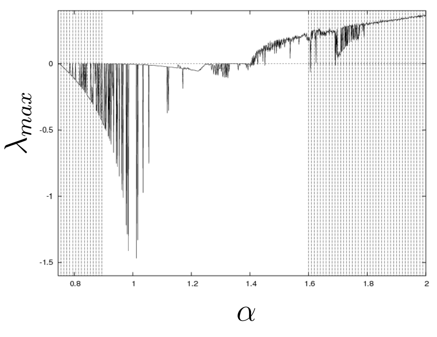
(c) Spatial return map at iteration 5000: $M = 500$



(d) Spatial return map at iteration 5000: $M = 50000$

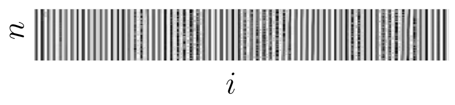


(e) Lyapunov spectrum: $M = 64$ and transient is 2000

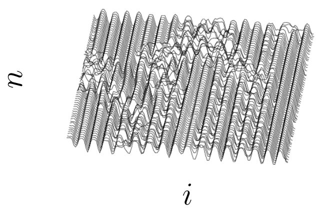


(f) Unmarked region: approximately the interval where the frozen random pattern can be found, depends also on the coupling parameter γ

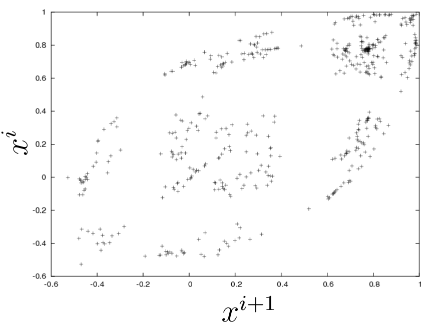
Figure 11.6: Characteristics of the frozen random pattern ($\alpha = 1.43$, $\gamma = 0.3$)



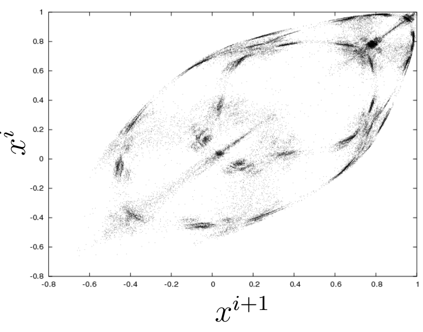
(a) Space-time grey scale plot: $M = 500$, starting at the top with iteration 3000, every 32nd time step is shown



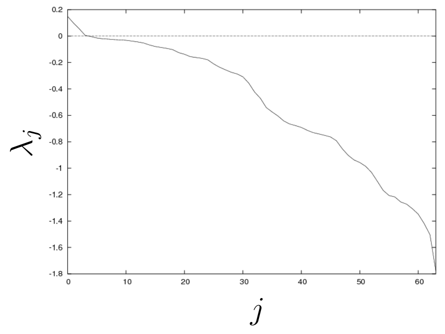
(b) Space-time plot with lines: 128 sites of the run of figure 11.7(a) are shown



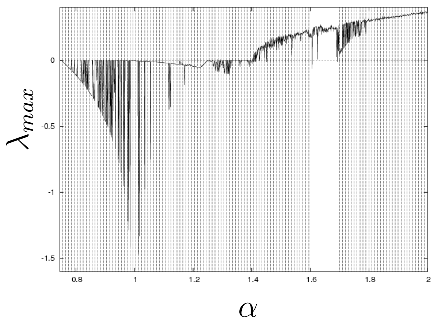
(c) Spatial return map at iteration 5000: $M = 500$



(d) Spatial return map at iteration 5000: $M = 50000$

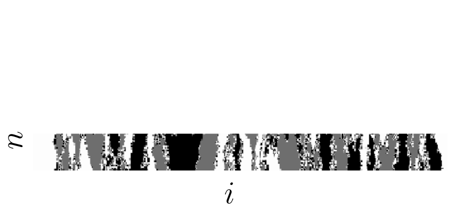


(e) Lyapunov spectrum: $M = 64$ and transient is 2000

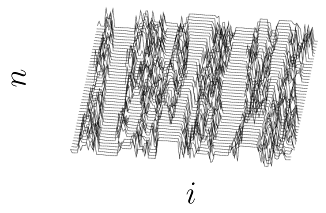


(f) Unmarked region: approximately the interval where the pattern selection can be found, depends also on the coupling parameter γ

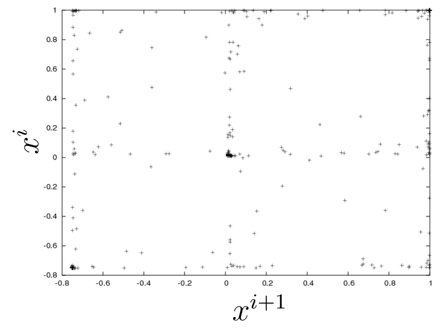
Figure 11.7: Characteristics of pattern selection ($\alpha = 1.7$, $\gamma = 0.4$)



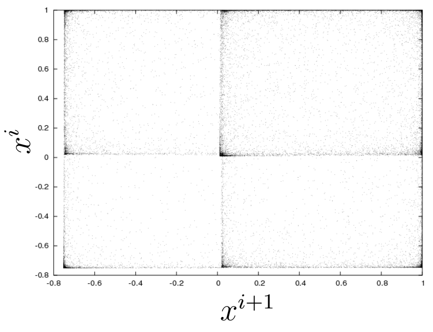
(a) Space-time grey scale plot: $M = 500$, starting at the top with iteration 3000, every 45th time step is shown



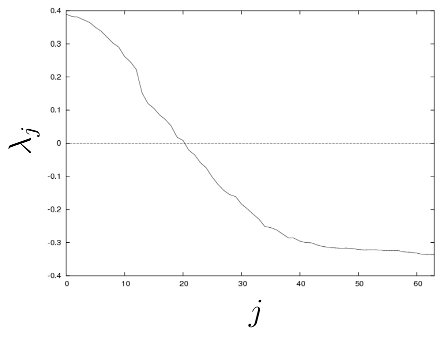
(b) Space-time plot with lines: 128 sites of the run of figure 11.8(a) are shown



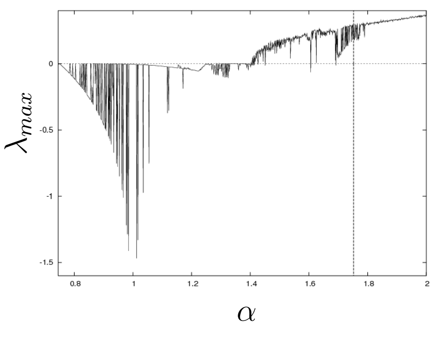
(c) Spatial return map at iteration 5000: $M = 500$



(d) Spatial return map at iteration 5000: $M = 50000$

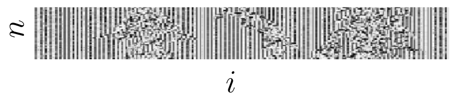


(e) Lyapunov spectrum: $M = 64$ and transient is 2000

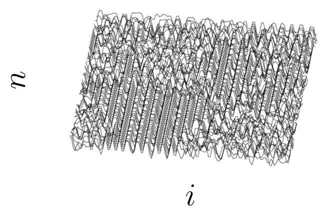


(f) Marked region: approximately the interval where spatiotemporal intermittency of type-I can be found, depends also on the coupling parameter γ

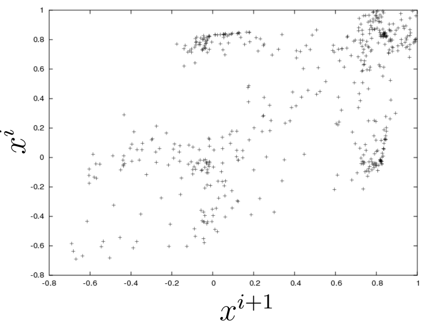
Figure 11.8: Characteristics of the spatiotemporal intermittency of type-I ($\alpha = 1.752$, $\gamma = 0.001$)



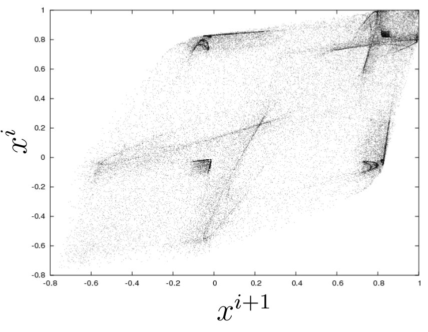
(a) Space-time grey scale plot: $M = 500$, starting at the top with iteration 3000, every 32nd time step is shown



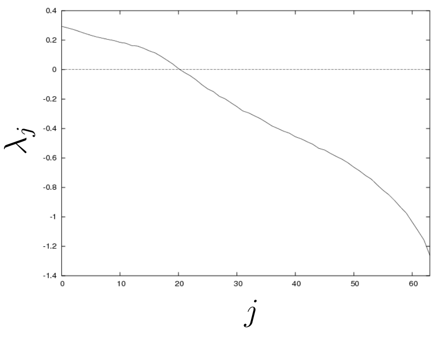
(b) Space-time plot with lines: 128 sites of the run of figure 11.9(a) are shown



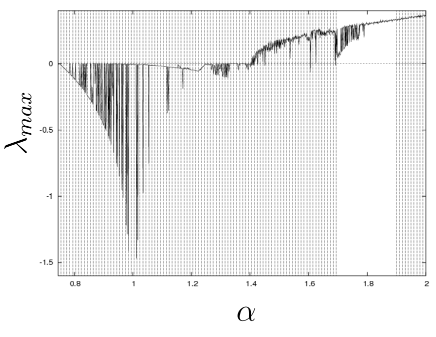
(c) Spatial return map at iteration 5000: $M = 500$



(d) Spatial return map at iteration 5000: $M = 50000$



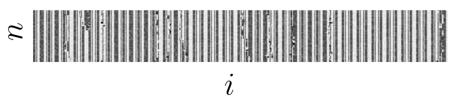
(e) Lyapunov spectrum: $M = 64$ and transient is 2000



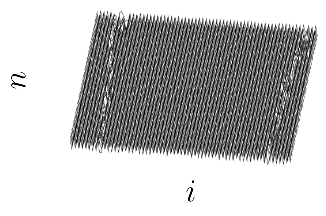
(f) Unmarked region: approximately the interval where spatiotemporal intermittency of type-II can be found, depends also on the coupling parameter γ

Figure 11.9: Characteristics of spatiotemporal intermittency of type-II ($\alpha = 1.77$, $\gamma = 0.3$)

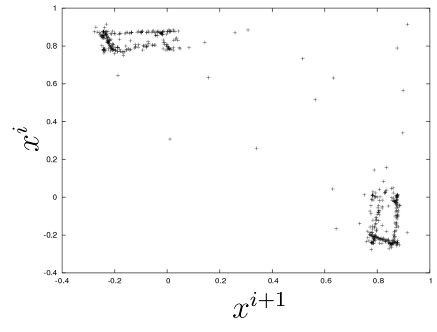
Chapter 11
Examples



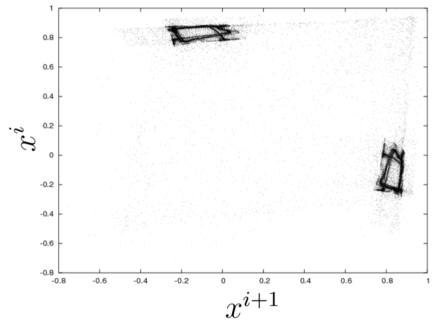
(a) Space-time grey scale plot: $M = 500$, starting at the top with iteration 3000, every 32nd time step is shown



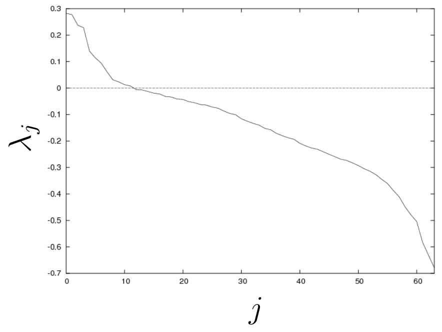
(b) Space-time plot with lines: 128 sites of the run of figure 11.10(b) are shown



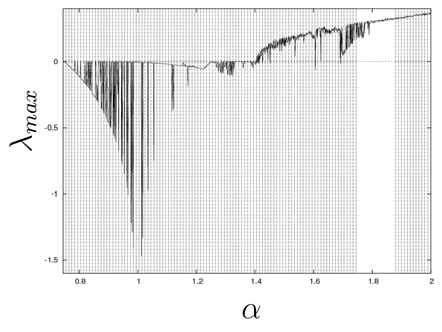
(c) Spatial return map at iteration 5000: $M = 500$



(d) Spatial return map at iteration 5000: $M = 50000$

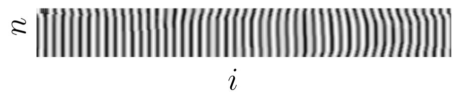


(e) Lyapunov spectrum: $M = 64$ and transient is 2000

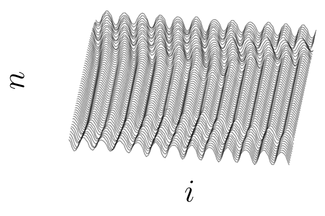


(f) Unmarked region: approximately the interval where the zigzag pattern can be found, depends also on the coupling parameter γ and the number of iterations

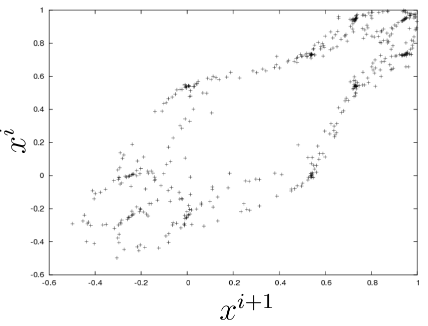
Figure 11.10: Characteristics of the zigzag pattern ($\alpha = 1.79$, $\gamma = 0.1$)



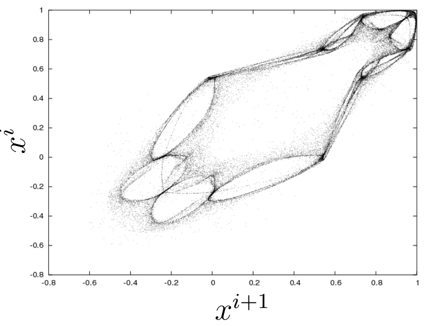
(a) Space-time grey scale plot: $M = 500$, starting at the top with iteration 5000, every 256th time step is shown



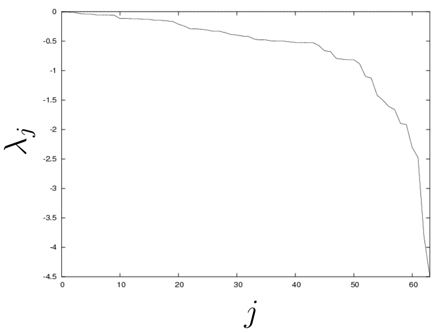
(b) Space-time plot with lines: 128 sites of the run of figure 11.11(a) are shown



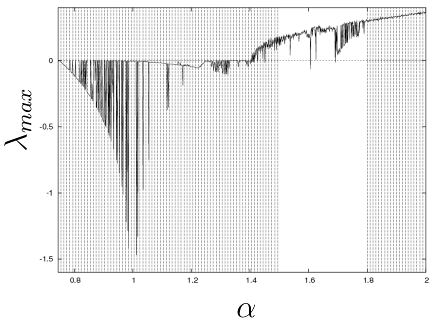
(c) Spatial return map at iteration 5000: $M = 500$



(d) Spatial return map at iteration 5000: $M = 50000$

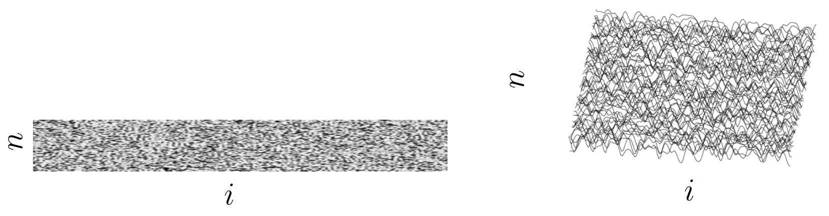


(e) Lyapunov spectrum: $M = 64$ and transient is 2000



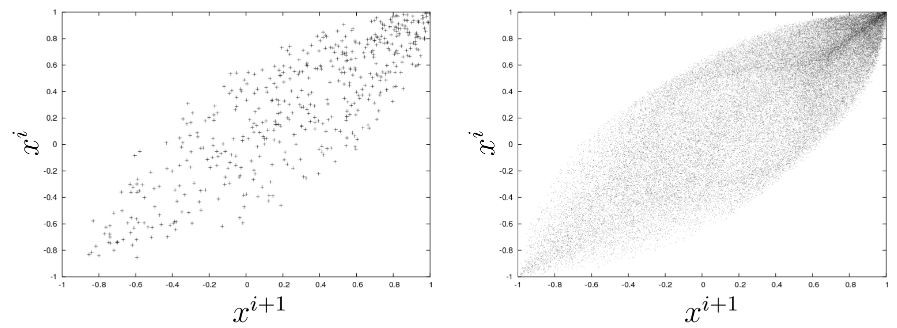
(f) Unmarked region: approximately the interval where the traveling wave can be found, depends also on the coupling parameter γ and the lattice size

Figure 11.11: Characteristics of the traveling wave ($\alpha = 1.69$, $\gamma = 0.9$)



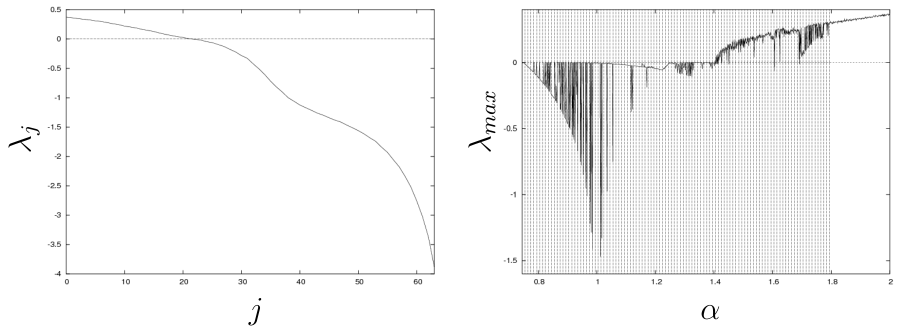
(a) Space-time grey scale plot: $M = 500$, starting at the top with iteration 3000, every 32nd time step is shown

(b) Space-time plot with lines: 128 sites of the run of figure 11.12(a) are shown



(c) Spatial return map at iteration 5000: $M = 500$

(d) Spatial return map at iteration 5000: $M = 50000$



(e) Lyapunov spectrum: $M = 64$ and transient is 2000

(f) Unmarked region: approximately the interval where fully developed chaos can be found, depends also on the coupling parameter γ

Figure 11.12: Characteristics of fully developed chaos ($\alpha = 1.99$, $\gamma = 0.6$)

11.4 The Aizawa system

This dynamical system belongs to the class of ordinary differential equations (see Sec. 3.4.1.1). It has a state space dimension of $N_s = 3$ and a parameter space dimension of $N_p = 6$. It exhibits picturesque chaotic attractors shown in chapter 9. The system is defined by:

Definition: ***Aizawa system***

$$\begin{aligned}
&\frac{d}{dt}\underline{s}(t) = \underline{f}(\underline{s}(t), \underline{p})\,, \quad \underline{s}(t) = (x(t), y(t), z(t))^T \in \mathbb{R}^3 \\
&\quad \underline{p} = (\alpha, \beta, \lambda, \omega, \rho, \epsilon) \in \mathbb{R}^6 \\
&\quad \underline{f}(x, y, z, \underline{p}) = \begin{pmatrix} (z-\beta)x - \omega y \\ \omega x + (z-\beta)y \\ \lambda + \alpha z - \frac{1}{3}z^3 - (x^2+y^2)(1+\rho z) + \epsilon z x^3 \end{pmatrix}
\end{aligned} \tag{11.4}$$

■

In figure 11.13, the three Lyapunov exponents of this dynamical system continuous in time are shown, whereby the system parameter ϵ was varied in the interval $\epsilon \in [0.02, 0.1]$ using $N^s = 4000$ scan points.

11.5 The Rössler system

This example belongs again to the class of ordinary differential equations (see Sec. 3.4.1.1). Like the Lorenz system (see Sec. 3.4.1.1) it is a well-known system in the field of nonlinear dynamics. It was discovered by Rössler in 1976 [144], as he investigated numerically certain types of chemical reactions (see also [186]). The system has a state space dimension of $N_s = 3$ and a parameter space dimension of $N_p = 3$.

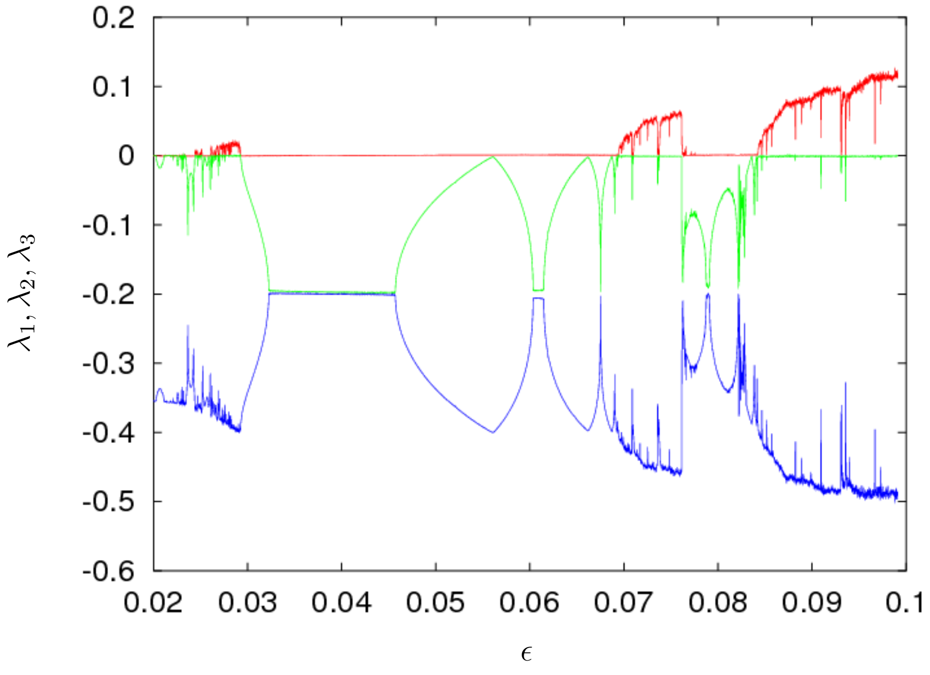


Figure 11.13: Aizawa system: Lyapunov spectrum.
Depending on whether the largest Lyapunov exponent (red) is zero or positive, it shows characteristic regions in the parameter interval $\epsilon \in [0.02, 0.1]$ where the dynamic behavior of the system is periodic or chaotic. Note the characteristic tongues prior to chaotic regimes which indicate regions in the parameter interval where period doubling scenarios are present.

Definition: ***Rössler system***

$$
\begin{aligned}
\frac{d}{dt}\underline{s}(t) = \underline{f}(\underline{s}(t), \underline{p})\,, \quad & \underline{s}(t) = (x(t), y(t), z(t))^T \in \mathbb{R}^3 \\
\underline{p} = (a, b, c) \in \mathbb{R}^3 & \qquad\qquad (11.5) \\
\underline{f}(x, y, z, \underline{p}) = & \begin{pmatrix} -(y+z) \\ x + ay \\ b + z(x-c) \end{pmatrix}
\end{aligned}
$$

■

In figure 11.14, the Lyapunov exponents of this dynamical system continuous in time are shown, whereby in figure 11.14(a) all three Lyapunov exponents were presented with the smallest Lyapunov exponent multiplied by the factor $\frac{1}{10}$ for reasons of a better representation. In figure 11.14(b), only at the two largest Lyapunov exponents are presented which leads to an even better observation of the characteristic structure. The system parameter c (see Eq. (11.5)) was varied in the interval $c \in [0.5, 40]$ using $N^s = 1024$ scan points.

11.6 The Lorenz 84 system

Like the first model presented by Lorenz in 1963 (see Sec. 3.4.1.1), this systems stems from the field of meteorology as well and was derived again by Lorenz in 1984 [98]. Like its famous predecessor it has a state space dimension of $N_s = 3$ and a parameter space dimension of $N_p = 4$.

Definition: ***Lorenz84 system***

$$
\begin{aligned}
\frac{d}{dt}\underline{s}(t) = \underline{f}(\underline{s}(t), \underline{p})\,, \quad & \underline{s}(t) = (x(t), y(t), z(t))^T \in \mathbb{R}^3 \\
\underline{p} = (a, b, F, G) \in \mathbb{R}^4 & \qquad\qquad (11.6) \\
\underline{f}(x, y, z, \underline{p}) = & \begin{pmatrix} a(F-x) - (y^2+z^2) \\ -y + G + (xy - bxz) \\ -z + (bxy + xz) \end{pmatrix}
\end{aligned}
$$

■

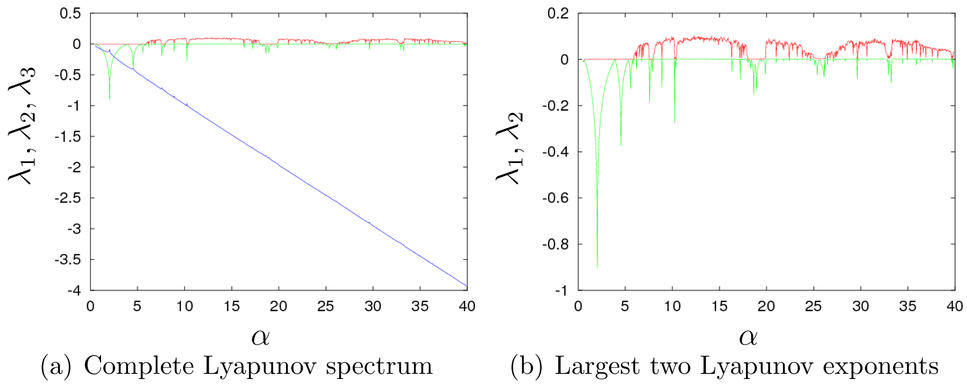
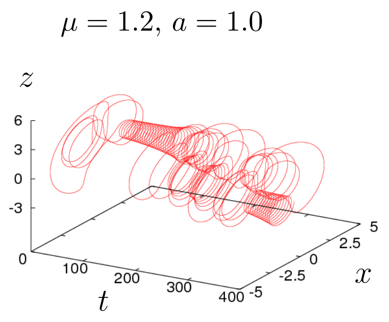


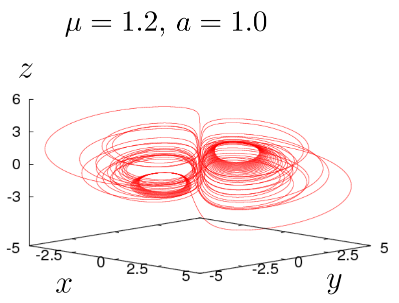
Figure 11.14: Rössler system: Lyapunov exponents.
In (a) the complete Lyapunov spectrum, that is all three Lyapunov exponents, of the Rössler system are shown. For reasons of a better representation the smallest exponent was multiplied by a factor of $\frac{1}{10}$. *In (b) only the two largest Lyapunov exponents are shown. Here one can identify in the left part, where the largest exponent is zero, a period doubling scenario and as usual periodic windows in the chaotic regime in the center and right part.*

11.7 The Rikitake system

This dynamical system of the ODE class is a model proposed by Rikitake [141], as a model for the self-generation of the geomagnetic field by large current carrying eddies in the core. The Rikitake system exhibits a dynamic behavior (see Fig. 11.15), which is very similar to that of the Lorenz system (see Fig. 3.10 in Sec. 3.4.1.1) and attempts to explain the irregular polarity switching of the Earth's magnetic field. The system describes the currents of two coupled dynamo disks and is defined by:



(a) chaotic attractor



(b) chaotic attractor

Figure 11.15: Rikitake system: chaotic attractor
In (a), a chaotic attractor of the Rikitake system in a three dimensional projection of the four dimensional extended state space is shown, whereas in (b) the same chaotic attractor is shown in the three dimensional state space. Note the geometric structure of the attractor, which is quite similar to the structure of the attractors of the Lorenz system.

Definition: ***Rikitake system***

$$\begin{aligned}
&\frac{d}{dt}\underline{s}(t) = \underline{f}(\underline{s}(t), \underline{p}), \quad \underline{s}(t) = (x(t), y(t), z(t))^T \in \mathbb{R}^3 \\
&\underline{p} = (\mu, a) \in \mathbb{R}^2 \qquad (11.7) \\
&\underline{f}(x, y, z, \underline{p}) = \begin{pmatrix} -\mu x + zy \\ -\mu y + (z-a)x \\ 1 - xy \end{pmatrix}
\end{aligned}$$

■

11.8 The "full" Brusselator system

This dynamical system of the ODE class is a kinetic model of a multi-molecular chemical reaction proposed in [71], as the three dimensional extension of the two-dimensional original Brusselator system (see also [169]). The original Brusselator (see for instance [82, 118], was derived as a simplified model of some reaction diffusion equations of the Belousov-Zhabotinsky type (see for instance [47]). The system is defined by:

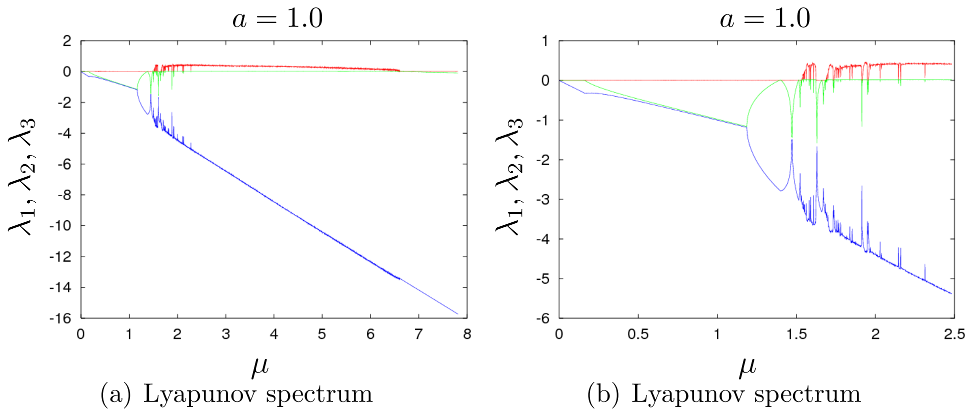


Figure 11.16: Rikitake system: Lyapunov exponents.

In (a), the Lyapunov spectrum of the Rikitake system is shown in the parameter interval $\mu \in [0, 8]$*. This one-dimensional system parameter scan was done using 8001 scan points. In (b), the interval* $\mu \in [0, 2.5]$ *is shown enlarged. Parameter setting:* $a = 1.0$

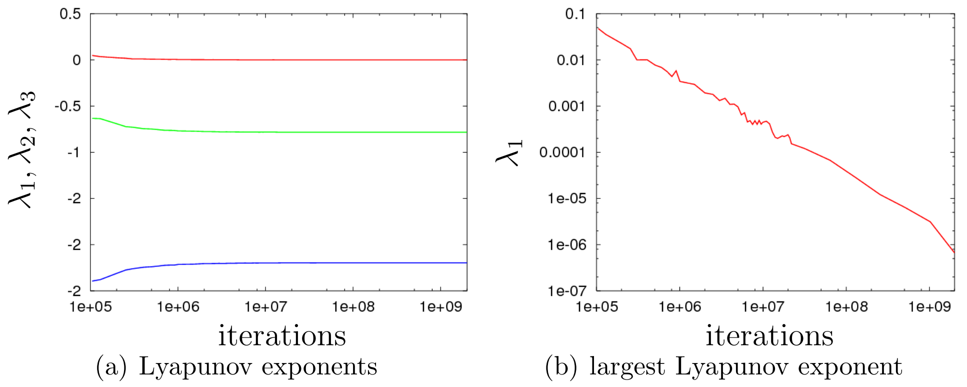


Figure 11.17: Rikitake system: Convergence of the Lyapunov exponents.

In (a), the convergence of the three Lyapunov exponents of the Rikitake system is shown, whereas in (b) the convergence of the largest Lyapunov exponent only is shown in a doubly logarithmic representation. As one can see, the convergence behavior is quite good. Even for a small number of iterations, the calculated exponents are relatively good approximated. The number of iterations was varied from 10^5 *up to* $2 \cdot 10^9$*. The used integration method was the classical Runge-Kutta scheme with a fixed step size of* 10^{-3}.

Definition: ***"Full" brusselator system***

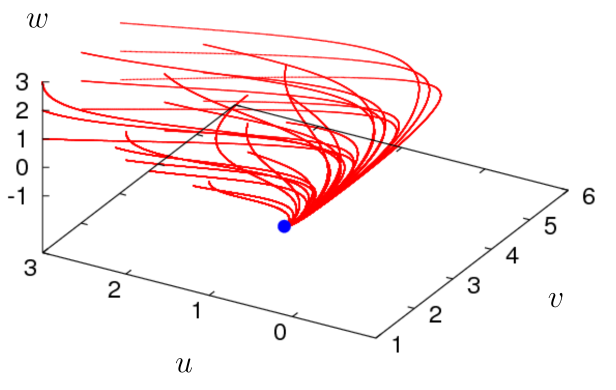
$$
\begin{aligned}
\frac{d}{dt}\underline{s}(t) = \underline{f}(\underline{s}(t), \underline{p})\,, \quad & \underline{s}(t) = (u(t), v(t), w(t))^T \in \mathbb{R}^3 \\
\underline{p} = (\alpha) \in \mathbb{R} & \\
\underline{f}(u, v, w, \underline{p}) = & \begin{pmatrix} 1 + u^2 v - (w+1)u \\ uw - u^2 v \\ \alpha - uw \end{pmatrix}
\end{aligned}
\tag{11.8}
$$

■

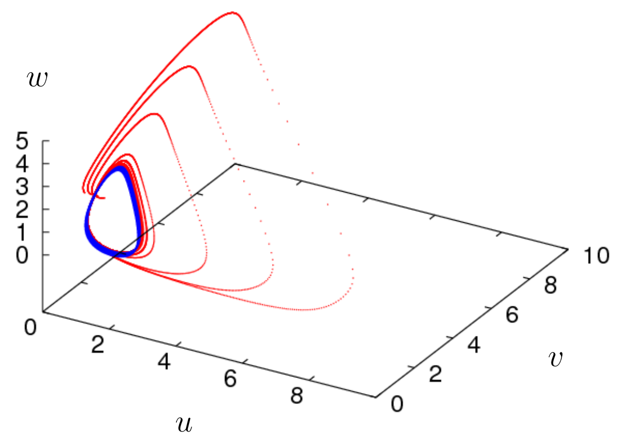
System (11.8) possesses a critical point $\underline{s}_c = (1, \alpha, \alpha)$ whereby the characteristic polynomial of the corresponding Jacobian

$$
\left.\frac{\partial \underline{f}}{\partial \underline{s}}\right|_{\underline{s}_c} = \begin{pmatrix} \alpha - 1 & 1 & -1 \\ -\alpha & -1 & 1 \\ -\alpha & 0 & -1 \end{pmatrix}
\tag{11.9}
$$

is $p(\lambda) = \lambda^3 + (3-\alpha)\lambda^2 + (3-2\alpha)\lambda + 1$. From this one can read off, that the stability criterion of the fixed point is satisfied for $\alpha < \frac{9-\sqrt{17}}{4}$. Hence, at the parameter value $\alpha = \frac{9-\sqrt{17}}{4} \approx 1.21922$, a supercritical Hopf bifurcation occurs and a stable limit cycle emerges. In figure 11.18(a), a three-dimensional initial states scan was performed for the parameter value $\alpha = 0.1$ and as expected, the trajectories of all initial states converge to the fixed point $\underline{s}_c = (1, \alpha, \alpha)$. In figure 11.18(b) a one-dimensional initial states scan was performed at the parameter value $\alpha = 1.35$. Hence the trajectories converge to the stable blue colored limit cycle.



(a) fixed point



(b) limit cycle

Figure 11.18: Brusselator system: dynamic behavior

In (a), a three-dimensional initial states scan was performed with $u(0), v(0), w(0) \in \{1, 2, 3\}^3$ *at the parameter value* $\alpha = 0.1$. *For all initial states, the trajectories converge to the stationary fixed point* $(1, \alpha, \alpha)$ *marked as a blue point. In (b) a one-dimensional initial states scan was performed with* $u(0) \in \{0.8, 1.0, 1.2, 1.4\}$ *and* $v(0) = 0.8$, $w(0) = 2.5$ *and the parameter value* $\alpha = 1.35$. *To illustrate the drastic different orbital velocities, in (b) the states - equidistant in time - are plotted in point style.*

11.9 An extended example investigation

This section is an example of an extended investigation using the **AnT 4.669** software package. It demonstrates the capabilities of the software and how they can be used for the simulation and investigation. The considered system is a one-dimensional piecewise-smooth dynamical system, representing a Poincaré return map for dynamical systems of the Lorenz type. This system shows a bifurcation scenario similar to the classical period doubling one, but which is influenced by so-called border collision phenomena and denoted as border collision period doubling bifurcation scenario. This scenario is formed by a sequence of pairs of bifurcations, whereby each pair consists of a border collision bifurcation and a pitchfork bifurcation. The mechanism leading to this scenario and its characteristic properties, like symmetry-breaking and symmetry-recovering as well as emergence of coexisting attractors, are investigated.

11.9.1 Introduction

Investigation of dynamical systems with a piecewise-smooth system function, motivated from a theoretical point of view as well as by practical applications, is a central topic of many scientific works published in the recent years. Especially several power electronic circuits (for instance DC/DC converters, such as boost, buck and buck-boost ones) lead to models, which belong to this class and show a rich bifurcation behavior [134, 148, 101, 104, 200, 31, 9, 33, 83]. Another application field of piecewise-smooth models are mechanical systems with impact or stick-slip phenomena [17, 92, 48, 12, 135, 78, 162, 10, 13, 45, 113]. In the field of nonlinear dynamics one-dimensional maps with a piecewise-smooth system function are well-known as return maps, obtained by the investigation of Poincaré sections of several dynamical systems continuous in time [70, 53]. Hereby the discontinuity of the return map is caused by the stretching, squeezing and folding mechanism which is inherent for chaotic attractors like for instance in systems of the Lorenz type [97, 151].
The behavior of piecewise-smooth dynamical systems is mainly influenced by phenomena occurring at the border between partitions in the state space. Early works in this field are presented by Feigin in the Russian publications [42, 43, 44]. In the Western literature the first works on bor-

der collision bifurcations are performed by Nusse, Yorke, Ott and Grebogi [122, 121, 17, 123, 39]. A lot of important results are discovered by Maistrenko [101, 104, 102, 103], di Bernardo and Budd [92, 27, 28, 30, 29, 89] as well as by other authors [119, 112, 48, 120]. Recently, several types of border collision related bifurcations are found, like corner collision, sliding and grazing bifurcations [32, 28]. An overview about bifurcations in piecewise-smooth dynamical systems and related phenomena is given in [203].

Here a one-dimensional map with a piecewise-smooth system function is considered. This map is closely related to a special kind of Poincaré return map of the Lorenz system [53, 137]. Some aspects of the dynamic behavior of this system concerning coexisting attractors [117] and the application of symbolic dynamics [201, 202] were reported until now. However the bifurcation scenarios occurring in systems like the presented one were not well investigated. It turns out, that dynamical systems like the one presented here show a sequence of bifurcations, where attractors with twice the period emerge, but these bifurcations are not the well-known flip bifurcations. Such bifurcations have been already observed experimentally [9], whereby a class of two-dimensional maps with a piecewise-smooth, but continuous system function are investigated. The system which is investigated does not belong to this class, because it is one-dimensional and possesses a system function with a discontinuity point. However, compared with systems investigated in [9], it has two remarkable advantages: Firstly it is one-dimensional and hence more easy to analyze. Secondly it shows a complete bifurcation scenario similar to the well-known period-doubling scenario, but dominated by the border collision phenomenon. This scenario, in the following denoted as border collision period doubling scenario, is the main topic here.

11.9.2 Border Collision Period Doubling Scenario

11.9.3 Investigated dynamical system

For the investigation of the border collision period doubling scenario the following dynamical system discrete in time is considered:

$$x_{n+1} = f(x_n, \alpha) = \begin{cases} f_l(x_n, \alpha) = \alpha x_n (1 - x_n) & \text{if } \ x_n < \frac{1}{2} \\ f_c(x_n) = \frac{1}{2} & \text{if } \ x_n = \frac{1}{2} \\ f_r(x_n, \alpha) = \alpha x_n (x_n - 1) + 1 & \text{if } \ x_n > \frac{1}{2} \end{cases} \tag{11.10}$$

with $x \in [0, 1]$ and the parameter $\alpha \in [0, 4]$. The following properties of system (11.10) are important:

1. For all parameter values except $\alpha = 2$, the system function f is discontinuous at the point $x = \frac{1}{2}$ (see Fig. 11.19). Therefore it can be expected, that the dynamical behavior of system (11.10) is influenced by the border collision phenomena, which are typical for piecewise-smooth dynamical systems.

2. On the interval $x \in \left[0, \frac{1}{2}\right)$ the system function f is identical with the system function of the logistic map

$$x_{n+1} = \alpha x_n(1 - x_n) \quad \text{with} \quad x \in [0, 1], \quad \alpha \in [0, 4] \tag{11.11}$$

   Hence, one can expect, that some aspects of the dynamic behavior of the logistic map (11.11) are preserved in the case of system (11.10). Especially the question arises, whether system (11.10) shows a behavior, analogous to the well investigated period doubling bifurcation scenario of the logistic map.

3. In contrast to the logistic map, the system function f is symmetrical with respect to its discontinuity point $x = \frac{1}{2}$, namely

$$f(x, \alpha) = 1 - f(1 - x, \alpha) \tag{11.12}$$

   Therefore, the dynamic behavior of system (11.10) must be influenced by symmetry breaking - symmetry recovering phenomena, leading to asymptotic dynamics taking place either on symmetric attractors or pairs of coexisting attractors symmetric to each other.

In the literature ([53, 137, 201, 202]), discontinuous maps on an interval with a single point of discontinuity (i.e. dynamical systems like (11.10)) are considered only in non-symmetric variants. Usually the discontinuity point is assumed to belong either to the left or to the right half-interval. However the symmetric variant with a special treatment of the discontinuity point seems to be more plausible, if the investigated map is considered as a Poincaré return map of a dynamical systems continuous in time. In [53] it is shown, that system (11.10) represents a Poincaré return map of the Lorenz system, whereby the point of discontinuity of system (11.10) corresponds to the stable manifold of the fixed point in the origin. Therefore this point must be treated specifically.

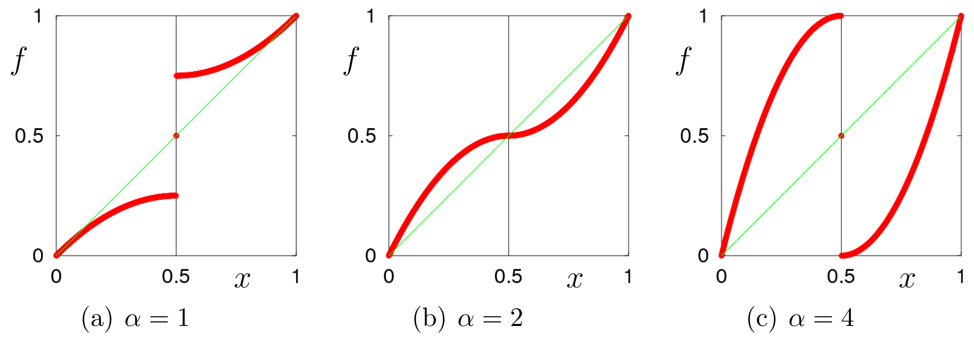


Figure 11.19: Typical shapes of the system function $f(x, \alpha)$

11.9.4 Description of the bifurcation scenario

By variation of the parameter α, system (11.10) shows a bifurcation scenario, which one can denote as **border collision period doubling** scenario. As one can see from Fig. 11.20(a), the period diagram of this scenario can not be distinguished from the one of the period doubling scenario taking place in the logistic map. One observes here also a sequence of periodic attractors, whereby the subsequent periods represent a geometrical series $p_n = p_0 2^n$, with $n = 0, 1, 2, \ldots, \infty$ and $p_0 = 1$. The diagram of the Lyapunov exponent (Fig. 11.20(b)) shows also the well-known behavior with $\lambda = 0$ at the local bifurcation points and $\lambda \to -\infty$ at super-stable points, which lie between each two subsequent local bifurcations. However the bifurcation diagram (Fig. 11.21) is totally different from the classical period doubling scenario. The bifurcations observed here are clearly not the usual flip bifurcations. Hence, the important question we have to deal with, is, how the bifurcation scenario emerges here.

11.9.5 Fixed points and periodic orbits of the investigated system

Let us consider the behavior of system (11.10) in the complete interval $\alpha \in [0, 4]$. Firstly one can see, that for all parameter values the system possesses three fixed points $x_1^* = 0$, $x_2^* = \frac{1}{2}$ and $x_3^* = 1$. Using the linear stability analysis, one finds, that in the parameter interval $0 \leq \alpha < 1$ the fixed points x_1^* and x_3^* are stable. The basins of attraction of these fixed points for $\alpha \in [0, 1)$ are shown in Fig. 11.22(a). As one can see, all initial values from $\left[0, \frac{1}{2}\right)$ tend to the fixed point x_1^*, whereas all initial values from $\left(\frac{1}{2}, 1\right]$

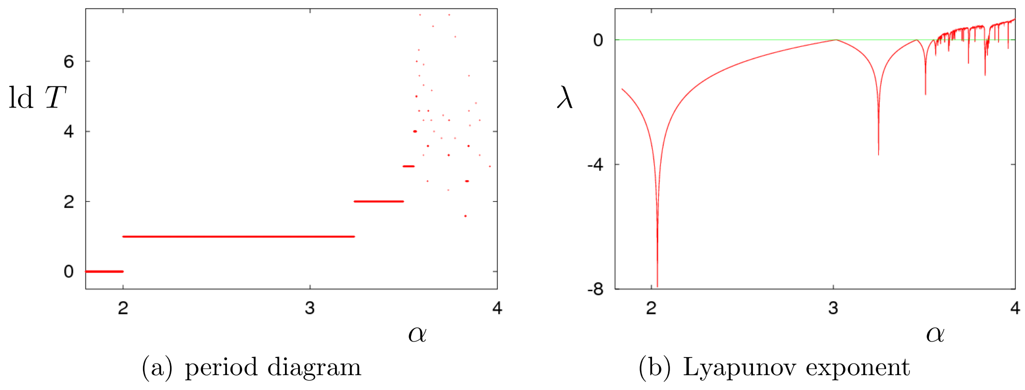


Figure 11.20: Border collision period doubling scenario: periods and Lyapunov exponents

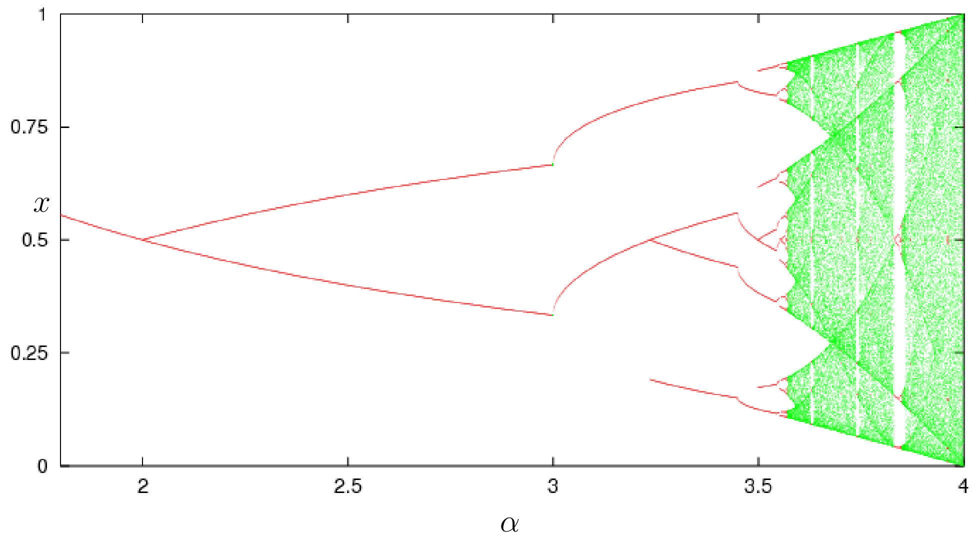
Shown are the periods T (logarithmic plot) and the Lyapunov exponents λ. Note, that these diagrams are identical with the corresponding diagrams of the period doubling scenario in the case of the logistic map, although the underlying mechanisms are totally different.

are mapped to the fixed point x_3^*.

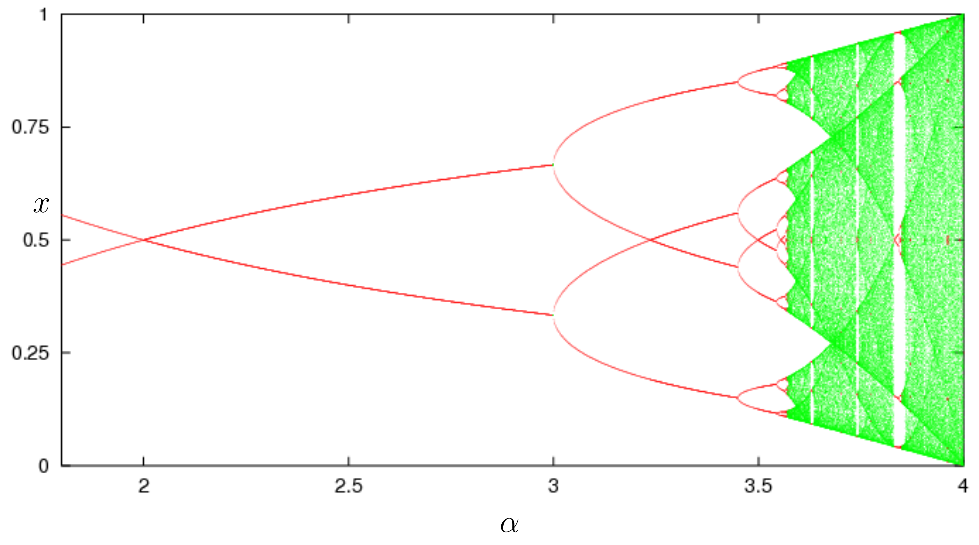
The stability of the fixed point x_2^* can not be determined using linear stability analysis, because the derivative of the system function f is not defined at this point. However, the fixed point x_2^* is unstable for all parameter values $\alpha \in [0, 2)$. This can be shown taking into account, that orbits with initial values $x_0 = x_2^* \pm \varepsilon$ for any arbitrary small deviation ε converge for $n \to \infty$ either to the fixed point x_1^* or to the fixed point x_3^*.

Both fixed points x_1^* and x_3^* become unstable by a **transcritical bifurcation**, which occurs at the parameter value $\alpha = \alpha^t = 1$ (see Fig. 11.23). At this point two new stable fixed points $x_4^* = 1 - \frac{1}{\alpha}$ and $x_5^* = \frac{1}{\alpha}$ emerge in the domain $[0, 1]$. The basins of attraction of the fixed points x_4^* and x_5^* for $\alpha \in (1, 2)$ are also shown in Fig. 11.22(a). As one can see, the initial values from $\left(0, \frac{1}{2}\right)$ tend to x_5^* and all initial values from $\left(\frac{1}{2}, 1\right)$ are finally mapped to x_4^*. Note that in Fig. 11.21 for reasons of simplicity and clarity only one fixed point, namely x_4^* is shown.

For parameter values α between 1 and 2 the fixed points x_1^*, x_2^* and x_3^* are unstable and the fixed points x_4^*, x_5^* are stable. At the parameter value $\alpha = \alpha_1^{bc} = 2$ the **first border collision bifurcation** (see Fig. 11.24



(a)



(b)

Figure 11.21: Border collision period doubling scenario: bifurcation diagram

The bifurcation diagram of the border collision period doubling scenario shows remarkable differences compared with that of the classical period doubling scenario of the logistic map. The diagrams are calculated for a single initial value (a) and for two symmetrical initial values (b).

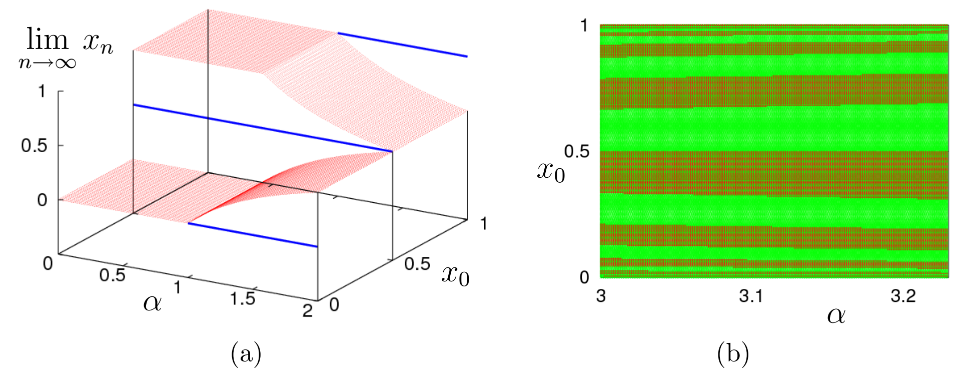


Figure 11.22: Asymptotic dynamics and basins of attraction

(a) Asymptotic dynamics of system (11.10) in dependence on the initial values for $\alpha \in [0, 2]$.

(b) Basins of attraction for the limit cycles $\{x_1^{**}, x_2^{**}\}$ (red) and $\{x_3^{**}, x_4^{**}\}$ (green).

and Fig. 11.23) occurs. Hereby two facts are important. Firstly, the fixed points x_4^* and x_5^* vanish at the bifurcation point. Note, that due to the border collision these fixed points do not lose their stability, as it is typical for local bifurcations, but disappear at all. Secondly, a stable limit cycle with period two emerges. This limit cycle consists of the points $1 - \frac{1}{\alpha}$ and $\frac{1}{\alpha}$, which were fixed points before the border collision bifurcation (for this reason we denote this limit cycle $\{x_4^*, x_5^*\}$). This behavior can be explained taking Fig. 11.24 into consideration. As one can see from this figure, before the bifurcation the functions f_l and f_r intersect the angles bisector in their domains $\left[0, \frac{1}{2}\right)$ and $\left(\frac{1}{2}, 1\right]$. Hence, these intersection points are fixed points of system (11.10). After the border collision bifurcation the intersection points leave the domains where the functions f_l and f_r have effect, but the second iterated function intersects now the angles bisector at the same points.

In addition one can remark, that the fixed points x_4^* and x_5^* collide at the bifurcation point not only with each other, but also with the fixed point x_2^*. Hence, the fixed point x_2^*, which is unstable before the bifurcation, is stable at the bifurcation point itself and after the bifurcation the fixed point x_2^* becomes unstable again. The described behavior is not essential for the border collision bifurcation taking place at $\alpha = 2$. The border collision bifurcation occurs as a result of the collision of the fixed points x_4^* and x_5^* with the border between the partitions and not due to their collision with

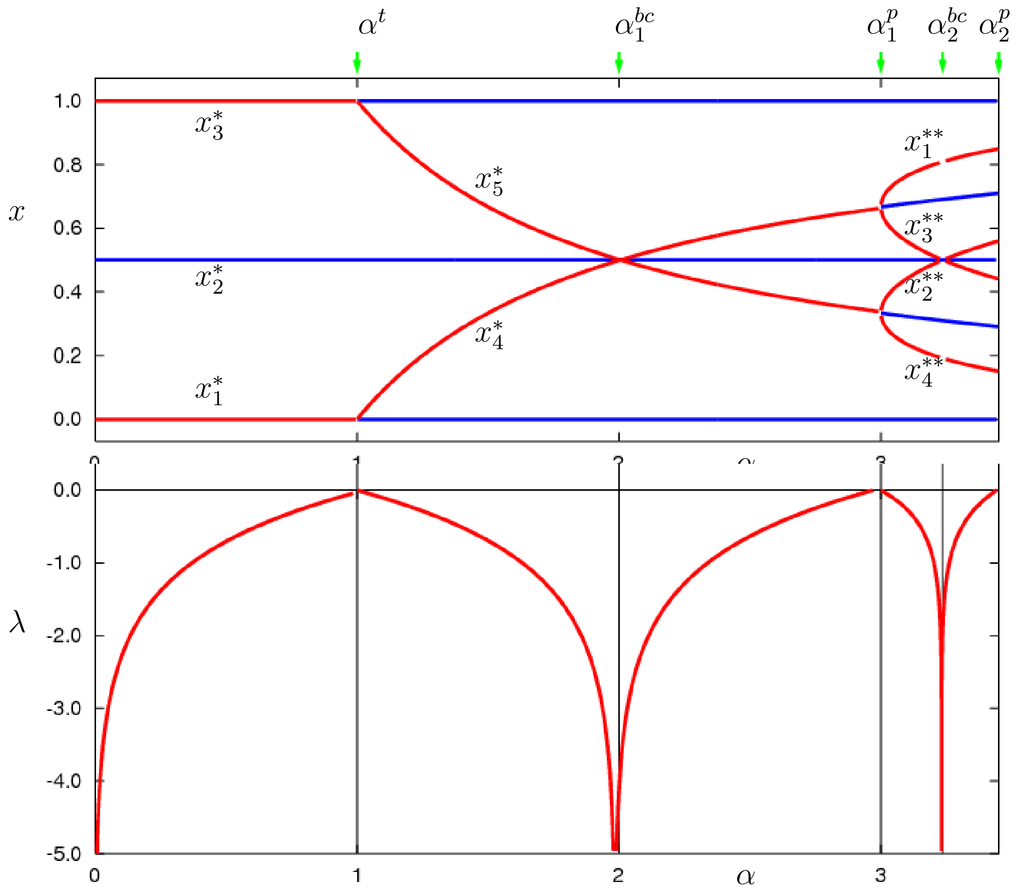


Figure 11.23: Analytic results

Analytical results about the attractors of system (11.10) and their Lyapunov exponents. The following bifurcation points are marked: α^t - transcritical bifurcation, α_1^{bc} - first border collision bifurcation, α_1^p - first pitchfork bifurcation, α_2^{bc} - second border collision bifurcation, α_2^p - second pitchfork bifurcation. The points x_1^*, x_2^* and x_3^* are the fixed points. The points x_4^*, x_5^* are fixed points between α^t and α_1^{bc} and build a limit cycle with period two after α_1^{bc}. The points x_1^{**}, x_2^{**}, x_3^{**}, x_4^{**} build two coexisting limit cycles with period two between α_1^p and α_2^{bc} and a limit cycle with period four after α_2^{bc}. In the upper part the stable solutions are shown as red lines, whereas the blue lines correspond to the unstable solutions.

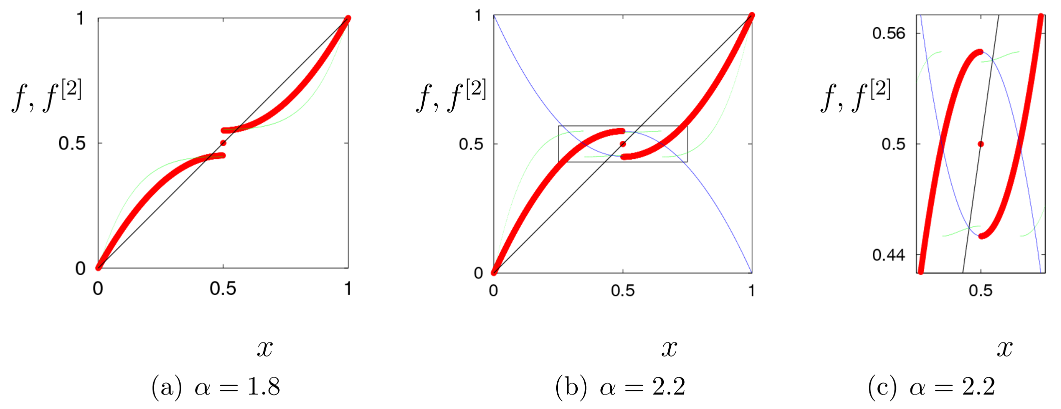


Figure 11.24: First border collision bifurcation

First border collision at $\alpha = \alpha_1^{bc} = 2$. Shown are the system function (red) and its second iterated function (green) before the bifurcation (a) and after the bifurcation (b). Fig. (c) is a blow-up of the rectangle marked in Fig. (b). The blue lines in Figs. (b,c) mark the functions f_l and f_r outside their domains.

the fixed point x_2^*, which in the considered case lies on this border.
For parameter values $2 < \alpha < 3$ the limit cycle $\{x_4^*, x_5^*\}$ is the global symmetric attractor of system (11.10). The fixed points x_1^*, x_2^* and x_3^* are unstable and the fixed points x_4^*, x_5^* do not exist after the first border collision bifurcation. At the parameter value $\alpha = \alpha_1^p = 3$ this limit cycle undergoes the **first pitchfork bifurcation** (see Fig. 11.23). Therefore, it loses its stability and two coexisting stable limit cycles with period two emerge. These limit cycles are given by

$$\{x_1^{**}, x_2^{**}\} = \frac{1}{2} + \frac{1}{2\alpha}\left(\pm 1 + \sqrt{a^2 - 2a - 3}\right) \tag{11.13}$$

and

$$\{x_3^{**}, x_4^{**}\} = \frac{1}{2} + \frac{1}{2\alpha}\left(\pm 1 - \sqrt{a^2 - 2a - 3}\right) \tag{11.14}$$

Note, that the period of the asymptotical dynamics does not change at the pitchfork bifurcation. The basins of attraction for these two limit cycles are shown in Fig. 11.22(b). Note further, that in Fig. 11.21 the limit cycle $\{x_3^{**}, x_4^{**}\}$ is not presented. As expected, the two limit cycles are symmetric to each other with respect to the point $x = \frac{1}{2}$, namely $x_1^{**} = \frac{1}{2} - x_4^{**}$, $x_2^{**} = \frac{1}{2} - x_3^{**}$.

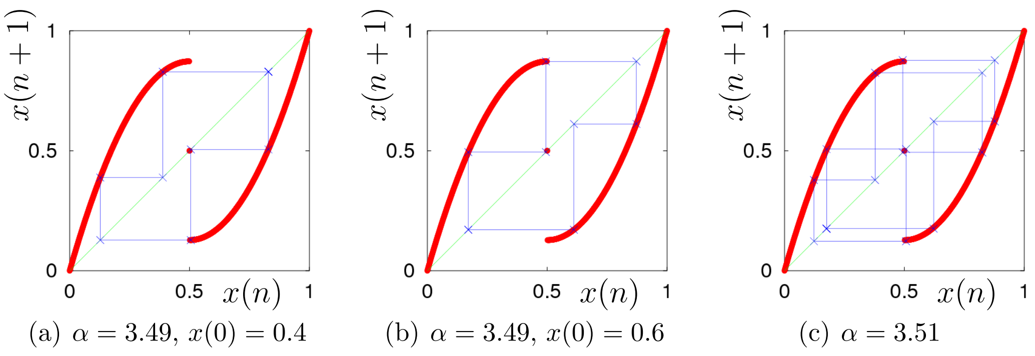


Figure 11.25: Third border collision bifurcation

Third border collision bifurcation at $\alpha = \alpha_3^{bc} \approx 3.4985$. Two coexisting asymmetric limit cycles with period $T = 4$ before the bifurcation (a), (b). A symmetric limit cycle with period $T = 8$ after the bifurcation (c).

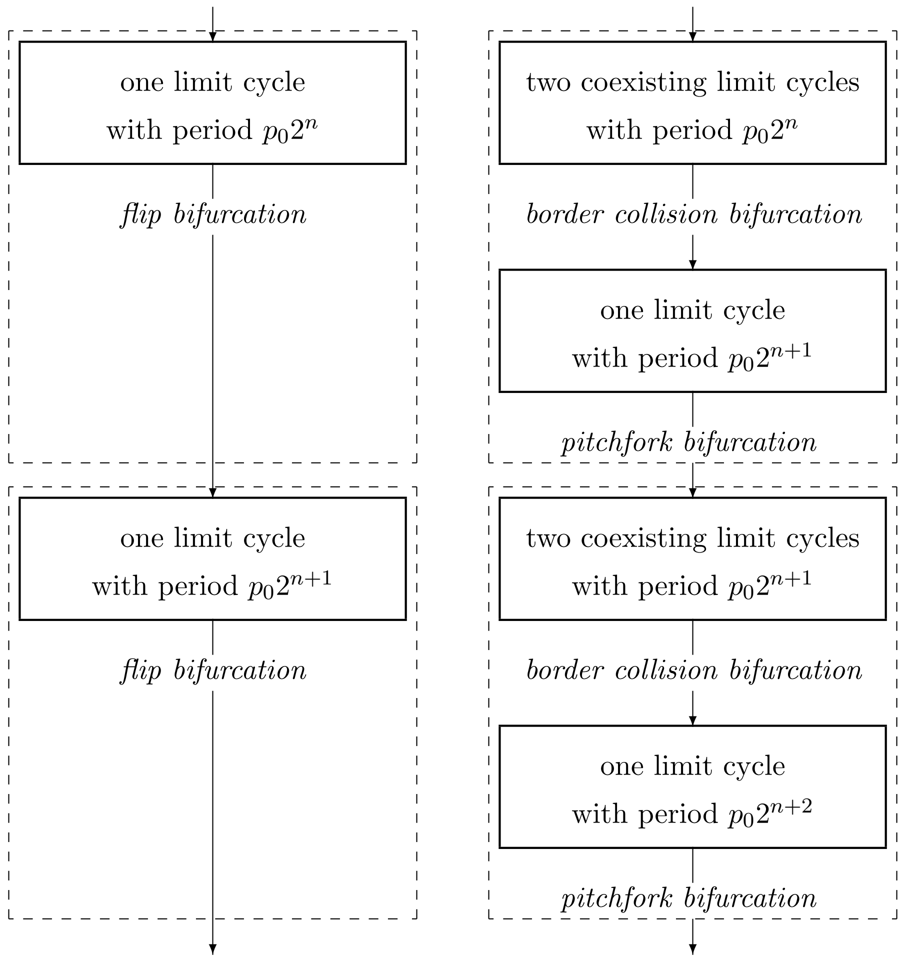
The **second border collision bifurcation** occurs at the parameter value $\alpha = \alpha_2^{bc} = 1 + \sqrt{5} \approx 3.2361$ (see Fig. 11.23). Here the two coexisting limit cycles $\{x_1^{**}, x_2^{**}\}$ and $\{x_3^{**}, x_4^{**}\}$ undergo the same scenario as the two coexisting fixed points at the first border collision bifurcation. That means, they do not exist any more after the bifurcation, and a stable limit cycle with period four emerges. Again, the new limit cycle after the border collision has twice the period as the coexisting limit cycles before. It consists of four points, which form the two coexisting limit cycles before the border collision. Accordingly, this new limit cycle is denoted as $\{x_1^{**}, x_2^{**}, x_3^{**}, x_4^{**}\}$.

The limit cycle $\{x_1^{**}, x_2^{**}, x_3^{**}, x_4^{**}\}$ represents the symmetric global attractor until the **second pitchfork bifurcation** takes place at $\alpha = \alpha_2^p = 1 + \sqrt{6} \approx 3.4495$ (see Fig. 11.23). There it loses it's stability and two new limit cycles with the same period emerge. These limit cycles coexist until the next border collision bifurcation, and the scenario continues with the same pattern (see Fig. 11.25). Note, that the described behavior is not specific for the parameter value α_2^{bc} of the second border collision bifurcation, but takes place at all following border collision bifurcations α_n^{bc} $(n > 2)$ as well.

Using the results presented above, one is able to calculate the Lyapunov exponent for all attractors existing in the parameter range $\alpha \in [0, \alpha_2^p]$. Because the natural measure $\rho(x)$ of fixed points as well as of limit cycles is concentrated on their points, and due to the well-known relation for maps on an interval $\lambda = \int_{\mathcal{A}} \rho(x) \ln |f'(x)| dx$, where $\mathcal{A}$ denotes the corresponding attractor, one obtains:

$$\lambda = \begin{cases} \lambda_1 & = & \ln(\alpha) & \text{if} \quad \alpha \leq \alpha^t \\ \lambda_2 & = & \ln|2-\alpha| & \text{if} \quad \alpha^t < \alpha \leq \alpha_1^p \\ \lambda_3 & = & \ln\sqrt{|\alpha^2 - 2\alpha - 4|} & \text{if} \quad \alpha_1^p < \alpha \leq \alpha_2^p \end{cases} \tag{11.15}$$

This means, that the value λ_1 holds for both fixed points x_1^* and x_3^*; the value λ_2 holds for fixed points x_4^* and x_5^* and for the limit cycle $\{x_4^*, x_5^*\}$. The value λ_3 holds for both limit cycles $\{x_1^{**}, x_2^{**}\}$, $\{x_3^{**}, x_4^{**}\}$ and for the limit cycle $\{x_1^{**}, x_2^{**}, x_3^{**}, x_4^{**}\}$.
As one can see from Fig. 11.20, the border collision bifurcations takes place at the super-stable points, where $\lambda \to -\infty$ holds. However, that this is not a general property of border collision bifurcations, but a specific feature of the system (11.10). This property is here due to the fact, that the derivatives of both functions $f_l(x)$ and $f_r(x)$ are equal to zero at the point of discontinuity of the function $f(x)$, i.e. at the point $x = \frac{1}{2}$.
Summarizing the results obtained so far and compare the border collision period doubling scenario described here with the usual period doubling scenario. In both cases there exists a sequence of periodic attractors with periods $p_0 2^n$, $n \geq 0$. In the case of the usual period doubling scenario the sequence can be illustrated with the diagram shown in Fig. 11.26.(a). In contrast to this, the border collision period doubling scenario is formed by a sequence of pairs of bifurcations. Each of them consists of two bifurcations, a border collision bifurcation and a pitchfork bifurcation, as it is schematically shown in Fig. 11.26.(b).
Both scenarios converge to the same parameter value α_∞, where an attractor of the Feigenbaum type (a strange, but not chaotic one) exists. Note, that the scaling properties of the border collision period doubling scenario of system (11.10) are the same as the scaling properties of the classical period doubling scenario of the logistic map (11.11). Indeed, the border collision bifurcations occur in system (11.10) at the same parameter values, where the logistic map has the super-stable orbits. The pitchfork bifurcations in system (11.10) take place at the same parameter values, where the logistic map has the flip bifurcations. Hence, the Feigenbaum constant corresponding to the scaling behavior in the parameter space of the border collision period doubling scenario in system (11.10) have to be the same as in the case of the logistic map. Furthermore, also the Feigenbaum constant corresponding to the scaling behavior in the state space have to be the same for both systems. This is due to the fact that both parabola, that of each pitchfork bifurcation


one limit cycle
with period $p_0 2^n$
flip bifurcation
one limit cycle
with period $p_0 2^{n+1}$
flip bifurcation
two coexisting limit cycles
with period $p_0 2^n$
border collision bifurcation
one limit cycle
with period $p_0 2^{n+1}$
pitchfork bifurcation
two coexisting limit cycles
with period $p_0 2^{n+1}$
border collision bifurcation
one limit cycle
with period $p_0 2^{n+2}$
pitchfork bifurcation

(a) (b)

Figure 11.26: Ordinary and border collision period doubling scenario
Schematic representation of the ordinary or classical period doubling scenario (a) and the border collision period doubling scenario (b). The dashed boxes mark the regions, which can be denoted as one step of the corresponding scenario.

in the border collision period doubling scenario of system (11.10) and that of the corresponding flip bifurcation of the classical period doubling scenario of the logistic map (11.11) are identical.
Concerning the symmetry breaking - symmetry recovering property of the border collision period doubling scenario, mentioned in section 11.9.3, one yields now the following: in each step of the scenario the symmetry breaking takes place at the pitchfork bifurcation, where a symmetric limit cycle becomes unstable and splits into two coexisting asymmetric limit cycles with the same period, which are symmetric to each other. Note, that Fig. 11.21, Fig. 11.27 and Fig. 11.28 show only one of the coexisting limit cycles. The symmetry is recovered by the next border collision bifurcation, whereby the asymmetric limit cycles disappear, and a new symmetric one with twice the period emerges (see Fig. 11.25). This behavior is illustrated in Fig. 11.29, which shows the mean point $\bar{x}$ of the attractors, defined by

$$\bar{x}(\mathcal{A}) = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \mathcal{A} = \{x_1 \dots x_N\} \tag{11.16}$$

depending on the parameter α. For a symmetric attractor $\mathcal{A}$ of system (11.10) it holds $\bar{x}(\mathcal{A}) = \frac{1}{2}$, whereas for two asymmetric attractors $\mathcal{A}_1$ and $\mathcal{A}_2$, symmetric to each other, it holds $\bar{x}(\mathcal{A}_1) = 1 - \bar{x}(\mathcal{A}_2)$.

11.9.6 Kneading orbits and locating of band merging bifurcations

A lot of interesting results for the investigated system can be obtained using the technique of kneading orbits [84, 110, 19]. The usual approach here is to investigate itineraries of some critical points. For system (11.10) this is obviously the point $x = \frac{1}{2}$. Due to the fact, that the point $x = \frac{1}{2}$ is a fixed point of system (11.10), one has to track itineraries of points in its vicinity. Therefore we introduce the following functions:

$$g_l(\alpha) = \lim_{\varepsilon \to 0} f\left(\frac{1}{2} - \varepsilon, \alpha\right) = f_l\left(\frac{1}{2}, \alpha\right) \tag{11.17}$$

$$g_r(\alpha) = \lim_{\varepsilon \to 0} f\left(\frac{1}{2} + \varepsilon, \alpha\right) = f_r\left(\frac{1}{2}, \alpha\right) \tag{11.18}$$

Higher itineraries of critical points $\frac{1}{2} - \varepsilon$ and $\frac{1}{2} + \varepsilon$ ($\varepsilon \to 0$) can be calculated iteratively. For a sequence s consisting of alternating symbols l and r the

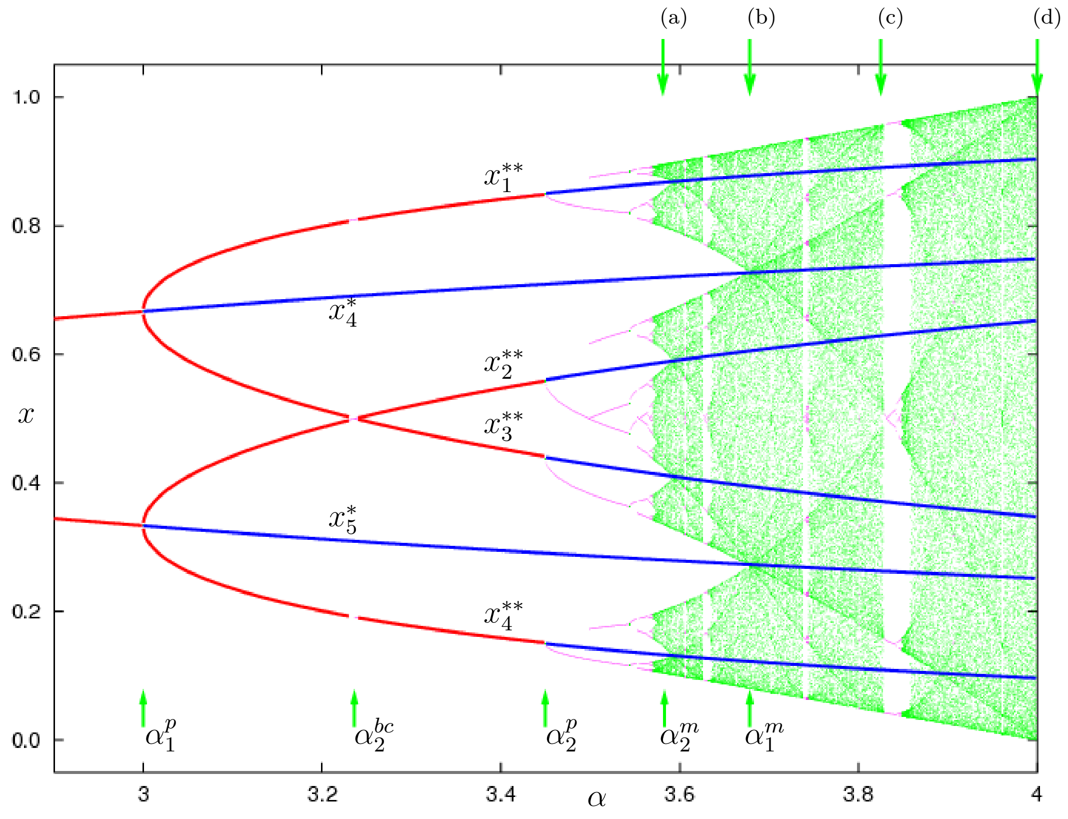


Figure 11.27: Band merging cascade

Band merging cascade in system (11.10). Parameter values corresponding to Fig. 11.31 are marked with (a), (b), (c) and (d). See text for a detailed description.

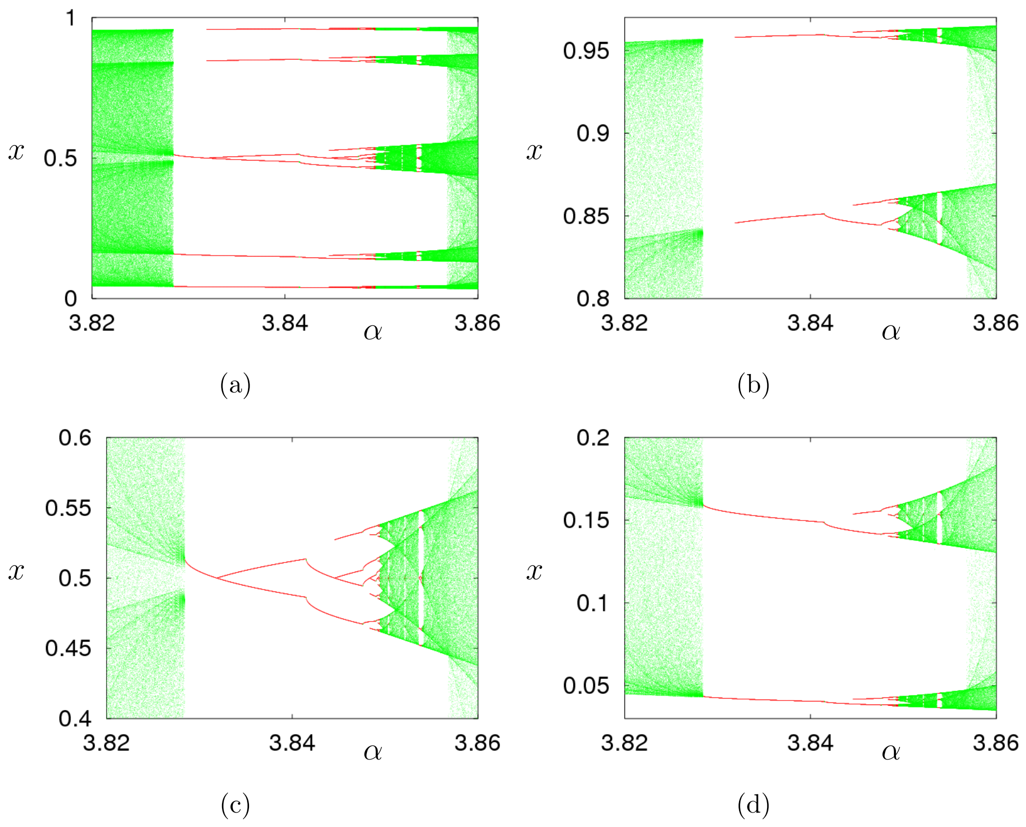


Figure 11.28: Border collision period doubling scenario in a 3-periodic window

Border collision period doubling scenario within the 3-periodic window for a single initial value (a). Enlarged are the upper part (b), the middle part (c), and the lower part (d) of the scenario.

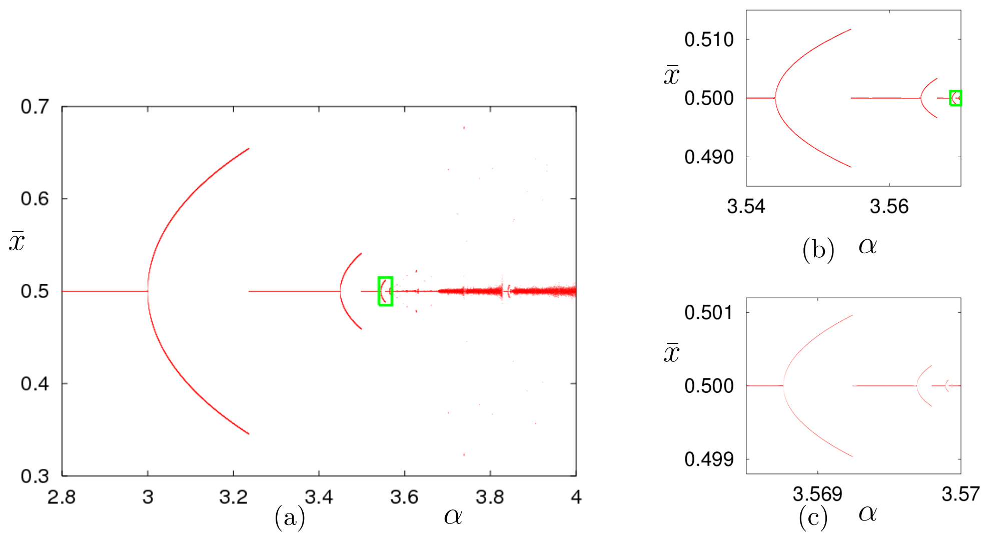


Figure 11.29: Mean points of attractors

Mean points $\bar{x}$ of the attractors within the border collision period doubling scenario, obtained for two symmetric initial values ($x(0) = 0.25$ and $x(0) = 0.75$). The green rectangles in (a) and (b) are shown enlarged in Figs. (b) and (c).

functions

$$\begin{aligned} g_{slr}(\alpha) &= f_r(g_{sl}(\alpha), \alpha) \\ g_{srl}(\alpha) &= f_l(g_{sr}(\alpha), \alpha) \end{aligned} \tag{11.19}$$

are defined, which lead to the higher itineraries of critical points mentioned above. The functions $g_{\ldots}(\alpha)$ represent polynomials and can be calculated analytically. One can calculate for instance:

$$g_l(\alpha) = \frac{\alpha}{4} = 1 - g_r(\alpha) \tag{11.20}$$

$$g_{lr}(\alpha) = \frac{1}{16}\alpha^3 - \frac{1}{4}\alpha^2 + 1 = 1 - g_{rl}(\alpha) \tag{11.21}$$

$$\begin{aligned} g_{lrl}(\alpha) &= \frac{1}{256}\alpha^7 - \frac{1}{32}\alpha^6 + \frac{1}{16}\alpha^5 + \frac{1}{16}\alpha^4 - \frac{1}{4}\alpha^3 + 1 \\ &= 1 - g_{rlr}(\alpha) \end{aligned} \tag{11.22}$$

It turns out, that the functions $g_{\ldots}(\alpha)$ are useful for several purposes, for instance for locating of the border collision bifurcations. These bifurcations occur at the parameter values, where the itineraries of critical points reach these points again. Therefore, one can in principle calculate these parameter values solving the corresponding equations $g_{\dots}(\alpha) = \frac{1}{2}$. Especially when dealing with the border collision period doubling between the parameter values α^t and α_∞ one has to consider the equations

$$g_{(lr)^{2^k}}(\alpha) = g_{\underbrace{lr\dots lr}_{2^k \text{times}}}(\alpha) = \frac{1}{2}, \qquad k = 1, 2, 3, \dots \tag{11.23}$$

For instance, the value α_2^{bc} can be found from the equation $g_{lr}(\alpha) = \frac{1}{2}$, the next value α_3^{bc} - from the equation $g_{lrlr}(\alpha) = \frac{1}{2}$, and so on (see Fig. 11.30). Note, that because of the symmetry of the investigated system the equations

$$g_{(lr)^{2^k}}(\alpha) = \frac{1}{2} \quad \text{and} \quad g_{(rl)^{2^k}}(\alpha) = \frac{1}{2} \tag{11.24}$$

have the same solutions. Due to the increasing degrees of the polynomials the equations for α_k^{bc}, $k \geq 3$ can be solved only numerically. In particular one obtains:

$$\alpha_1^{bc} = 2 \tag{11.25}$$
$$\alpha_2^{bc} = 1 + \sqrt{5} \tag{11.26}$$
$$\alpha_3^{bc} \approx 3.498561699327 \tag{11.27}$$
$$\alpha_4^{bc} \approx 3.554640862769 \tag{11.28}$$

We remark additionally, that using the described procedure one can calculate not only the parameter values of the border collision bifurcations between α^t and α_∞, but also the points of these bifurcations beyond α_∞.

11.9.7 Behavior of the investigated system beyond α_∞

Until now the behavior of system (11.10) within the (main) border collision period doubling scenario was described, i.e. for parameter values $0 < \alpha < \alpha_\infty$. For $\alpha > \alpha_\infty$ system (11.10) show also a lot of interesting phenomena, like the band merging scenario, an infinite number of periodic windows within the chaotic regime, and the border collision period doubling scenario within these windows. Again we observe some similarities as well as some differences with the behavior of the logistic map. These topics are briefly described in this section.

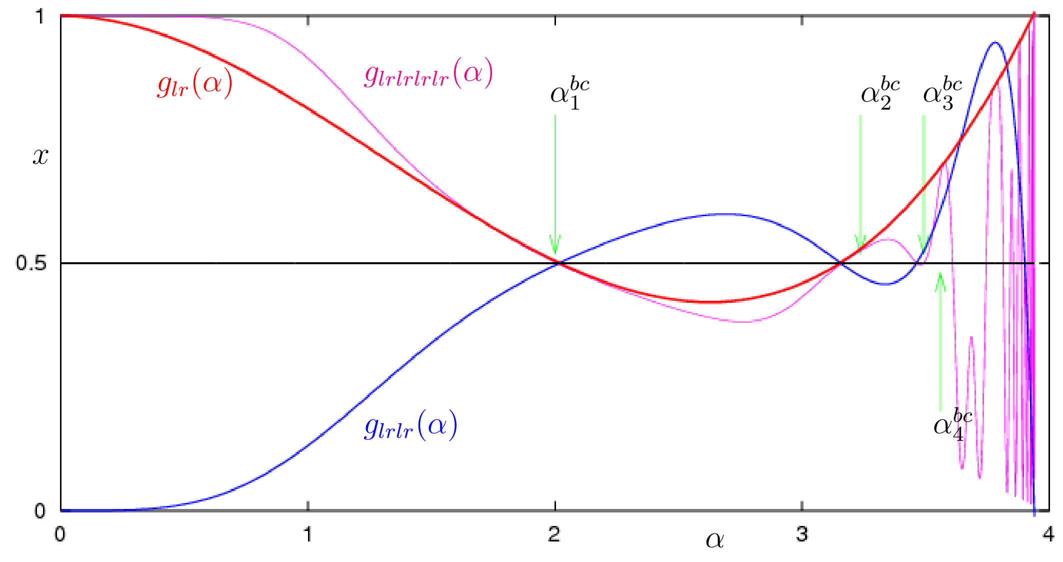


Figure 11.30: Locating of border collision bifurcations

Locating border collision bifurcations in system (11.10) using the itineraries of the critical point $x = \frac{1}{2}$.

11.9.8 Influence of the border collision period doubling scenario on the band merging scenario

Like the period doubling scenario, also the band merging bifurcation cascade is well-known for the logistic map. After the parameter value α_∞ the logistic map shows an infinite sequence of bifurcations, whereby the number of bands of the multi-band attractors is divided by two at each bifurcation point α_n^m. Therefore, the n-th bifurcation in this sequence can be described as follows. Before the bifurcation point a chaotic attractor with 2^n bands exists. At the bifurcation point the bands of this attractor merge pairwise with each other. Additionally the merging points collide with the points of an unstable limit cycle with period 2^{n-1}, which emerges at the n-th flip bifurcation and becomes unstable at the $(n+1)$-th one. After the bifurcation point a chaotic attractor with 2^{n-1} bands exists. Obviously, all multi-band attractors of the logistic map have an even number of bands.
System (11.10) shows also a band merging bifurcation cascade, which is similar to the one described above. However the symmetry of system (11.10) leads to remarkable difference between the dynamics of this system and the one of the logistic map. In the case of system (11.10) there exists also an infinite sequence of bifurcations (see Fig. 11.27), where the bands of the multi-band attractors merge pairwise, but these attractors always have an odd number of bands. More precisely, before the n-th bifurcation in the band merging cascade a chaotic attractor with $2^{n+1} - 1$ bands exists. At the bifurcation point its bands merge pairwise with each other and additionally collide with the points of an unstable limit cycle with the period 2^n. As described in the previous section, this limit cycle emerges within the border collision bifurcation scenario at the point of the n-th border collision bifurcation α_n^{bc} and becomes unstable at the point of the n-th pitchfork bifurcation α_n^p. After the bifurcation point a chaotic attractor with $2^n - 1$ bands exists.
In Fig. 11.27 the described dynamics is illustrated in more detail for the first two (or the last two, depending on the preferred direction in parameter space) band merging bifurcations. Before the band merging bifurcations at the parameter value α_2^m there exists a chaotic seven-band-attractor (see Fig. 11.31.(a)). At the bifurcations point its bands merge and collide with the points of the limit cycle $\{x_1^{**}, x_2^{**}, x_3^{**}, x_4^{**}\}$, which emerges at the second border collision bifurcation at α_2^{bc} and becomes unstable at the second pitchfork bifurcation α_2^p. After the band merging bifurcations a chaotic three-band-attractor exists. Its bands collide at the parameter value α_1^m with each other

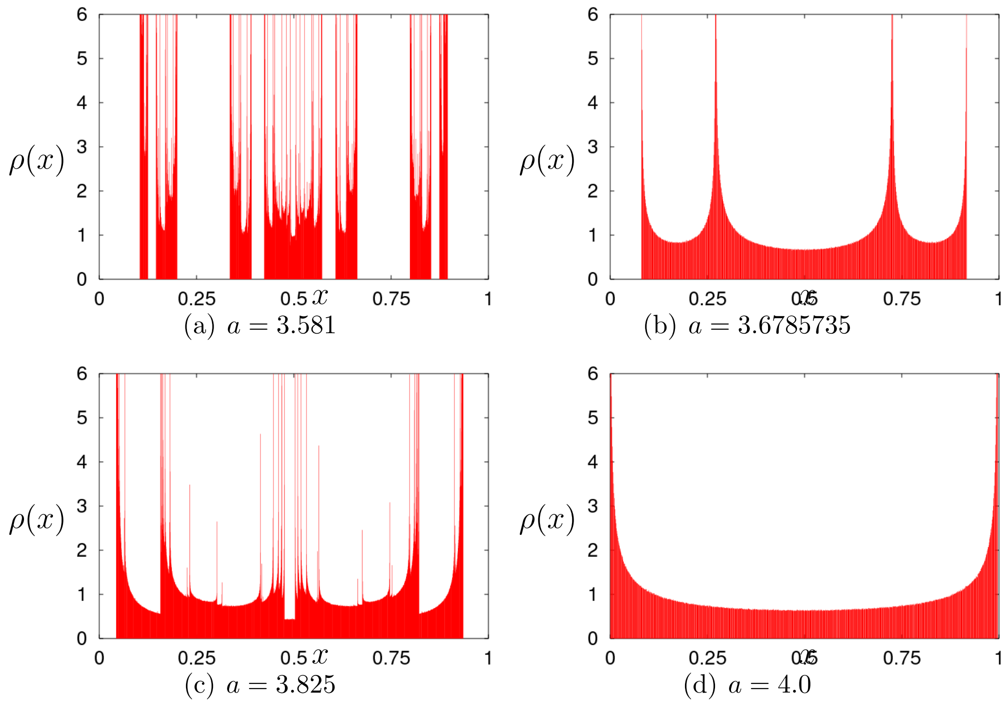


Figure 11.31: Invariant measures of chaotic attractors

Invariant measure of chaotic attractors of system (11.10) for the parameter values marked in Fig. 11.27. See text for a detailed description.

and with the limit cycle $\{x_4^*, x_5^*\}$, which emerges at the first border collision bifurcation α_1^{bc} and becomes unstable at the first pitchfork bifurcation α_1^p. Comparing this behavior with the one of the logistic map, one can remark, that in this system at the first band merging bifurcation α_1^m the bands of a chaotic two-band-attractor merge with each other and collide with an unstable fixed point, which emerges at the first flip bifurcation and becomes unstable at the second one.

11.9.9 Influence of kneading orbits on the behavior of the investigated system beyond α_∞

Boundaries of chaotic attractors

The boundaries of the chaotic attractors of system (11.10) are given by itineraries of critical points $\frac{1}{2} - \varepsilon$ and $\frac{1}{2} + \varepsilon$ ($\varepsilon \to 0$). Note, that the point $\frac{1}{2}$ is a fixed point of this system and therefore its itineraries do not belong to attractors. For all parameter values the boundaries of chaotic attractors of system (11.10) do not belong to this attractors. This property shows a remarkable difference between system (11.10) and the logistic map. In most of the cases, the boundaries of chaotic attractors of the logistic map belong to attractors.

Dealing with multi-band attractors, the following notation is used. For a n-band attractor, the limes supremum of the upper boundary of its m-th band with $m = 1, \ldots, n$ is denoted as $x^{up}_{[n,m]}$. Analogously, the limes infimum of the lower boundary of this band is denoted as $x^{lo}_{[n,m]}$.

The smallest and the largest boundaries for all chaotic attractors of system (11.10) are directly given by the functions g_l and g_r (see Eq. (11.20)):

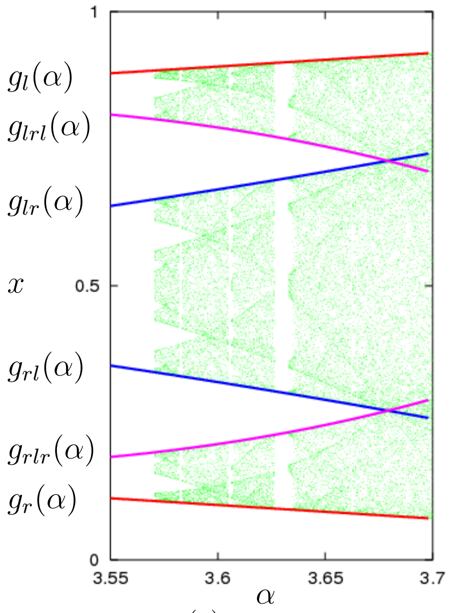
$$x^{up}_{[n,n]} = g_l(\alpha) \qquad x^{lo}_{[n,1]} = g_r(\alpha) \qquad \forall n = 2^k - 1,\ k \in \mathbb{N} \tag{11.29}$$

As expected, the values $x^{up}_{[n,n]}$ and $x^{lo}_{[n,1]}$ are symmetric to each other with respect to the point $x = \frac{1}{2}$.
The functions given by Eq. (11.19) determine the boundaries of the n-band attractors with $n = 3, 7, 15, \ldots$ (that means $\forall n = 2^k - 1$, $k > 1$) existing in the parameter interval $\alpha_\infty < \alpha < \alpha_1^m$. Especially for the 3-band attractors one obtains

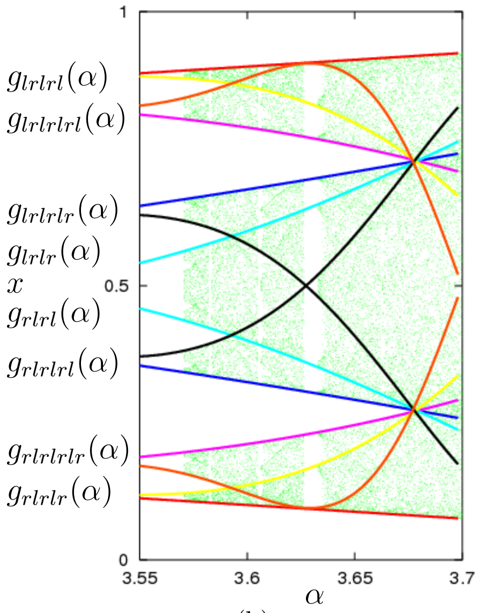
$$\begin{array}{ll} x^{up}_{[3,1]} = g_{rlr}(\alpha) & x^{up}_{[3,2]} = g_{lr}(\alpha) \\ x^{lo}_{[3,2]} = g_{rl}(\alpha) & x^{lo}_{[3,3]} = g_{lrl}(\alpha) \end{array} \tag{11.30}$$

whereby the used functions are determined by Eq. (11.21) and (11.22). Note, that the boundaries $x^{lo}_{[3,1]}$, $x^{up}_{[3,3]}$ are already determined by Eq. (11.29). This is the analytic result for all six boundaries of the 3-band attractors of system (11.10), which exists in the parameter interval $\alpha_2^m \leq \alpha < \alpha_1^m$ (see Fig. 11.32.(a)).
The same procedure can be applied for the further chaotic attractors. For instance, from the 14 boundaries of the 7-band attractors six boundaries are given by the same functions as the boundaries of the 3-band attractors:

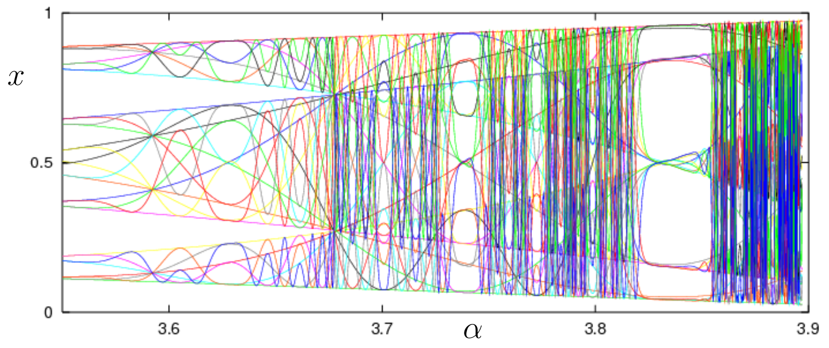
$$\begin{array}{lllllll} x^{lo}_{[7,1]} & = & g_r(\alpha) & & x^{up}_{[7,5]} & = & g_{lr}(\alpha) \\ x^{up}_{[7,2]} & = & g_{rlr}(\alpha) & & x^{lo}_{[7,6]} & = & g_{lrl}(\alpha) \\ x^{lo}_{[7,3]} & = & g_{rl}(\alpha) & & x^{up}_{[7,7]} & = & g_l(\alpha) \end{array} \tag{11.31}$$



(a)



(b)



(c)

Figure 11.32: Boundaries of chaotic attractors

Boundaries of chaotic attractors. In (a) are shown the six functions, defining the boundaries of 3-band attractors. In (b) are shown additionally the eight functions, which define together with the six function of (a) the 14 boundaries of 7-band attractors. In (c) are shown the 30 functions defining the boundaries of 15-band attractors. Note, that these function also reveal the basic structure of the bifurcation diagram, including periodic windows, presented in Figs. 11.21 and 11.27.

The remaining eight boundaries (see Fig. 11.32.(b)) can be determined as follows:

$$\begin{array}{lcllcl} x^{up}_{[7,1]} & = & g_{rlrlr}(\alpha) & x^{up}_{[7,4]} & = & g_{lrlr}(\alpha) \\ x^{lo}_{[7,2]} & = & g_{rlrlrlr}(\alpha) & x^{lo}_{[7,5]} & = & g_{lrlrlr}(\alpha) \\ x^{up}_{[7,3]} & = & g_{rlrlrl}(\alpha) & x^{up}_{[7,6]} & = & g_{lrlrlrl}(\alpha) \\ x^{lo}_{[7,4]} & = & g_{rlrl}(\alpha) & x^{lo}_{[7,7]} & = & g_{lrlrl}(\alpha) \end{array} \qquad (11.32)$$

Locating of band merging bifurcations
Using the knowledge about the boundaries of the multi-band attractors existing within the band merging bifurcation cascade, one is able to calculate the parameter values where the band merging bifurcations occur. For this aim one have to calculate the cross-sections of the corresponding boundaries. For instance, the parameter value α^m_1 can be found using any of the equations $x^{up}_{[3,1]} = x^{lo}_{[3,2]}$ or $x^{up}_{[3,2]} = x^{lo}_{[3,3]}$. Additionally we remark, that the points α^m_k for the special cases $k = 1, 2$ can be found more simple taking into consideration, that the bands collide here not only with each other, but also with the already known unstable limit cycle $\{x^*_4, x^*_5\}$ (for $k = 1$) and $\{x^{**}_1, x^{**}_2, x^{**}_3, x^{**}_4\}$ (for $k = 2$). Therefore the value α^m_1 can be found by solving the equations $x^{up}_{[3,1]} = x^*_5$ or $x^{lo}_{[3,3]} = x^*_4$, and the value α^m_2 results for instance as a solution of the equation $x^{up}_{[7,4]} = x^{**}_2$. One gets:

$$\alpha^m_1 = \frac{2}{3}\left(\sqrt[3]{19+3\sqrt{33}} + \frac{4}{\sqrt[3]{19+3\sqrt{33}}} + 1\right) \approx 3.678573511 \qquad (11.33)$$

$$\alpha^m_2 \approx 3.59257218410697865 \qquad (11.34)$$

Peaks of the invariant measure
The functions $g_{\dots}(\alpha)$, defined so far, play an important role also after the merging of the corresponding bands. Here these functions (see Fig. 11.32.(c)) determine the peaks of the invariant measure of the chaotic attractors, as it is shown in [84]. Due to this fact all chaotic attractors of system (11.10) are symmetric with respect to the point $x = \frac{1}{2}$. Fig. 11.31 demonstrates the invariant measure $\rho(x)$ for some of these attractors. The first one (Fig. 11.31.(a)) corresponds to a seven-band-attractor, which exists in the band merging bifurcation cascade before the merging with the unstable 4-periodic limit cycle. The second one (Fig. 11.31.(b)) emerge at the point of the band merging bifurcation $\alpha = \alpha^m_1$, given by Eq. (11.33). Note, that

the invariant measure of the chaotic attractor at this parameter value is a smooth curve. It is because all relevant functions $g_{\dots}$ (it means all except g_l and g_r) take for $\alpha = \alpha_1^m$ either the value x_4^* or the value x_5^*. The next two figures (Fig. 11.31.(c) and (d)) show symmetrical one-band attractors. For the attractor at the parameter value $\alpha = 4$ the invariant measure can be calculated analytically and is given by $\rho(x) = \left(\pi^2 x(1-x)\right)^{-\frac{1}{2}}$. Note, that this invariant measure is identical with the invariant measure of the logistic map at the same parameter value. It is well-known, that at this parameter value a diffeomorphism exists between the logistic map on the one hand and the tent map $x(n+1) = \mu\left(1 - 2\left|x(n) - \frac{1}{2}\right|\right)$ at the parameter value $\mu = 1$ on the other hand. This diffeomorphism is given by $h(x) = \frac{2}{\pi} \arcsin \sqrt{x}$. It can be shown, that the same diffeomorphism exists between system (11.10) and the tent map also. The invariant measure can be determined by applying the diffeomorphism to the invariant measure of the attractor of the tent map, which is given by $\rho(x) = 1$.

11.9.10 Influence of the border collision period doubling scenario on the behavior within periodic windows

The last interesting property of system (11.10), which we would like to consider, concerns the periodic windows within the chaotic regime. Again there are similarities and also differences between system (11.10) and the logistic map. For both systems the periodic windows exist at the same parameter values and occur in the same order, which can be described using the Metropolis-Stein-Stein sequences [109]. The difference between the logistic map and system (11.10) is, that within each specific window in the case of the logistic map the period doubling cascade takes place, and in the case of system (11.10) the border collision period doubling cascade. The bifurcation leading to the formation of the periodic windows is the same in both cases, namely, the tangent bifurcation. For the logistic map there exists a pair of limit cycles after this bifurcation, a stable and an unstable one. For system (11.10) two such pairs emerge after the bifurcation. For increasing parameter values the stable limit cycles of both pairs undergo the border collision bifurcation described above. This bifurcation leads to the formation of a single limit cycle with twice the period. After that the border collision period doubling scenario continues as described above for $\alpha < \alpha_\infty$. Note also, that the two unstable limit cycles emerging at the tangent bifurcation do not collide with the partition border. As in the case of the logistic map, these limit cycles lead to the global bifurcation (crisis) at which the periodic window is closed.

11.9.11 Summary and outlook

The border collision period doubling scenario is considered here. Using a discontinuous map on the interval $[0, 1]$, the properties of this scenario, its similarities and differences with the classical period doubling scenario are investigated. It is shown that the border collision period doubling scenario is formed by a sequence of pairs of bifurcations. Each pair consists of a border collision bifurcation and a pitchfork bifurcation. The symmetry breaking and symmetry recovering phenomenon within each pair of bifurcations plays an important role for the understanding of this scenario. It is further shown, how the border collision phenomenon influences the band merging scenario and the behavior within periodic windows in the chaotic regime. Some results concerning boundaries of chaotic attractors as well as determining of border collision and band merging bifurcations are obtained based on the technique of kneading orbits.

The following question remains still open: as it is shown in [53], the investigated system (11.10) represents a special kind of Poincaré return map for the well-known Lorenz system [97]. Therefore, a relationship between bifurcations occurring in the Lorenz system and the border collision period doubling scenario must exist. Based on the hypothesis, that the border collision bifurcations in system (11.10) correspond to homoclinic bifurcations in the Lorenz system, this relationship should be investigated in more detail.

Chapter 12

Conclusion

In this work, the **AnT 4.669** software package is presented. Its capabilities and features are illustrated by many examples from the field of nonlinear dynamics. A lot of results of typical investigation and analysis tasks are shown, demonstrating the flexibility and the broad applicability of the software. The main advantages are:

- ▶ multi-platform support (UNIX-like systems and Windows systems)
- ▶ system function plug in mechanism
- ▶ powerful scan mechanism
- ▶ distributed computing
- ▶ open source development
- ▶ usage of generic concepts
- ▶ support of many classes of dynamical systems
- ▶ graphical user interface
- ▶ visualization

Of course there are a lot of other simulation and analysis tools in the field of nonlinear dynamics. Among them are:

- ▶ AUTO [6]
- ▶ DsTool [37]
- ▶ Dynamics Solver [40]
- ▶ PHASER [136]
- ▶ XPP [199]

See also [38] and [5] for a more complete list and an overview about the features and capabilities of the software tools. Each of these tools has its capabilities, features and advantages and often its own specific application area making a comparison of the tools really difficult if not impossible. However, none of the tools known so far allows the simulation of such a broad spectrum of dynamical systems, provides such a powerful scanning mechanism and allows the distributed computing in such an easy way as the **AnT 4.669** software package. Currently, there are still a lot of possible improvements and enhancements like:

- Extension of integration methods:
  - usage of third party ODE, DDE and FDE integrators
  - implementation of symplectic integrators
- Extension of PDE solvers
  - Implementation of 2D-PDEs
  - Implementation of adaptive grid methods for PDEs
- Extension of investigation methods:
  - continuation method
  - local divergence rates
  - ...
- Improvement of the visualization
  - 'attractor flight'
  - more sophisticated coloring schemes
- Implementation of new system classes
  - Differential Algebraic Equations (DAEs)
  - Integro Differential Equations (IDEs)
  - Partial Functional Differential Equations (PFDEs)
  - ...

Appendix A

Reduction of the parameter space

The following example shall elucidate the reduction of the parameter space mentioned in Sec. 3.2. Consider for instance a particle or in the strict mathematical sense a ***mass point*** attached to an ***ideal spring***[1]: When excited, the relaxation of the mass point from its displacement of the equilibrium represents a movement of the mass point in the course of time, and is hence a dynamical process. This system is denoted as ***damped oscillator***. Due to the fact, that the relaxation is a continuous movement, the adequate mathematical model is a dynamical system continuous in time, although it is possible to derive also dynamical systems discrete in time if one is interested only in some specific parts of the dynamics. When the movement of the mass point is restricted to only one dimension, the mass of the ideal spring and the air friction are neglected, the corresponding equation of motion of this dynamical system is given by:

$$m\frac{d^2}{dt^2}s\left(t\right)+f\frac{d}{dt}s\left(t\right)+ks\left(t\right) \quad = \quad 0 \tag{A.1}$$

This natural description involves besides the time t and the state variable $s(t)$ as a function of time, representing the displacement of the mass point from its equilibrium, also the three parameters m, f and k, resulting in a

[1]An ideal spring is a spring for which Hooks law is exactly valid for arbitrary small or large displacements.

parameter space dimension of $N_p = 3$. Hereby m is the particle mass or mass of the mass point, f is the ***friction constant*** and k is the ***spring constant***. As already mentioned, a scaling of time and the state variable reduces redundant parameters. Using the following scaling of time and state

$$t = \alpha t' \tag{A.2}$$
$$s(t) = \beta \tilde{s}(t) = \beta \tilde{s}(\alpha t') = \beta s'(t') \tag{A.3}$$

involves the two ***scaling parameters*** α and β and leads to the following equation of motion for the scaled state variable $s'(t')$:

$$\frac{d^2}{dt'^2} s'(t') + f' \frac{d}{dt'} s'(t') + s'(t') = 0 \tag{A.4}$$

when the scaling parameters α and β are chosen as

$$\alpha = \sqrt{m\beta} \tag{A.5}$$
$$\beta = \frac{1}{k} \tag{A.6}$$

Hereby, the new single parameter f' defined by:

$$f' = \frac{\beta}{\alpha} f \tag{A.7}$$

The resulting dynamical system (A.4) has only the single parameter f' defined by:

$$f' = \frac{\beta}{\alpha} f \tag{A.8}$$

and hence the minimal set of parameters with a parameter space dimension of $N_p = 1$.
As pointed out, an appropriate scaling of time and space or state leads in some cases to a remarkable reduction of the parameter space which causes a simplification of investigations and especially parameter studies due to the decreased parameter space dimension.

Appendix B

Addition of machine numbers

B.1 A simple example program

The fact, that arithmetic operations of machine numbers are not closed can be easily demonstrated. The following simple C program performs the addition of two floating point numbers, which are exactly representable by machine numbers according to the IEEE 754 standard.

```
#include<stdio.h>
#include<math.h>

int main (void) {
    int e1,e2; double x;

    printf("\nPlease enter e1 = ");scanf("%d",&e1);
    printf("Please enter e2 = ");scanf("%d",&e2);

    printf("\nmn1 = 2^e1 =          %20.41g\n",pow(2,e1));
    printf("mn2 = 2^e2 =           %20.41g\n",pow(2,e2));

    x = pow(2,e1) + pow(2,e2);

    printf("\n(float) (mn1 + mn2) =  %20.41g\n",  (float) (x));
    printf("(double) (mn1 + mn2) = %20.41g\n", (double) (x));

    return(0);  }
```

The program requires two *integer* input values. For these two values it outputs the corresponding powers to the basis 2 which are both representable machine numbers according to the IEEE 754 standard. After that it performs a simple addition of these two machine numbers and stores the result in the *double precision* variable `x`. Then it outputs this result in *float* and *double* precision.

B.2 Characteristic case studies

The program presented in section B.1 produces the following five outputs, when provided with the corresponding input values.

1.) Addition of 2^0 and 2^{23}

```
Please enter e1 = 0
Please enter e2 = 23

mn1 = 2^e1 =                             1
mn2 = 2^e2 =                       8388608

(float) (mn1 + mn2) =              8388609
(double) (mn1 + mn2) =             8388609
```

As one can easily see, the result is correct for both precisions.

2.) Addition of 2^0 and 2^{24}

```
Please enter e1 = 0
Please enter e2 = 24

mn1 = 2^e1 =                             1
mn2 = 2^e2 =                      16777216

(float) (mn1 + mn2) =             16777216
(double) (mn1 + mn2) =            16777217
```

Now only the *double* precision result is correct. This is due to the fact, that the sum of 2^0 and 2^{24} is no longer a representable machine number in *float* precision (see sections 10.2.1 and 10.3 for a detailed explanation).

3.) Addition of 2^{20} and 2^{44}

```
Please enter e1 = 20
Please enter e2 = 44

mn1 = 2^e1 =                          1048576
mn2 = 2^e2 =                   17592186044416

(float) (mn1 + mn2) =          17592186044416
(double) (mn1 + mn2) =         17592187092992
```

In this case the error is not just a small quantity although of course the relative error is of the same order of magnitude.

4.) Addition of 2^{-10} and 2^{42}

```
Please enter e1 = -10
Please enter e2 = 42

mn1 = 2^e1 =                     0.0009765625
mn2 = 2^e2 =                    4398046511104

(float) (mn1 + mn2) =           4398046511104
(double) (mn1 + mn2) = 4398046511104.0009765625
```

In this example the two exponents are chosen such that their difference of 52 is exactly the number of bits in the mantissa of the *double* precision representation. Hence the result of the addition is a representable machine number and therefore correct within the *double* precision representation.

5.) Addition of 2^{-10} and 2^{43}

```
Please enter e1 = -10
Please enter e2 = 43

mn1 = 2^e1 =                      0.0009765625
mn2 = 2^e2 =                     8796093022208

(float) (mn1 + mn2) =            8796093022208
(double) (mn1 + mn2) =           8796093022208
```

Now the difference of the two exponents is 53 and therefore the result of the addition is no longer a representable machine number using *double precision.* As a result of that, the output represents the machine number to which the exact result of the addition is mapped.
As one can see, the result of a simple addition of two floating point numbers is not always correct, because it is not a representable machine number with respect to the chosen precision, whereby it even cannot be guaranteed, that the error is a small quantity. This demonstrates, that one has to be always careful when interpreting computational results.

Appendix C

Differential equations with higher order time derivatives

C.1 Motivation

Due to the architecture of the **AnT 4.669** simulation package, only of first order time derivatives of the state vector are treatable. To be able to deal with ordinary differential equations (ODEs), delay differential equations (DDEs) and partial differential equations (PDEs) with higher order time derivatives of the state vector, one has to transform the original differential equation into an equivalent system of differential equations with first order time derivatives. Due to the fact, that non-autonomous dynamical systems can be easily converted into autonomous ones by extending the state space introducing a new state variable representing the time, only autonomous dynamical systems are considered (see Chap. 3, the remark 1 on page page 47).

C.2 Transformation of differential equations with higher order time derivatives

In the following, the compact notation $d_t^k = \frac{d^k}{dt^k}$ for the time derivative operators is used.

C.2.1 Transformation of ordinary differential equations (ODEs)

Consider the following ordinary differential equation (ODE):

$$d_t^n \underline{s}(t) \quad = \quad \underline{f}\big(\underline{s}(t), d_t\underline{s}(t), \ldots, d_t^{n-1}\underline{s}(t), \underline{p}\big) \tag{C.1}$$

As introduced in section 3.2, $\underline{s}(t)$ is the N_s-dimensional state vector and $\underline{p}$ is a N_p-dimensional vector of system parameters. Defining now $n-1$ new state vectors $\underline{s}_1(t), \ldots, \underline{s}_{n-1}(t)$ and identifying $\underline{s}_0(t)$ with $\underline{s}(t)$, Eq. (C.1) can be easily converted into the following set of differential equations where only first order time derivatives of the new state vectors appear on the right hand side.

$$\begin{aligned} d_t\underline{s}_0(t) &= \underline{s}_1(t) \\ &\vdots \\ d_t\underline{s}_i(t) &= \underline{s}_{i+1}(t) \\ &\vdots \\ d_t\underline{s}_{n-1}(t) &= \underline{f}\big(\underline{s}_0(t), \underline{s}_1(t), \ldots, \underline{s}_{n-1}(t), \underline{p}\big) \end{aligned} \tag{C.2}$$

The differential equation (C.1) and the system of differential equations (C.2) are equivalent and as a consequence, dynamical systems which can be described by Eq. (C.1) are fully supported by the **AnT 4.669** software package.

Due to the fact, that this transformation is always possible for all explicit ODEs, all differential equations of this type can be simulated and investigated by the **AnT 4.669** software package.

C.2.2 Transformation of delay differential equations (DDEs)

Consider the following delay differential equation (DDE):

$$\begin{aligned} d_t^n \underline{s}(t) \quad = \quad & \underline{f}\big(\underline{s}(t), \underline{s}(t-\tau_1), \ldots, \underline{s}(t-\tau_m), \\ & \quad d_t\underline{s}(t), d_t\underline{s}(t-\tau_1), \ldots, d_t\underline{s}(t-\tau_m), \ldots, \\ & \quad d_t^{n-1}\underline{s}(t), d_t^{n-1}\underline{s}(t-\tau_1), \ldots, d_t^{n-1}\underline{s}(t-\tau_m), \underline{p}\big) \end{aligned} \tag{C.3}$$

Again as introduced ins section 3.2, $\underline{s}(t)$ is the N_s-dimensional state vector and $\underline{p}$ is a N_p-dimensional vector of system parameters and $\tau_1, \ldots, \tau_m$ is a finite set of time delays. Now, as in section C.2.1, the $n-1$ new state vectors $\underline{s}_1(t), \ldots, \underline{s}_{n-1}(t)$ are defined and $\underline{s}_0(t)$ is identified with $\underline{s}(t)$. Eq. (C.3) can now easily be converted into the following set of differential equations where on the right hand side only first order time derivatives of the new state vectors appear.

$$\begin{aligned}
d_t\underline{s}_0(t) &= \underline{s}_1(t) \\
&\vdots \\
d_t\underline{s}_i(t) &= \underline{s}_{i+1}(t) \\
&\vdots \\
d_t\underline{s}_{n-1}(t) &= \underline{f}\big(\underline{s}_0(t), \underline{s}_0(t-\tau_1), \ldots, \underline{s}_0(t-\tau_m), \\
&\quad \underline{s}_1(t), \underline{s}_1(t-\tau_1), \ldots, \underline{s}_1(t-\tau_m), \ldots, \\
&\quad \underline{s}_{n-1}(t), \underline{s}_{n-1}(t-\tau_1), \ldots, \underline{s}_{n-1}(t-\tau_m), \underline{p}\big)
\end{aligned} \tag{C.4}$$

The differential equation (C.3) and the system of differential equations (C.4) are equivalent and as a consequence, dynamical systems which can be described by Eq. (C.3) are fully supported by the **AnT 4.669** software package.

Due to the fact, that this transformation is always possible for explicit DDEs, all differential equations of this type can be simulated and investigated by the **AnT 4.669** software package.

Appendix D

Partial differential equations with higher order time derivatives

D.1 Motivation

Due to the architecture of the **AnT 4.669** simulation package, only first order time derivatives of the state vector are allowed. To be able to treat partial differential equations with higher order time derivatives or such equations where mixed partial derivatives with respect to time and other independent variables occur, one has to transform the original PDE into a system of PDEs with only first order time derivatives. Due to the fact, that this transformation is not always possible, not all partial differential equations can be simulated and investigated by **AnT 4.669**.

D.2 Transformation of partial differential equations with higher order time derivatives

D.2.1 PDEs with pure time and space derivatives

Consider the following partial differential equation:

$$\begin{aligned}
\frac{\partial^n}{\partial t^n}\underline{s}(\underline{x},t) &= \underline{f}\big(\underline{s}(\underline{x},t), \frac{\partial \underline{s}(\underline{x},t)}{\partial x_i}, \frac{\partial \underline{s}(\underline{x},t)}{\partial t}, \frac{\partial^2 \underline{s}(\underline{x},t)}{\partial x_{i_1}\partial x_{i_2}}, \frac{\partial^2 \underline{s}(\underline{x},t)}{\partial t^2}, \\
&\quad \ldots, \frac{\partial^{n-1} \underline{s}(\underline{x},t)}{\partial x_{i_1}\ldots\partial x_{i_{n-1}}}, \frac{\partial^{n-1}}{\partial t^{n-1}}\underline{s}(\underline{x},t), \underline{p}\big) \qquad \text{(D.1)}
\end{aligned}$$

As introduced in section 3.2, $\underline{s}(\underline{x},t)$ is the N_s-dimensional state vector and $\underline{p}$ is a N_p-dimensional vector of system parameters. If one defines $n-1$ new state vectors $\underline{s}_1(\underline{x},t),\ldots,\underline{s}_{n-1}(\underline{x},t)$, whereby $\underline{s}_0(\underline{x},t)$ is identified with $\underline{s}(\underline{x},t)$, then Eq. (D.1) can be converted into the following set of partial differential equations where on the right hand side only first order time derivatives of the new state vectors appear.

$$\begin{aligned}
\frac{\partial}{\partial t}\underline{s}_0(\underline{x},t) &= \underline{s}_1(\underline{x},t) \\
&\vdots \\
\frac{\partial}{\partial t}\underline{s}_i(\underline{x},t) &= \underline{s}_{i+1}(\underline{x},t) \qquad \text{(D.2)} \\
&\vdots \\
\frac{\partial}{\partial t}\underline{s}_{n-1}(\underline{x},t) &= \underline{f}\big(\underline{s}_0(\underline{x},t), \frac{\partial \underline{s}_0(\underline{x},t)}{\partial x_i}, \underline{s}_1(\underline{x},t), \frac{\partial^2 \underline{s}_0(\underline{x},t)}{\partial x_{i_1}\partial x_{i_2}}, \underline{s}_2(\underline{x},t), \\
&\quad \ldots, \frac{\partial^{n-1} \underline{s}_0(\underline{x},t)}{\partial x_{i_1}\ldots\partial x_{i_{n-1}}}, \underline{s}_{n-1}(\underline{x},t), \underline{p}\big)
\end{aligned}$$

The partial differential equation (D.1) and the system of partial differential equations (D.2) are equivalent and as a consequence, dynamical systems which can be described by partial differential equations like (D.1) are supported by the **AnT 4.669** software package.

D.2.2 PDEs with mixed time and space derivatives

If the dynamical system has to be described by partial differential equations with mixed time and spatial derivatives, a conversion to a system of partial differential equations like the conversion presented in section D.2 is not always possible. However, the dynamical systems corresponding described by partial differential equations where the conversion is possible are also supported by the **AnT 4.669** software package. For instance, if the highest order time derivative is a pure time derivative, then the procedure presented in section D.2.1 is still applicable and the dynamical system is supported.

Appendix E

Client/server example runs

E.1 Motivation

In chapter 7 the client/server architecture of the **AnT 4.669** software package is presented. In this chapter some illustrative examples are collected which demonstrate some of the features and capabilities of this architecture. The capability of the computing clients to adapt their number of fetched scan-points from the server to their current load for instance is illustrated in section E.2.

E.2 Load based adaptation of the clients

The following is a typical output of a client during the execution of a scan. In this case the logistic map was investigated in the parameter interval $\alpha \in [0.001, 4]$ using 300 scan points with 50000 iterations per scan-point and the period analysis investigation method which tries to select the parameter intervals in which the periods $2, 3, 4, 5, 6, 7, 8, 16, 32, 64$ occur. Because this is an illustrative example, a server and only one client were used. The server and the client were started using the following commands:

```
AnT logistic -m server
AnT logistic -m client -s serverhostname -n 1 -t 3
```

As a consequence of the options `-n 1`, specifying the number of scan-points to be fetched from the server initially and `-t 3`, specifying the time in seconds which a client should run before fetching further scan-points from the server.

After the number of scan-points which were fetched the first time are processed, the client can calculate the time necessary for the tasks to be done for one scan-point and with this information the client can calculate how many scan points it needs to be busy for the specified time. Here, the client fetches the next time about 30 scan-points from the server. After the third fetch, additional processes were started on the client causing a decrease in the number of fetched scan-points form server to about 20 scan-points in the average. In the last phase of the scan, the number of fetched scan-points increases again to about 30, because the additional processes were stopped. This simple example demonstrates the capability of adaptation of the clients.

```
--//------------/------------------------------
 // AnT 4.669  / Release 3a, (c) 1999-2004
//------------/--------------------------------
Name of the dynamical system: logistic
Name of the configuration file:
runmode: client
Adding investigation method for key: 'period_analysis' ...
investigation method added successfully.
Adding investigation method for key: 'general_trajectory_evaluations' ...
investigation method added successfully.
Adding investigation method for key: 'lyapunov_exponents_analysis' ...
Number of lyapunov exponents to be calculated: 1
use L2 norm for vectors.
use random vectors as initial deviations.
investigation method added successfully.
allocated size of the continuous orbit: 1000
the following real-valued objects are scannable:
    alpha
    initial_state[0][0]
    parameter[0]
the following int-typed objects are scannable:
    lyapunov_exponents_analysis::steps_between_reorthonormalization
scannable objects inspected.
loading the system... system loaded successfully
```

```
starting scanMachine...
Client speed: 0 scanpoints/second
fetching 1 scanpoints
numScanPoints: 1
Client speed: 4.052142976 scanpoints/second
fetching 32 scanpoints
numScanPoints: 32
Client speed: 10.72182689 scanpoints/second
fetching 32 scanpoints
numScanPoints: 32
Client speed: 5.910639261 scanpoints/second
fetching 17 scanpoints
numScanPoints: 17
Client speed: 7.430482374 scanpoints/second
fetching 22 scanpoints
numScanPoints: 22
Client speed: 10.67583365 scanpoints/second
fetching 32 scanpoints
numScanPoints: 32
Client speed: 6.991285145 scanpoints/second
fetching 21 scanpoints
numScanPoints: 21
Client speed: 4.339294836 scanpoints/second
fetching 15 scanpoints
numScanPoints: 15
Client speed: 4.100501655 scanpoints/second
fetching 16 scanpoints
numScanPoints: 16
Client speed: 6.333102175 scanpoints/second
fetching 26 scanpoints
numScanPoints: 26
Client speed: 6.506928252 scanpoints/second
fetching 21 scanpoints
numScanPoints: 21
Client speed: 7.51931928 scanpoints/second
fetching 29 scanpoints
numScanPoints: 29
Client speed: 10.89106568 scanpoints/second
```

```
fetching 33 scanpoints
numScanPoints: 33
Client speed: 9.574331044 scanpoints/second
fetching 30 scanpoints
numScanPoints: 3
Client speed: 7.618041554 scanpoints/second
fetching 29 scanpoints
numScanPoints: 0
'ANPClient::getScanPoint': could not fetch any scan points!
Client is getting down... Bye!
Bye!
```

E.3 Taking over for a broken down client

The following example demonstrates the possibility of taking over the tasks given to another client, but which is broken down. The server and the client were started using the following commands:

```
AnT logistic -m server
AnT logistic -m client -s serverhostname -n 100
```

Then after the complete calculation is finished, without breakdown of the client, the server stops with the following message:

```
total: 300, calculated 300, progress 100.00 %.
closing socket: 4
accepting connections now
connection established
on opened socket: 4
Done.
closing socket: 4
83.309132 sec
3.601045801 scanpoints/sec
scanMachine stopped.
simulation successfully completed.
Bye!
```

In the case of a breakdown of the only computing client, the computation stops but the server is still running and accepting connections from possible other clients. Starting another computing client on the same (after a possible reboot) or another machine, the output of the server after the complete calculation is finished looks like:

```
total: 300, calculated 300, progress 100.00 %.
closing socket: 4
accepting connections now
connection established
on opened socket: 4
Done.
closing socket: 4
130.825503 sec
2.293130874 scanpoints/sec
scanMachine stopped.
simulation successfully completed.
Bye!
```

Of course - as expected - when starting again only one comparable client, the overall performance is less in this case as indicated by the total time needed for the scan and the average value of the calculated scan-points per second.

Bibliography

[1] Alekseev, V. M. *Symbolic Dynamics, 11th Mathematical School.* Kiev, 1976. In Russian.

[2] Annual report of the Lawrence Livermore National Laboratory. `http://www.llnl.gov/annual01/`, 2001.

[3] Annual report of the Lawrence Livermore National Laboratory. `http://www.llnl.gov/annual02/`, 2002.

[4] The Ant 4.669 software package. `http://www.ant4669.de/`.

[5] The ant 4.669 software package. `http://www.AnT4669.de`.

[6] Auto. `http://indy.cs.concordia.ca/auto/`.

[7] Avrutin, V. *Zum Verhalten dynamischer Systeme mit einer stckweise glatten Systemfunktion.* PhD thesis, Universität Stuttgart, 2004.

[8] Avrutin, V., Wackenhut, G., and Schanz, M. On dynamical systems with piecewise defined system functions. In *Proc. of Int. Conf. "'Tools for Mathematical Modelling"' MATHTOOLS'99* (1999).

[9] Banerjee, S., and Grebogi, C. Border collision bifurcation in two-dimensional piecewise smooth maps. *Phys. Rev.* **E59**, 4 (1999), 4052–4061.

[10] Batista, A., and Carlson, J. M. Bifurcations from steady sliding to slip in boundary lubrication. *Phys. Rev.* **E57** (1998), 4986–4996.

[11] Berryman, A. Population cycles of the Douglas-fir tussock moth (Lepidoptera: Lymantriidae): the time-delay hypothesis. *Canadian Entomologist* **110** (1978), 513–518.

[12] Blazejczyk-Okolewska, B., and Kapitaniak, T. Dynamics of impact oscillators with dry friction. *Chaos, Solitons & Fractals* **7**, 9 (1996), 1455–1459.

[13] Blazejczyk-Okolewska, B., and Kapitaniak, T. Co-existing attractors of impact oscillators. *Chaos, Solitons & Fractals* **9**, 8 (1998), 1439–1443.

[14] Bowen, R. Symbolic dynamics. *Ann. Math. Soc. 8* (1982).

[15] Brigham, E. *The Fast Fourier Transform and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[16] Butcher, M., Nassauer, O., and Young, S. Nuclear Futures: Western European Options for Nuclear Risk Reduction. Tech. rep., British American Security Information Council and the Berlin Information-center for Transatlantic Security (BITS)., December 1998. BASIC/BITS Research Report 98.6.

[17] Chin, W., Ott, E., Nusse, H. E., and Grebogi, C. Grazing bifurcations in impact oscillators. *Phys. Rev.* **E50**, 6 (1994).

[18] CLAPACK (f2c'ed version of LAPACK). `http://www.netlib.org/clapack/`.

[19] Collet, P., and Eckmann, J.-P. *Iterated maps on the interval as dynamical systems.* Birkhäuser, 1980.

[20] Cook, L. Oscillation in the simple logistic growth model. *Nature* **207** (1965), 316.

[21] Cooley, J., and Tukey, O. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comput.* **19** (1965), 297.

[22] Crawford, D. CTH 3D Comet Impact Simulations. `http://sherpa.sandia.gov/planet-impact/comet/`. Sandia National Laboratories.

[23] Crawford, D. The Impact of an Asteroid off the New York Coast. http://sherpa.sandia.gov/planet-impact/asteroid/, 1998. Sandia National Laboratories.

[24] Crawford, D. Real (not reel) deep impacts: Sandia scientists predict what an asteroid strike would look like, really. http://www.sandia.gov/media/comethit.htm, 1998. Sandia National Laboratories.

[25] Concurrent Versions System. http://www.cvshome.org/.

[26] cygwin. http://www.cygwin.com/.

[27] di Bernardo, M., Budd, C. J., and Champneys, A. R. Grazing, skipping and sliding: analysis of the nonsmooth dynamics of the dc/dc buck converter. *Nonlinearity* **11**, 4 (1998), 858–890.

[28] di Bernardo, M., Budd, C. J., and Champneys, A. R. Corner collision implies border-collision bifurcation. *Physica* **D154**, 3-4 (2001), 171–194.

[29] di Bernardo, M., Budd, C. J., and Champneys, A. R. Grazing and Border-Collisions in Piecewise-Smooth Systems: A Unified Analytical Framework. *Phys. Rev. Lett.* **86**, 12 (2001), 2554–2556.

[30] di Bernardo, M., Budd, C. J., and Champneys, A. R. Normal form maps for grazing bifurcations in n-dimensional piecewise-smooth dynamical systems. *Physica* **D160**, 3-4 (2001), 222–254.

[31] di Bernardo, M., Garofalo, F., Glielmo, L., and Vasca, F. Switchings, Bifurcations and Chaos in DC/DC Converters. *Fundamental Theory and Applications* **45** (1998), 133–141.

[32] di Bernardo, M., Johansson, K. H., and Vasca, F. Sliding bifurcations in piecewise smooth dynamical systems. In *Proceedings NDES00* (Catania, Italy, June 2000).

[33] di Bernardo, M., Vasca, F., and Olivar, G. Routes to chaos in the voltage controlled buck converter without latch. In *Nonlinear phenomena in Power Electronic Circuits*. IEEE Press, 2001.

[34] Ding, E. Wave number for unimodal maps. *Phys. Rev.* **A37**, 5 (1988), 1827–1830.

[35] Ding, E. Wave number and symbolic dynamics. *Phys. Rev.* **A39**, 2 (1989), 816–829.

[36] Doxygen.
`http://www.doxygen.org/`.

[37] Dynamical systems: Simulation and visualization.
`http://www.geom.uiuc.edu/software/dstool/`.

[38] Dynamical systems software.
`http://www.dynamicalsystems.org/sw/sw/`.

[39] Dutta, M., Nusse, H. E., Ott, E., Yorke, J. A., and Yuan, G. Multi-attractor bifurcations: a source for unpredictability in piecewise smooth systems. *Phys. Rev. Lett.* **83** (1999), 4281–4284.

[40] Dynamics solver.
`http://tp.lc.ehu.es/jma/ds/ds.html`.

[41] Edwards, D., and Hamson, M. *Guide to Mathematical Modelling (Mathematical Guides).* Palgrave Macmillan, 1994.

[42] Feigin, M. I. Doubling of the Oscillation Period with C-Bifurcations in Piecewise-Continuous Systems. *Prikl. Math. Mekh.* **34** (1970), 861–869. (in russian).

[43] Feigin, M. I. On the Generation of Subharmonic Modes in a Piecewise-Continuous System. *Prikl. Math. Mekh.* **38** (1975), 810–818. (in russian).

[44] Feigin, M. I. On the Structure of C-Bifurcation Boundaries of Piecewise-Continuous Systems. *Prikl. Math. Mekh.* **42** (1978), 820–829. (in russian).

[45] Feudel, U., Witt, A., Lai, Y.-C., and Grebogi, C. Basin bifurcation in quasiperiodically forced systems. *Phys. Rev.* **E58**, 3 (1998), 3060–3066.

[46] Fftw.
`http://www.fftw.org/`.

[47] Field, R., Körös, E., and Noyes, R. Oscillations in Chemical Systems. II. Thorough Analysis of Temporal Oscillation in the Bromate-Cerium-Malonic Acid System. *J. Am. Chem. Soc.* **94** (1972).

[48] Foale, S. Analytical determination of bifurcations in an impact oscillator. *Proc. R. Soc. Lond.* **A347** (1994), 353–364.

[49] Free Software Foundation.
`http://www.gnu.org/fsf/`.

[50] Fukuda, T., Buss, M., Hosokai, H., and Kawauchi, Y. Cell structured robotic system cebot: Control, planning and communication methods. *Robotics and Autonomous Systems 7* (1991), 239 – 248.

[51] Fukuda, T., Kawauchi, Y., and Asama, H. Dynamically reconfigurable robotic systems - optimal knowledge allocation for cellular robotic system (cebot). *Journal of Robotics and Mechatronics 2*, 6 (1990), 22 – 30.

[52] Fukuda, T., and Ueyama, T. *Cellular Robotics and Micro Robotic Systems*, vol. 10 of *World Scientific Series in Robotics and Automated Systems*. World Scientific, 1994.

[53] Gambaudo, J.-M., Procaccia, I., Thomae, S., and Tresser, C. New Universal Scenarios for the Onset of Chaos in Lorenz-Type Flows. *Phys. Rev. Lett.* **57**, 8 (1986), 925–928.

[54] Garey, M., and Johnson, D. *Computers and Intractability.* Freeman and Company, San Francisco, 1979.

[55] Garloff, J. Interval mathematics. a bibliography. *Freiburger Intervall-Berichte 85/6* (1985), 1–222.

[56] Garloff, J. Bibliography on interval mathematics. continuation. *Freiburger Intervall-Berichte 87/2* (1987), 1–50.

[57] Gause, G. *The struggle for existence.* Williams and Wilkins, Baltimore, 1934.

[58] Glass, L., and Mackey, M. Oszillation and chaos in physiological control systems. *Science* **197** (1977), 287.

[59] Glass, L., and Mackey, M. *From Clocks to Chaos, The Rhythms of Life*. Princeton University Press, Princeton, NJ, 1988.

[60] Gleick, J. *Chaos: Making a New Science*. Penguin Books, New York, 1988. 144-153.

[61] Gleick, J. *Chaos: Making a New Science*. Penguin Books, New York, 1988.

[62] GNU's Not Unix!
`http://www.gnu.org/`.

[63] GNU autoconf.
`http://www.gnu.org/software/autoconf/`.

[64] GNU automake.
`http://www.gnu.org/software/automake/`.

[65] GNU libtool.
`http://www.gnu.org/software/libtool/`.

[66] Licences.
`http://www.gnu.org/licenses/licenses.html`.

[67] Golub, G., and Loan, C. V. *Matrix Computations*. North Oxford Academic, Oxford, 1983.

[68] GNU General Public License.
`http://www.gnu.org/copyleft/gpl.html`.

[69] Grassberger, P., and Procaccia, I. Measuring the strangeness of strange attractors. *Physica 9* (1983), 189–208.

[70] Guckenheimer, J., and Williams, R. F. Structural Stability of Lorenz Attractors. *Publ. Math. IHES* **50** (1979), 307–320.

[71] Hairer, E., Nørsett, S., and Wanner, G. *Solving Ordinary Differential Equations I*, 2nd revised edition 2000 ed. Springer-Verlag, 1987.

[72] Haken, H. *Laser Theory*, vol. XXV/2c of *Encyclopedia of Physics*. Springer-Verlag, 1970.

[73] Haken, H. *Licht und Materie I: Elemente der Quantenoptik (in german)*. B.I. Wissenschaftsverlag, 1979.

[74] Haken, H. *Licht und Materie II: Laser (in german)*. B.I. Wissenschaftsverlag, 1985.

[75] Haken, H., Schanz, M., and Starke, J. Treatment of combinatorial optimization problems using selection equations with cost terms. part i: Two-dimensional assignment problems. *Physcica* **D134/2** (1999), 227–241.

[76] Hall, B. Beej's guide to network programming. `http://www.ecst.csuchico.edu/~beej/guide/net/`, 2001.

[77] Hénon, M. A two-dimensional mapping with a strange attractor. *Commun. Math. Phys.* **50** (1976), 69.

[78] Hinrichs, N., Oestreich, M., and Popp, K. Dynamics of oscillators with impact and friction. *Chaos, Solitons & Fractals* **8**, 4 (1997), 535–558.

[79] Hitzl, D., and Zele, F. An exploration of the hénon quadratic map. *Physica* **D14** (1985), 305–326.

[80] Hsu, C. S. *Cell-to-Cell Mappings*. Springer, N.Y., 1987.

[81] Hunt, C. *TCP/IP Network Administration, Second Edition*. O'Reilly and Associates, Inc., 1998.

[82] I, P., and Lefever, R. Symmetries breaking instabilities in dissipative systems II. *Journal of Physical Chemistry* **48** (1968), 1695.

[83] Iu, H. H. C., and Tse, C. K. Bifurcation behavior in parallel-connected buck converters. *Fundamental Theory and Applications* **48** (2001), 233–240.

[84] Jensen, R. V., and Myers, C. R. Images of the Critical Points of Nonlinear Maps. *Phys. Rev.* **A32** (1985), 1222–1224.

[85] Kaneko, K. *Theory and applications of coupled map lattices.* John Wiley & Sons, Ltd., 1993.

[86] Karhunen, K. Zur Spektraltheorie Stochastischer Prozesse. *Ann. Acad. Sci. Fennicae* **37** (1946).

[87] Kernighan, B., and Ritchie, D. *The C Programming Language.* Prentice-Hall, Englewood Cliffs New Jersey, 1978.

[88] Kingsland, S. *Modeling nature.* University of Chicago Press, Chicago, 1985.

[89] Kowalczyk, P., and di Bernardo, M. On a Novel Class of Bifurcations in Hybrid Dynamical Systems. In *Hybrid Systems: Computation and Control* (2001), M. di Bebedetto and A. Sangiovanni-Vincentelli, Eds., LNCS 2034, Springer, pp. 361–374.

[90] Kulisch, U., and Miranker, W., Eds. *Symposium held at IBM Research Center, Yorktown Heights, N. Y., 1982* (New York, 1983), Academic Press. ISBN 0-12-428660-7.

[91] Lafrenz, R., Schulé, M., Becht, M., Schanz, M., Molnár, P., Starke, J., and Levi, P. Experimental study of self–organized fault–tolerant behavior in robotic systems. In *Proceedings of the 16th national conference on Autonomous Mobile Systems* (2000), pp. 210–217.

[92] Lamba, H., and Budd, C. J. Scaling of lyapunv exponents at non-smooth bifurcations. *Phys. Rev.* **E50**, 1 (1994), 84–90.

[93] Lauwerier, H. *Fractals: Endlessly Repeated Geometric Figures.* Princeton University Press, Princeton, NJ, 1991. 128-133.

[94] Lauwerier, H. *Fractals: Endlessly Repeated Geometric Figures.* Princeton University Press, Princeton, NJ, 1991.

[95] Lind, D., and Marcus, B. *An introduction to symbolic dynamics and coding.* New York, 1995.

[96] Loève, M. *Probability Theory.* VanNostrand, Princeton, N.J., 1955.

[97] Lorenz, E. N. Deterministic non-periodic flows. *J. Atmos. Sci.* **20** (1963), 130.

[98] Lorenz, E. N. Irregularity: A fundamental property of the atmosphere. *Tellus A* (1984), 36.

[99] Lotka, A. *Elements of Physical Biology.* Williams and Wilkins, Baltimore, 1925.

[100] Lozi, R. Un attracteur étrange du dype attracteur de hénon. *J. Phys. (Paris)* **39** (1978), 69–77.

[101] Maistrenko, Y. L., Maistrenko, V. L., and Chua, L. O. Cycles of Chaotic Intervals in a Time–Delayed Chua's Circuit. *Int. J. Bifurcation and Chaos* **3** (1993), 1557–1572.

[102] Maistrenko, Y. L., Maistrenko, V. L., and Vikul, S. I. Bifurcations of attracting cycles of piecewise linear interval maps. *J. Tech. Phys.* **37**, 3-4 (1996), 367–370.

[103] Maistrenko, Y. L., Maistrenko, V. L., and Vikul, S. I. On period–adding sequences of attracting cycles in piecewise linear maps. *Chaos, Solitons & Fractals* **9**, 1/2 (1998), 67–75.

[104] Maistrenko, Y. L., Maistrenko, V. L., Vikul, S. I., and Chua, L. O. Bifurcations of Attracting Cycles from Time–Delayed Chua's Circuit. *Int. J. Bifurcation and Chaos* **5** (1995), 653–671.

[105] Maple.
`http://www.maplesoft.com/`.

[106] Mathematica.
`http://www.wolfram.com/`.

[107] The Mesa 3D Graphics Library.
`http://mesa3d.sourceforge.net/`.

[108] Metcalfe, V. [comp.unix.programmer] unix-socket-faq for network programming.
`http://www.faqs.org/faqs/unix-faq/socket/`.

[109] Metropolis, N., Stein, M. L., and Stein, P. R. On Finite Limit Sets for Transformations on the Unit Interval. *J. Comb. Theory* **A15** (1973), 25–44.

[110] Milnor, J., and Thurston, W. On iterated maps of the interval. In *Dynamical systems*, J. C. Alexander, Ed., vol. 1342 of *Lecture Notes in Mathematics*. Springer, 1987, pp. 465–563.

[111] MinGW.
`http://www.mingw.org/`.

[112] Misiurevicz, M., and Kawczyński, A. L. Periodic orbits for interval maps with sharp cusps. *Physica* **D52** (1991), 191–203.

[113] Molenaar, J., de Weger, J. G., and van de Water, W. Mappings of grazing-impact oscillators. *Nonlinearity* **14** (2001), 301–321.

[114] Moore, R. *Interval Analysis.* Prentice-Hall, Englewood Cliffs N. J., 1966.

[115] Moore, R. The dawning. *Reliable Computing 5* (1999), 423–424.

[116] Morosawa, S., Nishimura, Y., Taniguchi, M., and Ueda, T. Dynamics of generalized hénon maps. In *Holomorphic Dynamics.* Cambridge University Press, 2000, ch. 7.

[117] Nair, P. R. K., and Nandakumaran, V. M. Existence of multiple attractors and the nature of bifurcations in a discontinuous logistic map. *Pramana* **51**, 3-4 (1998), 377–385.

[118] Nicolis, G., and Prigogine, I. *Self-orgaization in non-equilibrium systems.* Wiley-Interscience, 1977.

[119] Nordmark, A. B. Non-periodic motion caused by grazing incidence in an impact oscillator. *J. Sound Vib.* **145**, 2 (1991), 279–297.

[120] Nordmark, A. B. Universal limit mapping in grazing bifurcations. *Phys. Rev.* **E55**, 1 (1997), 266–270.

[121] Nusse, H. E., Ott, E., and Yorke, J. A. Border-collision bifurcations: An explanation for observed bifurcation phenomena. *Phys. Rev.* **E49**, 2 (1994), 1073–1076.

[122] Nusse, H. E., and Yorke, J. A. Border-Collision Bifurcations including 'period two to period three' Bifurcation for Piecewise Smooth Systems. *Physica* **D57** (1992), 39–57.

[123] Nusse, H. E., and Yorke, J. A. Border-collision bifurcations for piecewise smooth one-dimensional map. *Int. J. Bif. Chaos* **5** (1995), 189–207.

[124] OpenGL.
`http://www.opengl.org/`.

[125] Osipenko, G. Morse spectrum of dynamical systems and symbolic dynamics. *Proceedings of the 15th IMACS World Congress 1* (1997), 25 – 30.

[126] Osipenko, G. Spectrum of a dynamical system and applied symbolic dynamics. *Journal of Mathematical Analysis and Applications 252*, 2 (2000), 587 – 616.

[127] Osipenko, G. S. On a symbolic image of dynamical system. *Interuniv. Collect. sci. Works* (1983), 101 – 105. In Russian.

[128] Osipenko, G. S. Localization of the chain recurrent set by symbolic dynamics methods. In *Proceedings of Dynamics Systems and Applications* (1994), vol. 1, Dynamic Publishers Inc., pp. 227 – 282.

[129] Osipenko, G. S. The periodic points and symbolic dynamics. *Prog. Nonlinear Differ. Equ. Appl. 12* (1994), 261 – 267.

[130] Osipenko, G. S. Construction of attractors and filtrations. *Banach center publication 47* (1999), 173 – 192.

[131] Osipenko, G. S., Romanovsky, J. V., Ampilova, N. B., and Petrenko, E. I. Computation of the morse spectrum. *Journal of Mathematical Sciences 120*, 2 (2004), 1155 – 1166.

[132] Pearl, R., and Reed, L. On the Rate of Growth of the population of the United States since 1790. In *Proceedings of the National Academy of Science* (1920), vol. **6**, pp. 275–288.

[133] Peitgen, H.-O., and Richter, D. *The Beauty of Fractals: Images of Complex Dynamical Systems.* Springer-Verlag, New York, 1986.

[134] Perez, J. M. Mechanism for global features of chaos in a driven nonlinear oscillator. *Phys. Rev.* **A32**, 4 (1985), 2513–2516.

[135] Peterka, F. Bifurcations and transition phenomena in an impact oscillator. *Chaos, Solitons & Fractals* **7**, 10 (1996), 1635–1647.

[136] Phaser.
`http://www.phaser.com/`.

[137] Procaccia, I., Thomae, S., and Tresser, C. First-return maps as a unified renormalization scheme for dynamical systems. *Phys. Rev.* **A35**, 4 (1987), 1884–1900.

[138] Rfc 791: Internet protocol.
`http://www.rfc-editor.org/rfc/rfc791.txt`, 1981.

[139] Rfc 793: Transmission control protocol.
`http://www.rfc-editor.org/rfc/rfc793.txt`, 1981.

[140] Ricker, W. Stock and recruitment. *Journal of the Fisheries Research Board of Canada* **11** (1954), 559–623.

[141] Rikitake, T. Oscillations of a System of Disc Dynamos. *Proc. Cambridge Philos. Soc.* **54** (1958), 89.

[142] Ritchie, D. The development of the C language. *ACM SIGPLAN Notices 28*, 3 (March 1993), 201–208.

[143] Rivest, T. C. C. L. R. *Introduction to algorithms.* The MIT electrical engineering and computer science series. MIT Press, 2000.

[144] Roessler, O. E. An equation for continuous chaos. *Phys. Lett.* **A57** (1976), 397–398.

[145] Russell, D., Hanson, J., and Ott, E. Dimension of strange attractors. *Phys. Rev. Let.* **45** (1980), 1175–1178.

[146] Schanz, M. *Zur Analytik und Numerik zeitlich verzögerter synergetischer Systeme (in german).* PhD thesis, Universität Stuttgart, 1997. Shaker Verlag.

[147] Schulé, M., Schanz, M., Felger, H., Lafrenz, R., Starke, J., and Levi, P. Control of Autonomous Robots in the RoboCup Scenario Using Coupled Selection Equations. In *Proceedings of the 17th national conference on Autonomous Mobile Systems* (2001), pp. 57–63.

[148] Sharkovsky, A. N., and Chua, L. O. Chaos in some 1-d Discontinuous Maps that Appear in the Analysis of Electrical Circuits. *IEEE Trans. on circuits and systems I* **40**, 10 (1993), 722–731.

[149] Shayer, L., and Campbell, S. Stability, bifurcation and multistability in a system of two coupled neurons with multiple time delays. *SIAM J. Appl. Math.* **61**, 2 (2000), 673–700.

[150] SourceForge.net.
`http://sourceforge.net/`.

[151] Sparrow, C. *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors.* Springer-Verlag, 1973.

[152] Starke, J. *Kombinatorische Optimierung auf der Basis gekoppelter Selektionsgleichungen (in german).* PhD thesis, Universität Stuttgart, Verlag Shaker, Aachen, 1997.

[153] Starke, J., Kubota, N., and Fukuda, T. Combinatorial optimization with higher order neural networks - cost oriented competing processes in flexible manufacturing systems. In *Proceedings of the International Conference on Neural Networks (ICNN'95)* (November 1995), vol. 5, IEEE, pp. 2658 – 2663.

[154] Starke, J., and Schanz, M. Dynamical system approaches to combinatorial optimization. In *Handbook of Combinatorial Optimization*, D.-Z. Du and P. Pardalos, Eds., vol. 2. Kluwer Academic Publisher, 1998, pp. 471–524.

[155] Starke, J., Schanz, M., and Haken, H. Self-organized behaviour of distributed autonomous mobile robotic systems by pattern formation principles. In *Distributed Autonomous Robotic Systems 3*, T. Lueth, R. Dillmann, P. Dario, and H. Wörn, Eds. Springer Verlag, Heidelberg, Berlin, New York, 1998, pp. 89 – 100.

[156] Starke, J., Schanz, M., and Haken, H. Treatment of combinatorial optimization problems using selection equations with cost terms. part ii: Np-hard three-dimensional assignment problems. *Physica* **D134/2** (1999), 242–252.

[157] Stewart, G. On the Early History of the Singular Value Decomposition. *SIAM Review* **35** (1993), 551.

[158] Stroustrup, B. *The C++ Programming Language.* Addison-Wesley, 1991.

[159] Stroustrup, B. A history of C++: 1979-1991. *ACM SIGPLAN Notices 28*, 3 (March 1993), 271–297.

[160] Stroustrup, B. *The Design and Evolution of C++.* Addison-Wesley, 1994.

[161] Tarjan, R. *Data structures and network algorithms.* Society for Industrial and Applied Mathematics. Philadelphia, Pa., 1991.

[162] Todd, M. D., and Virgin, L. N. An experimental impact oscillator. *Chaos, Solitons & Fractals* **8**, 4 (1997), 699–714.

[163] TOP 500 SUPERCOMPUTER SITES.
`http://www.top500.org/`.

[164] Ulam, S., and von Neumann, J. On combination of stochastic and deterministic processes. *Bull. Am. Math. Soc.* **53** (1947), 1120.

[165] Vaughan, G., Elliston, B., and Taylor, T. T. I. GNU AUTOMAKE, AUTOCONF AND LIBTOOL.
`http://sources.redhat.com/autobook/`.

[166] Verhulst, P. Recherches mathematiques sur la loi d'accrossement de la population. *Memoirs de l'Academie Royal Bruxelles* **18** (1838), 1–38.

[167] Walters, P., Ed. *Symbolic dynamics and its applications* (July 1991), American Mathematical Society.

[168] Weisstein, E. ”Bogdanov Map.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/BogdanovMap.html`.

[169] Weisstein, E. ”Brusselator Equations.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/BrusselatorEquations.html`.

[170] Weisstein, E. ”Diffeomorphism.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/Diffeomorphism.html`.

[171] Weisstein, E. ”Dirac Equation.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/DiracEquation.html`.

[172] Weisstein, E. ”Dynamical System.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/DynamicalSystem.html`.

[173] Weisstein, E. ”Fast Fourier Transform.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/FastFourierTransform.html`.

[174] Weisstein, E. ”Fourier Transform.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/FourierTransform.html`.

[175] Weisstein, E. ”Gingerbreadman Map.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/GingerbreadmanMap.html`.

[176] Weisstein, E. ”Heat Conduction Equation.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/HeatConductionEquation.html`.

[177] Weisstein, E. ”Hénon Map.” From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/HenonMap.html`.

[178] Weisstein, E. ”Korteweg-de Vries Equation.” From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/Korteweg-deVriesEquation.html.

[179] Weisstein, E. ”Logistic Map.” From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/LogisticMap.html.

[180] Weisstein, E. ”lozi map.” From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/LoziMap.html.

[181] Weisstein, E. MathWorld.
http://mathworld.wolfram.com/.

[182] Weisstein, E. ”Maxwell Equations.” From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/MaxwellEquations.html.

[183] Weisstein, E. Neumann, John von (1903-1957).
http://scienceworld.wolfram.com/biography/NeumannJohnvon.html.

[184] Weisstein, E. ”Phase Space.” From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/PhaseSpace.html.

[185] Weisstein, E. Poincaré, Henri (1854-1912).
http://scienceworld.wolfram.com/biography/Poincare.html.

[186] Weisstein, E. ”Rössler Attractor.” From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/RoesslerAttractor.html.

[187] Weisstein, E. ”Schrödinger Equation.” From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/SchroedingerEquation.html.

[188] Weisstein, E. ”Sine Gordon Equation.” From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/Sine-GordonEquation.html.

[189] Weisstein, E. Three-Body Problem.
http://scienceworld.wolfram.com/physics/Three-BodyProblem.html.

[190] Weisstein, E. Turing, Alan (1912-1954).
http://scienceworld.wolfram.com/biography/Turing.html.

[191] Weisstein, E. "Wave Equation." From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/WaveEquation.html.

[192] Weisstein, E. "Web Diagram." From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/WebDiagram.html.

[193] Wikipedia. Alan Turing.
http://en.wikipedia.org/wiki/Alan_Turing/.

[194] Wikipedia. Dynamical system.
http://en.wikipedia.org/wiki/Dynamical_system.

[195] Wikipedia. John von Neumann.
http://en.wikipedia.org/wiki/John_von_Neumann/.

[196] Wikipedia. Logistic map.
http://en.wikipedia.org/wiki/Logistic_map.

[197] Wischert, W., Wunderlin, A., Pelster, A., Olivier, M., and Groslambert, J. Delay-Induced Instabilities in Nonlinear Feedback Systems. *Phys. Rev.* **E49** (1994), 203.

[198] Wolf, A., Swift, J., Swinney, H., and Vastano, J. Determining Lyapunov exponents from a time series. *Physica* **D16** (1985), 285–317.

[199] Xpp-aut.
http://www.math.pitt.edu/ bard/bardware/xpp/xpp.html.

[200] Yuan, G., Banerjee, S., Ott, E., and Yorke, J. A. Border-Collision Bifurcations in the Buck Converter. *Fundamental Theory and Application* **45** (1998), 707–716.

[201] Zheng, W.-M. Symbolic dynamics for the gap map. *Phys. Rev.* **A39**, 12 (1989), 6608–6610.

[202] Zheng, W.-M. Applied symbolic dynamics for the Lorenz-like map. *Phys. Rev.* **A42**, 4 (August 1990), 2076–2080.

[203] Zhusubaliyev, Z. T., and Mosekilde, E. *Bifurcations and Chaos in piecewise-smooth dynamical systems*, vol. **44** of *Nonlinear Science A*. World Scientific, 2003.

Index

Zusammenfassung

In dieser Ausarbeitung wird das Software-Projekt **AnT** vorgestellt, welches im Rahmen meiner Habilitation am Institut für Parallele und Verteilte Systeme an der Universität Stuttgart entstand. Ziel dieses Software-Projekts war die Entwicklung einer flexiblen, leicht wartbaren, plattformübergreifenden und offenen Simulations- und Analyse Software für nichtlineare dynamische Systeme. Dabei wurde von Anfang an ein großer Wert darauf gelegt, sowohl ein breites Spektrum von Klassen dynamischer Systeme zu unterstützen, als auch eine große Sammlung von Untersuchungsmethoden bereitzustellen. Daher wurde an vielen Stellen dem Einsatz generischer Konzepte Vorrang vor der Berücksichtigung reiner Performanzaspekte gegeben. Diese Entscheidung ist nicht zuletzt durch die rasante Entwicklung auf dem Gebiet der Hardware, sowohl was die Prozessoren als auch die Speicherkomponenten betrifft, begründet. Die geplanten Einsatzfelder der Software sind vor allem Ausbildung und Lehre sowie die Forschung. Die Arbeit ist folgendermaßen strukturiert:

In Kapitel eins (Introduction) wird die Arbeit motiviert und ein Überblick über die relevanten Themen und beteiligten wissenschaftlichen Disziplinen gegeben. Die Bedeutung der Begriffe Simulation, Analyse und mathematische Modellierung werden im Kontext der Arbeit präzisiert und diskutiert. Besonders hervorgehoben wird der Begriff des Simulationsprozesses, der Simulation und Analyse als einen in der Regel iterativ ablaufenden Prozess definiert und den Anwender beziehungsweise Designer des Systems als externe Prüfinstanz mit einbezieht. In der folgenden Abbildung 1 ist dieser Simulationsprozess schematisch dargestellt.

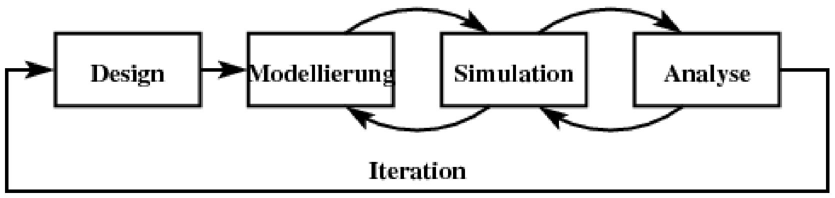


Abbildung 1: *Der Simulationsprozess (schematisch)*

Zusätzlich wird auf die interaktive und nicht-interaktive Simulation eingegangen. Im Rahmen des Software-Projekts **AnT** ist der Simulatorkern also der eigentliche Simulator als eine nicht interaktive Simulationsmaschine ausgelegt worden. Dies hat konzeptionelle Gründe die hauptsächlich im geplanten Einsatzgebiet der Software liegen. Der Simulatorkern wird im folgenden als AnT computation engine bezeichnet.

Das zweite Kapitel (The AnT project) stellt einen Überblick über die Ziele, Anforderungen, Randbedingungen und Eigenschaften des **AnT** Projekts dar. Es werden einige historische und technische Aspekte erläutert, sowie einige Grundprinzipien der Funktionalität beschrieben. Außerdem wird eine kurze Einführung in die Benutzung der Software und ihrer Fähigkeiten gegeben.

Im Kapitel drei (Supported classes of dynamical systems) werden die für das Verständnis der Arbeit notwendigen mathematischen Grundlagen kurz dargestellt. Insbesondere wird der Begriff eines dynamischen Systems definiert. Dies geschieht auf einem Abstraktionsniveau, welches es erlaubt, eine Vielzahl dynamischer Systeme unter einem einheitlichen und verallgemeinerten Gesichtspunkt zu betrachten. Daran orientierend werden die wichtigsten unterstützen Klassen dynamischer Systeme aufgeführt und ihre charakteristischen Eigenschaften im Kontext der Arbeit diskutiert, sowie repräsentative und illustrative Beispiele gegeben. Abschließend wird eine Klassifikation dynamischer Systeme auf Basis der erwähnten einheitlichen Sichtweise vorgenommen, die in der folgenden Tabelle 1 dargestellt ist.

Im vierten Kapitel (Scanning dynamical systems) wird der für die Untersuchung dynamischer Systeme wichtige Begriff eines scans eingeführt und seine Bedeutung auf dem Gebiet der nichtlinearen Dynamik betont. Die

Gedächtnis	Zustand	Indexierbarkeit	Zeit	Repräsentant
-	k	-	k	gewöhnliche DGL
-	k	-	d	Abbildungen
-	k	k	k	partielle DGL
-	k	k	d	*(nicht bekannt)*
-	k	d	k	gekoppelte gewöhnliche DGL
-	k	d	d	Gitter gekoppelter Abbildungen
-	d	-	k	diskrete Komponente hybrider gewöhnlicher DGL
-	d	-	d	endliche Automaten, Petri Netze, diskrete Komponente hybrider Abbildungen
-	d	k	k	*(nicht bekannt)*
-	d	k	d	*(nicht bekannt)*
-	d	d	k	*(nicht bekannt)*
-	d	d	d	zelluläre Automaten
+	k	-	k	zeitverzögerte DGL, Funktional-DGL
+	k	-	d	Rekurrente Abbildungen
+	k	k	k	partielle zeitverzögerte DGL, partielle Funktional-DGL
+	k	k	d	*(nicht bekannt)*
+	k	d	k	Gitter gekoppelter zeitverzögerter DGL
+	k	d	d	Gitter gekoppelter rekurrenter Abbildungen
+	d	-	k	diskrete Komponente hybrider zeitverzögerter DGL
+	d	-	d	Stapelautomaten diskrete Komponente hybrider rekurrenter Abbildungen
+	d	k	k	*(nicht bekannt)*
+	d	k	d	*(nicht bekannt)*
+	d	d	k	*(nicht bekannt)*
+	d	d	d	zelluläre Automaten mit Gedächtniseffekten

Tabelle 1: *Einige Klassen dynamischer Systeme*
Legende*: '*k*' - kontinuierlich, '*d*' - diskret, '*+*' - vorhanden, '*-*' nicht vorhanden*

verschiedenen Möglichkeiten scans durchzuführen, sowie die Begriffe scan item und scan item sequence werden erklärt. Direkt damit verbunden ist die Analyse des Verhaltens dynamischer Systeme in Abhängigkeit von bestimmten Einflussgrößen wie beispielsweise Systemparameter, Anfangswerte oder gar Methodenparameter. Dabei wird die folgende Sichtweise eingenommen: Unter einem scan versteht man die Untersuchung bestimmter Eigenschaften eines dynamischen Systems unter Variation einer oder mehrerer Einflussgrößen. Die Fähigkeit der **AnT** Software, scans sehr flexibel durchführen zu können, ist eine ihrer charakteristischen Stärken. Konzeptionsbedingt können fast beliebige scans durchgeführt werden und zwar sowohl eindimensional als auch mehrdimensional. Die Anzahl der zu variierenden Einflussgrößen ist dabei eigentlich nur durch Performanzaspekte beschränkt. Vorgestellt werden auch die verschiedenen implementierten Varianten scans durchzuführen.

Im Kapitel fünf (Supported investigation methods) werden die implementierten Untersuchungsmethoden detailliert beschrieben. Die Untersuchung des Verhaltens dynamischer Systeme ist die wichtigste Aufgabe auf dem Gebiet der nichtlinearen Dynamik, weil sie die Basis sowohl für qualitative als auch quantitative Aussagen, Vergleiche und Vorhersagen darstellt, sowie eine gezielte Verbesserung oder Veränderung der zugrundeliegenden mathematischen Modelle ermöglicht. Daher werden in diesem Kapitel einige Beispiele vorgestellt, anhand derer die Anwendung der implementierten Untersuchungsmethoden illustriert wird. Die folgende Liste zeigt die aktuell implementierten Untersuchungsmethoden:

- Allgemeine Trajektorienauswertung
- Periodenanalyse
- Berechnung der Lyapunov Exponenten
- Regionenanalyse
- Dimensionsanalyse
- Frequenzanalyse
- Hauptkomponentenanalyse
- Konditionsauswertung
- Symbolische Sequenzanalyse
- Analyse der Symbolischen Abbildung
- Berechnung verallgemeinerter Poincaré Schnitte

Was die Einzelheiten der Untersuchungsmethoden betrifft, so können diese

im Rahmen dieser Arbeit weder detailliert dargestellt werden, noch kann auf die mathematischen Grundlagen auf denen sie basieren genauer eingegangen werden. Es wird stattdessen ein pragmatischer Zugang gewählt, der es Anwendern die die Methoden bereits kennen ermöglicht, diese bei der Untersuchung der sie interessierenden dynamischen Systemen einzusetzen.

Im sechsten Kapitel (Simulating and investigating dynamical systems) werden generische Architekturkonzepte für die Simulation und Analyse zeitabhängiger Prozesse vorgestellt. Insbesondere werden die verwendeten Basiskonzepte und die darauf aufbauenden Strukturen eingeführt. Aus softwaretechnischer Sicht können diese Konzepte als Entwurfsmuster für die Simulation und die Analyse dynamischer Systeme angesehen werden. Sie stellen gewissermaßen eine Basisfunktionalität bereit auf der nach einem Baukastenprinzip die gesamte Architektur der Software beruht. Aus Sicht der Programmiersprachen, stellen die erwähnten Entwurfsmuster die grundlegenden Strukturelemente einer abstrakten Programmiersprache für die Simulation und die Analyse dynamischer Systeme dar. Der bereits erwähnte Schritt, dynamische Systeme auf einem bestimmten Abstraktionsniveau unter einem einheitlichen und verallgemeinerten Gesichtspunkt zu betrachten, stellt die notwendige Voraussetzung für die Anwendung der erwähnten generischen Simulationskonzepte dar. Die Tatsache, dass ein dynamisches System als ein iterativer bzw. zyklischer Prozess angesehen werden kann, der sukzessive einen Zustand $z(t)$ zum Zeitpunkt t in einen zeitlich darauffolgenden Zustand $z(t + \Delta t)$ abbildet, hat zwei wesentliche Designaspekte zur Folge. Zum einen legt es die Verwendung einer abstrakten Transition (abstract transition) nahe und zum anderen die eines abstrakten Iterators (abstract iterator), der in Abbildung 2 schematisch dargestellt ist. Die Transition kann beispielsweise verwendet werden, um den beschriebenen Zustandsübergang zu realisieren, während der Iterator verwendet wird um den Prozesscharakter softwaretechnisch abzubilden. Dabei spielt es keine Rolle, ob die zeitliche Änderung Δt in endlichen Zeitschritten erfolgt oder in infinitesimal kleinen, da die infinitesimale Änderung auf jeder digitalen Rechnerarchitektur sowieso diskretisiert werden muss, was letztlich dazu führt, dass auch zeitkontinuierliche dynamische Systeme aus softwaretechnischer Sicht als zeitdiskrete Systeme behandelt werden müssen. Die von einer Simulation geforderte wiederholte Anwendung des Zustandsübergangs legt ein weiteres Konzept nahe, das man als Maschinenkonzept bezeichnen kann. Sowohl ein einzelner Simulationslauf, als auch ein kompletter scan

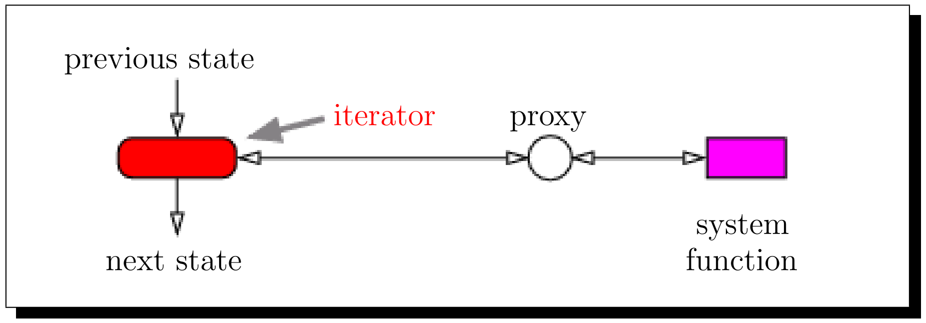


Abbildung 2: *Der Iterator (schematisch)*

sind softwaretechnisch über entsprechende Maschinen, nämlich die Iterator Maschine (iter machine) und die Scan Maschine (scan machine) realisiert. Die beiden Maschinen sind in den Abbildungen 3 und 4 schematisch dargestellt. Die Abbildungen verdeutlichen einerseits, wie die einzelnen Konzepte zusammenwirken und andererseits wie durch die Verwendung abstrakter Klassen die Wiederverwendung von Komponenten ermöglicht wird.

Im Kapitel sieben (Distributed computing) wird die Notwendigkeit und die große Bedeutung von verteiltem Rechnen für die Simulation und Analyse dynamischer Systeme begründet, sowie die Fähigkeiten der entwickelten Software in diesem Zusammenhang vorgestellt. Es wird die verwendete Client/Server Architektur und das speziell hierfür entwickelte Netzwerk-Protokoll (AnP) erklärt. Die Fähigkeit, bestimmte zeitaufwändige Berechnungen auf mehrere Rechenknoten zu verteilen ist eine weitere Stärke des entwickelten Softwarepakets. Insbesondere hochaufgelöste ein- oder mehrdimensionale scans sind ideal parallelisierbar, da die entsprechend der selektierten Untersuchungsmethoden durchzuführenden Berechnungsschritte unabhängig voneinander sind. Sie können beispielsweise auf einem Cluster sehr elegant in vertretbarer Zeit durchgeführt werden. Die Berechnungen können allerdings auch auf verschiedenartigen Rechenknoten und unter verschiedenen Betriebssystemen (Linux, Solaris, Windows) ausgeführt werden. Essenziell ist lediglich die Vernetzung der einzelnen Knoten wobei das dem entwickelten Netzwerk-Protokoll das TCP/IP-Protokoll zugrunde liegt. Der Server ist der einzige Knoten, dem die konkrete Aufgabenstellung, das heißt die Initialisierungsdatei zur Verfügung gestellt wird. Die einzelnen Clients erhalten die Information über die durchzuführenden Berechnungen direkt

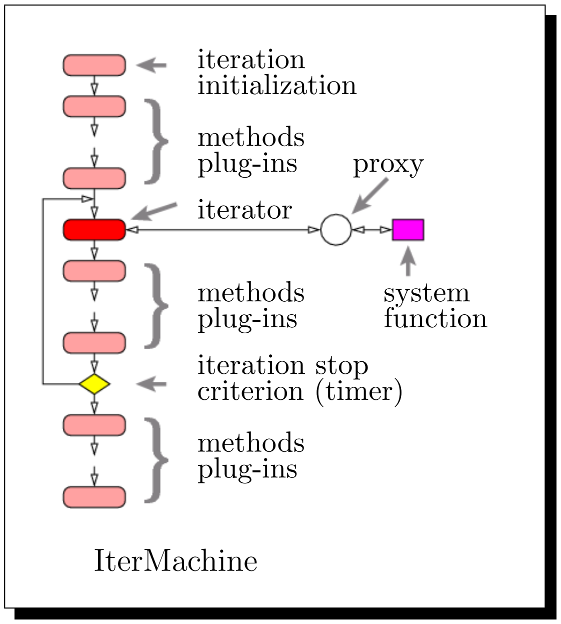


Abbildung 3: *Die Iterator Maschine (schematisch)*

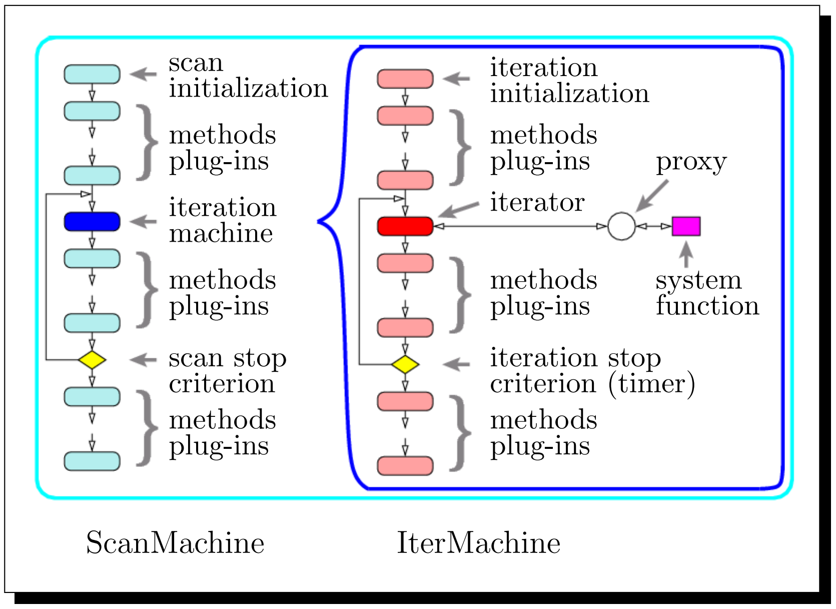


Abbildung 4: *Die Scan Maschine (schematisch)*

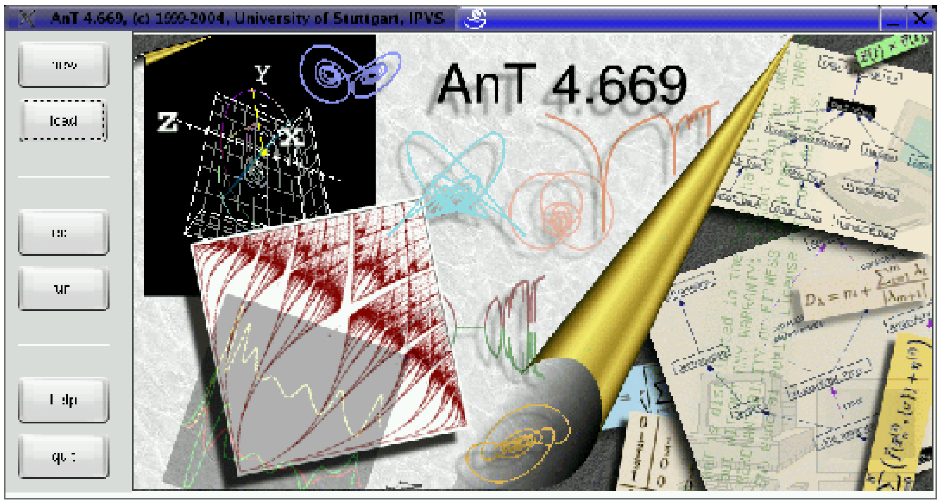


Abbildung 5: *Das Hauptfenster der grafischen Benutzeroberfläche AnT-gui*

Man kann von Grund auf mit einer völlig neuen Initialisierung starten oder eine bereits existierende Initialisierungsdatei laden und bearbeiten. Nach der Bearbeitung kann die neue Initialisierung gesichert und ein Simulationslauf direkt gestartet werden.

vom Server und benötigen daher nur die Information welcher Knoten der Server ist und welches System behandelt wird. Die Verteilung der einzelnen scan Punkte kann auf zwei verschiedene Weisen erfolgen. Zum einen kann eine beim Starten eines Clients fest eingestellte Anzahl von scan Punkten vergeben werden und zum anderen kann eine Zeitspanne vorgegeben werden, die einem Client für die Bearbeitung der scan Punkte zur Verfügung steht, bevor er erneut mit dem Server für die Übermittlung weiterer scan Punkte Verbindung aufnimmt. Unter einem scan Punkt versteht man beispielsweise bei einem Parameterscan, die Anwendung der in der Initialisierungsphase des Servers festgelegten Untersuchungsmethoden bei einem einzigen Wert des Parameters.

Kapitel acht (AnT-gui: the graphical user interface) befasst sich mit der grafischen Benutzeroberfläche der Simulationssoftware (siehe Abb. 5). Es werden die vier Initialisierungsschritte der AnT computation engine vorgestellt und der Zusammenhang zwischen ihnen erklärt. Es wird außerdem auf die vielseitigen Möglichkeiten des Initialisierungsprozesses eingegangen. Mit Hilfe der grafischen Benutzeroberfläche können Anwendern die Software relativ einfach für ihre Aufgaben- und Problemstellungen einsetzen. Aus softwaretechnischer Sicht ist hier besonders hervorzuheben, dass die Benutzeroberfläche selbst fast vollständig automatisch generiert wird und zwar auf Basis der Informationen einer zentralen Datei des Simulationspakets. Diese Datei enthält die gesamte Information über die vernetzten Zusammenhänge der Steuerungsmechansimen und Konfigurationen und wird auf der einen Seite von der AnT computation engine selbst verwendet um die Einstellungen eines Anwenders bei einem Simulationslauf auf Vollständigkeit und Konsistenz hin zu überprüfen. Auf der anderen Seite wird die gleiche Information verwendet und die grafische Benutzeroberfläche mit den notwendigen Eingabefeldern zu versehen und Abhängigkeiten automatisch aufzulösen bzw. auf unvollständige oder inkonsistente Eingaben aufmerksam zu machen.

Das Kapitel neun (Visualization of dynamical systems) befasst sich mit den direkten grafischen Ausgabemöglichkeiten der Software. Diese Animationen basieren auf der OpenGL Bibliothek und sind optional. Das heißt, dass Benutzer, die über diese Bibliothek nicht verfügen, diese Funktionalität zwar nicht verwenden können, dafür aber alle anderen vorgestellten Funktionalitäten. Die von der AnT computation engine erzeugten Ausgabedateien liegen im ASCII-Format vor und können daher mit allen gängigen Visualisierungswerkzeugen wie beispielsweise dem weit verbreiteten Programm gnuplot dargestellt werden.

Im zehnten Kapitel (Numerical aspects of simulation) werden einige Bemerkungen zu den numerischen Aspekten der Simulation gemacht und insbesondere auf das Auftreten numerischer Artefakte hingewiesen. Dieses Kapitel umreißt lediglich einen ganz kleinen Teil der numerischen Schwierigkeiten die bei der Simulation und Analyse dynamischer Systeme auftreten können. Generell kommen hier einerseits die üblichen und bekannten Effekte der Gleitpunktarithmetik zum tragen und andererseits spezielle Probleme und Phänomene die sehr eng mit bestimmten charakteristischen Eigenschaften der untersuchten dynamischen Systeme zusammenhängen.

Im Kapitel elf (Examples) werden einige weiterführende Beispiele ausführlich behandelt, die die Fähigkeiten der Software illustrieren. Insbesondere werden hier anhand der konkreter Beispiele fast alle implementierten Untersuchungsmethoden präsentiert. Außerdem werden ein- und mehrdimensionale scans vorgestellt und zwar nicht nur übliche scans nach den

Systemparametern sondern auch nach den Anfangswerten und sogar nach einigen Methodenparametern. Bei der Auswahl der Beispiele wurde explizit darauf geachtet, dass möglichst viele der unterstützten Systemklassen zum Einsatz kommen.

Die Arbeit wird abgeschlossen mit einem kurzen Ausblick im Kapitel zwölf (Conclusion). Hier werden die Hauptvorteile der Simulations- und Analysesoftware kurz zusammengefasst und basierend auf einem kleinen Überblick über ähnliche Software Projekte auf zukünftige Erweiterungsmöglichkeiten bzw. Entwicklungsrichtungen hingewiesen.