Institute of Cognitive Science
Neurocybernetics

Visualization of Neurodynamical Properties for an Evolutionary Robotics Environment

Bachelor-Thesis

November 17, 2010

Author: Till Faber
First Supervisor: M. Sc. Christian Rempis
Second Supervisor: Prof. Dr. Frank Pasemann

Abstract

Modularization of neural networks is one approach to help understand the function of neural networks. These modules can exhibit interesting dynamical behaviors, ranging from fixed point behavior to chaos. In this thesis a software tool is introduced, designed to analyze these behaviors graphically with the support of the existent NERD software. Therefore, the computational processes for the applications are described and an exemplary usage is presented after giving the necessary background in dynamical system theory and artificial neural networks.

Contents

1 Introduction

Understanding the principles of neural networks is a task which is as satisfiable as it is compelling. Single neurons can exhibit stunning behaviors and render large networks overwhelmingly complex. How can cognitive scientists cope with brains consisting of one hundred billion neurons and one hundred trillion synapses or more? One approach to help answering this question is to reduce the complexity to levels at which more information about networks can be obtained. Therefore smaller subnetworks are examined and the relation between their structure and behavior is studied. These sub-networks are considered as functional modules of the entire structure.

In this thesis the Dynamics Plotter is introduced as a tool to help understand network modules by analyzing their behavior graphically under various circumstances. My goal was to design a program that, embedded in the Neurodynamics and Evolutionary Robotics Development Kit (NERD), sets a high value on simple handling and extensibility. Without much effort, networks can be created and plots be generated to investigate the network behavior. Implementation of plots for further applications can be added without changing the code within the remainder of the program.

I was prompted by the question stated above and the demand this puts on science. Further, I was interested in the issue how graphical analysis of dynamical behavior of neural networks supports the understanding of these. This thesis is strongly motivated by these aspects.

At present, there exists a variety of tools for plotting dynamical behavior[1]. The value of this program lies in its integration to NERD and the specialization on artificial neural networks. Like NERD, the Dynamics Plotter is an open source program and thus extensible for purposes that are not implemented to this point. It benefits from the existent neuron and synapse models integrated into NERD and from the feature that these can be easily replaced, when needed. Hence, the application of the Dynamics Plotter to artificial neural networks is not limited to a certain neuron prototype.

[1]See for example [Nusse and Yorke, 1994] and [Ermentrout, 1990] for descriptions of software for the visualization of dynamics.

Nevertheless, it is immediately applicable due to the available models. The strong focus on neural networks limits the use to them. In contrast to most existing dynamics visualizers, it is not possible to analyze the behavior of, for example, nonlinear systems given as a system of equations.

In the following thesis I attempt to present the Dynamics Plotter in a compact manner. In section 2 the necessary preliminaries are described to understand the functioning of Dynamics Plotter. The concepts of dynamical systems, artificial neural networks are therefore summarized. Additionally, a short introduction to NERD is given. In the subsequent section 3 the design of the program is explained. Its modularization in computing and visualizing components is presented with a focus on the existent data computations. The Dynamics Plotter is applied to give an example how to handle it and show the results in section 4. Finally, the conclusion constitutes the last section by briefly summing up the core information and giving a short prospect.

2 Preliminaries

As the understanding of the Dynamics Plotter presupposes basic knowledge of dynamical systems, neural networks and the NERD-software in which it is embedded, I will give a short introduction to those topics in the following sections.

2.1 Dynamical Systems

To gain an insight into the comprehensive field of dynamical systems I will first explain what is generally perceived to be a dynamical system and further outline its contents and properties. Subsequently I examine the different kinds of settling behavior one can observe.

Roughly, dynamical systems consist of a state and some rules which describe the change of that state over time. More specifically it is a mapping $f(x,t)$ with $x \in M$. x is the *state* of the system, and M the *phase space*. The state gives the system's variable values (e.g. position, velocity and acceleration of a ball), it is thus a vector $x \in \mathbb{R}^n$ with n being the number of the system's variables. The phase space is the set of all possible states the system can have and is of dimension n. t is the time variable and can be either element of the integers or of the real numbers. If the time is discrete, that is $t \in \mathbb{Z}$, a dynamical system is called a *(time) discrete dynamical system*, in the case of continuous time, $t \in \mathbb{R}$, one speaks of a *(time) continuous dynamical system*. In the following only time discrete systems will be considered, as they are used with the NERD software[2]. The mapping f in discrete time is usually represented by difference equations of the form $x(t+1) = f(x(t))$. Taken together it holds that $f : M \times \mathbb{Z} \to M$. [Osipenko, 2007, p. 9f]

Generally dynamical systems are considered to be deterministic. Thus, there is no immanent random component and in principle one could predict the future behavior. As a lot of dynamical systems are nonlinear it is nonetheless often impossible to predict distant future states, even under the assumption of perfect knowledge, as I will point out later.

[2]See [Strogatz, 2000] for more on time continuous dynamical systems.

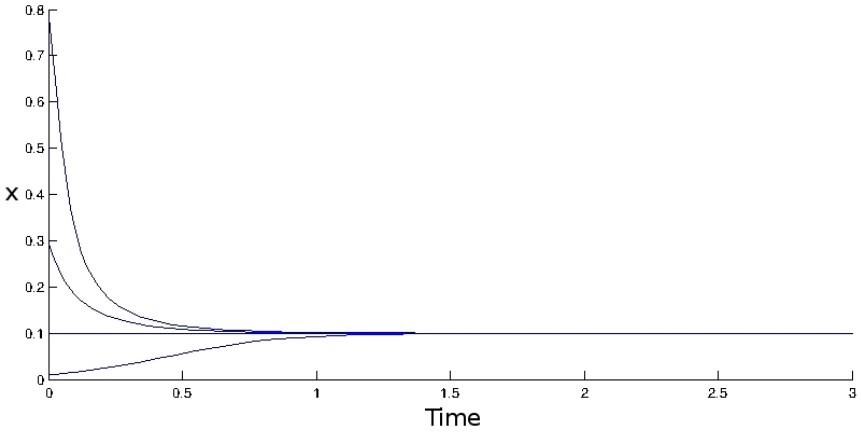

Figure 1: Transients and attracting fixed point of a one-dimensional system

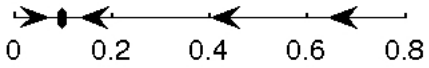

Figure 2: Phase space of a one-dimensional system

Considering the long-term behavior of a system frequently is of interest, one can wait for the system to converge to a persisting behavior. Therefore, over time the system moves through different states. The trace of these states is called a *trajectory*.

At some point in time the behavior of the system may settle into a, to some extent, orderly behavior. I will give an overview of the different long-term behaviors in a moment. That part of the trajectory, which is not yet showing the final behavior is called a *transient*. Transient behavior also occurs, when the system is perturbed after it has converged.

Figure 1 shows an example of the transients of an one-dimensional system with the variable state plotted against the time. This is not an image of the phase space, which would be a one-dimensional plot (see figure 2 for the phase space of the same system).

Some long-term behaviors have an attracting effect. In the long run transients therefore converge to these behaviors. All different kinds of behaviors can be attracting and those which are, are called *attractors*. The set of initial

states, from which the system will converge in an attractor, is called *basins of attraction* of the respective attractor. More than one attractor can exist in every dynamical system and hence can basins of attraction. The behavior the system settles in, therefore depends on the initial state.

The settling behaviors split in several sub-types. A partitioning into the rough kind of long-term behavior thus seems reasonable. The asymptotic behaviors can be categorized into *fixed point*, *periodic*, *quasi-periodic* and finally *chaotic* behavior.

Fixed Points

Fixed point behavior is characterized by that the system remains (if it is not disturbed) in one state. This state is called the fixed point, x^*, such that $f(x^*(t+1)) = f(x^*(t)) = x^*$.[Strogatz, 2000, p. 19, p. 349]

Fixed points can be further divided according to the behavior of the trajectories that start nearby. A fixed point x^* is called *attracting* if all trajectories starting close to x^* converge in x^* at some point in time. A system being in an attracting fixed point will return to it, if it is slightly disturbed. Figure 1 shows an example of an attracting fixed point at $x = 0.1$. In figure 2 the fixed point is shown as a black oval at 0.1. The arrows denote the direction of the state's movement and it is clear, that systems in a state with lower as well as with higher values than 0.1 are driven towards the fixed point. Contrary, *repelling* fixed points do not attract trajectories, but repulse them. Only states at exactly the same point remain there at rest. The image of a ball balanced on the tip of a hill can be considered as example for this process. As soon as the ball is slightly pushed it will move away. A hybrid of both attracting and repelling fixed points is the *saddle-point*, which attracts trajectories from some directions and repels in other directions.[Strogatz, 2000, p. 17, p. 128]

A further fixed point property is that of the *Liapunov stability*. It denotes that all trajectories close to a Liapunov stable point stay close to it for all time.[Strogatz, 2000, p. 129] Cases of fixed points which are Liapunov stable but not attracting are called *neutrally stable*.[Strogatz, 2000, p. 134]

Fixed points, which are both Liapunov stable and attracting are usually referred to as *stable* or *asymptotically stable*. For an illustration of this condition, a ball coming to rest in the deepest point of a valley can be considered.

Unstable fixed points are considered to be neither attracting nor Liapunov stable.[Strogatz, 2000, p. 129]

Periodicity

The time it takes until a system returns to a state is called its *period*. In dynamical systems behaviors with a large range of periods are abundant. To stress the ball example once more, a ball in a frictionless world, oscillating between the sides of a half-pipe would demonstrate periodic behavior. Hence, periodic behavior obviously differs from fixed point behavior through its motion. While fixed points define the halt of a system, periodicity implies changing of the system's state in a repetitive manner.

Mathematically, periodicity is given as $x(t + T) = x(t)$ or equivalently applying the mapping $f(x, t)$ T times to $x(t)$, such that $f^T(x(t)) = x(t)$ with a period T.

Quasi-Periodicity

The term 'quasi-periodicity' already suggests the similarity to periodicity, to which it the quasi-periodic behavior differs in that the trajectory never closes. Thus it will never run through a point twice, but it will instead come arbitrarily close to it in some point in time.

Quasi-periodicity occurs only with systems like coupled oscillators, whose phase space is a torus. The periods of such oscillators are of such numbers, that their division results in an irrational number. Hence, it is impossible that both return to the initial state at the same time. [Strogatz, 2000, p. 276]

Chaos

A survey of the notion of chaos is not a simple task, as there is no undisputed definition of it.[Strogatz, 2000, p. 323] Nevertheless, for the purpose of an

introduction, naming the most typical properties of chaotic behavior should suffice.

Chaotic behavior is usually associated with unpredictability. Therefore, a chaotic system is not periodic, since if it returned to a state it had been to before, it would repeat its behavior due the determinism of the system. At least in most models of dynamical systems, noise or randomness is not included. Thus, with perfect knowledge of the current state and the map, it is theoretically possible, to predict the future states of the system. Practically however, due to unsolvable nonlinear equations of the system and limitations in computer simulations only short-term predictions are possible. Indeed every error, generated for example by rounding, grows exponentially. More specifically, all trajectories, starting no matter how close together, will quickly drift apart and lose all similarity. As a result, the path of the trajectory strongly depends on the initial system state.[Strogatz, 2000, p. 323f]

Although chaotic behavior is unpredictable, it is often is nonetheless limited to some region in the state space. This region can be attracting and is called a 'strange attractor'.

To conclude this section, it remains to examine what happens if a dynamical system changes. Generally by varying parameters of the system, its attractor 'landscape' changes. Thus new attractors can be created or existing ones disappear. Changes in periodicity can occur as well. These changes are known as *bifurcations*. A detailed description of these can be found in section 3.2.1.

An interesting incident is the appearance of a *hysteresis*. In some cases, varying a parameter leads to a sudden jump in the system state from one attractor to another. Reversing the parameter change does surprisingly not necessarily lead to a return to the previous attractor. Only further continuation of the reversal beyond the parameter value that led to the jump eventually induces a changing to the initial state. [Strogatz, 2000, p. 60]

2.2 Artificial Neural Networks

In this section I will give a compact recapitulation of the concepts of artificial neural networks (ANNs) and their constituents. For that purpose I will explain the topic on a basic, but sufficient level to understand the subsequent chapters of this paper.

In principle, ANNs consist of basic computational units (*neurons*) which are interconnected and influence one another. A neuron i calculates its output depending on the input it receives from other neurons. The connection (*synapse*) from one neuron i to another neuron j can be weighted, such that one neuron has a large influence on the activity of one neuron and simultaneously a very weak on another even though there exist a connection w_{ji}. Moreover, in some models the connection can also be negated, having the opposite effect to one neuron as to another. Synapses can be either symmetric or asymmetric, such that, if symmetric, $w_{ji} = wij$. Further, a neuron can be influenced by a constant factor, the bias b_i, which is added to the synaptic input, independent from other neurons. Thus the input u_i of neuron i looks like

$$u_i = \sum_{j=1}^{m} w_{ij} \cdot x_j + b_i$$

with m being the number of neurons that have a synapse to i and x_j the output of neuron j. The output is then calculated by

$$x_i = \varphi(u_i)$$

with $\varphi(\cdot)$ being the transfer function, for example a step function, a sigmoid or the hyperbolic tangent. In the following, I will concentrate on the hyperbolic tangent as the activation function, as it is mostly used with the NERD neurons. The output obtained above is then used for the calculation of all neurons i is connected to. The calculation can be synchronous, such that all neuron outputs are calculated at once, using the input from the last time step. Alternatively, the output is updated asynchronously, by calculating the outputs one by one with the newest input the neuron received.[Haykin, 1998, p. 33]

Hyperbolic tangent transfer (tanh) functions have several important properties, such as the output being limited to some range, for example from -1 to 1. Additionally, the slope of tanh functions are approximately linear near the point of origin. Further a tanh tranfser function renders a system to be *dissipative*, thus the activations would decay over time. This is often countered by the bias terms which constantly add to the inputs.

Neural networks are often classified into recurrent or feedforward networks. Feedforward networks do not allow recurrent synapses. As reccurrence is a necessity for complex dynamical behavior I will only refer to those in this paper. The next section deals with how recurrence leads to dynamical behavior.

Additionally, the neurons of a network can be grouped according to their role. Thus, some neurons receive input from outside the network, for example from sensors or other networks. Those are known as 'input neurons'. Neurons sending signals outside of the network are accordingly called 'output neurons' and those with synapses only to other neurons within the network are named 'hidden' neurons.

When treating neural networks such as the brain, one has to deal with a very high complexity due to the great amount of neurons and even greater number of connections. One approach to this is to modularize a large network into functional sub networks.[Pasemann et al., 2001] Many of these sub systems can be controlled by influencing one input neuron and it could thus be of interest to study their behavior dependent on this single parameter.

Recurrent networks with nonlinear transfer functions can show highly complex behavior, namely all behaviors of a complex dynamical system. Thus, one is able to apply concepts of dynamical system theory to help understand more of this most complex matter.[Cessac and Samuelides, 2007]

2.3 The NERD environment

The Neurodynamics and Evolutionary Robotics Development Kit (NERD)[3] provides the environment for the Dynamics Plotter and in the following I will give a short overview of the program with a focus on the parts that are used with the Dynamics Plotter.

NERD is an open source program written in C++ with an object-oriented approach, while using Qt 4.5[4] for graphical user interface purposes and general platform independence. It serves as a tool for neuro-evolution and simulation of robots and artificial recurrent neural networks and is expandable to approach a large variety of problems.[Rempis et al., 2010]

As the Dynamics Plotter is concerned with artificial neural networks, the following gives an idea how NERD enables the user to examine these elaborately. With NERD's neural network library the user has a model and an editor for neural networks at hand.

Among other things, the neural network model includes prototypes for neurons and synapses, it can however be easily augmented by plug-ins. The available neurons can use different transfer functions such as hyperbolic tangents, parametrized sigmoids or functions adjusted to hardware applications. Additionally, the way the input is calculated can be chosen. It is further possible to change and read the neuron activity, output and bias. Neurons can be grouped or gathered in neuro-modules, which then can be saved and loaded separately and used for evolutionary algorithms. For that neurons can be signed as input or output neurons. Synapses can be assigned a weight and can be modulated to fit to their applications. NERD updates neuron synchronously and time discrete.

With help of the network editor, artificial neural networks can be edited, saved and loaded. Figure 3 shows a screenshot of the editor, loaded with a 2-neuron network. All neurons and synapses can be edited with the properties visible on the right side of the window.

During the execution of the network, the network editor can visualize the

[3] `http://ikw.uni-osnabrueck.de/~neurokybernetik/tools/nerd` - accessed November 14, 2010

[4] `http://qt.nokia.com/` - accessed November 14, 2010

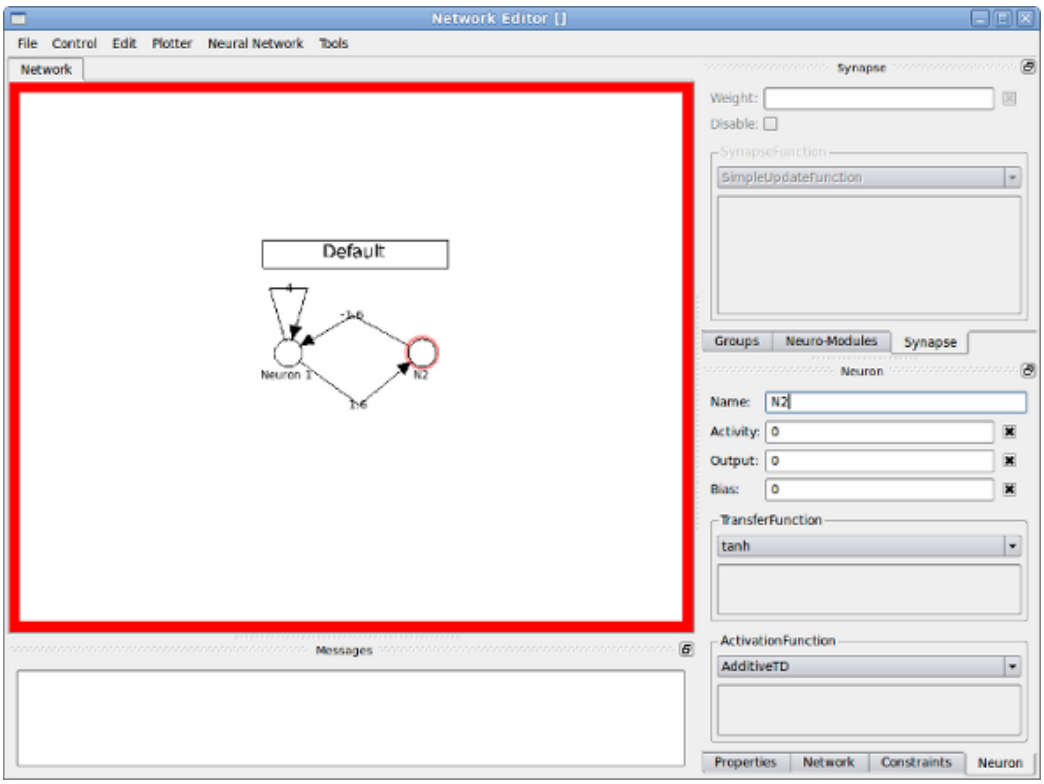

Figure 3: The NERD network editor

neuron activities by coloring the neurons. Additionally plotters can be used to show the neuron activities and outputs.

All network elements (neurons, synapses) have unique IDs which can be easily obtained with the network editor. These are of help, when the user wants to address a certain neuron or synapse.

NERD provides another tool which is important for the use of the Dynamics Plotter. For specifying properties, such as choosing which neurons shall be observed, the user can exploit the property panel. This graphical interface lists the properties that can be set and which then can be edited before execution. The property panel can also give feedback to the user, for example the elapsed time during a calculation.

3 Program Design

In the following, the design and structure of the program are depicted. I will first give an overview of the entire program, followed by a presentation of the individual constituents.

3.1 Scheme

As mentioned above, the Dynamics Plotter is an extension of the NERD software. Just as NERD, the extension is mainly written in C++ code, while using the Qt framework for the graphical user interface. It has a modular structure, consisting of largely independent parts. Thus, it is possible to add new components with minimal effort. For example, more diagram plotter types can be created and added to the program without changing those parts of the program, which are concerned with visualization of the data.

The entire program consists of three general parts. First, for the creation of the output data the *computation modules* use the NERD software to simulate the neural network behavior and generate a data matrix. They are specific to the field of application, the Dynamics Plotter is used for. Second, either an *exporter* or the *online plotter* is employed. Exporters create files readable of specific external programs, such as Matlab[5]. The online plotter creates a new window to plot the data directly within the NERD environment. *External scripts* and programs constitute the last part. Such scripts are able to read the respective output file and plot the data in a more powerful way.

I will give a more detailed description of the creation process of a diagram in section 4. For now, a shorter overview should suffice.

To begin, the user loads or creates a network with support of the network editor and decides, according to the plot that shall be created, which elements of the network are varied and observed. In the object property panel the properties values for the plotting algorithm are set (more on the properties in a moment) and the process is started. The particular computation module

[5] `http://www.mathworks.com/products/matlab/` - accessed November 15, 2010

will create the data in form of a matrix. The matrix entries hold the values for the respective diagram pixels. Additionally, the name of the employed computation module and the axes descriptions are saved to the matrix. The matrix is then either exported and can be plotted by, for instance, Matlab, or is illustrated by the internal online plotter. Moreover, it is possible to start the plotting during the data computation to observe the process of the calculation.

3.2 Computation

Due to the modular design, the actual applications differ in only one component of the Dynamics Plotter, namely the particular computation modules. The modules are specifically constructed for their respective tasks, whereas the other components can interact with all modules and are independent of the field of application.

Adjusted to their applications, the computation modules use specific methods of computation and require manual input to set various properties besides the network that is examined. The properties are partly shared between the modules, such as the axes descriptions, or can be specific. I will present the specific properties when covering the particular plots.

Properties, that are common among all plotter types are, for example, the width and height of the diagram. Both are given in pixels and determine, according to the computation module, either the accuracy of the output, the parameter step size or both. Thus, if for instance the parameter changes are plotted on the x-axis and the output neuron activity on the y-axis, the width of the plot determines how many parameters steps are taken, while the height gives the numbers of 'buckets', the output is discretized to. Every bucket represents a certain range of the output and the respective matrix entry will be updated when activity is found in that specific range. Additionally minima and maxima for the parameter and output range need to be set. The bucket and step sizes are calculated by dividing the ranges by the respective pixel number.

The user can further specify output neurons and network elements that

are varied. In some cases, varied elements are restricted to neuron biases, while with other modules it is possible and reasonable to modify synapse strengths.

Another property which is needed for most plotters is the 'tolerance', which specifies the accuracy of the calculation. It determines how small the difference between two values is until they are treated as the same. This is needed for the search for attractors, when a network state is compared to states, the network has been in before. If two states are found to be identical, an attractor must be found, due to the determinism of the system. That way, a larger tolerance can increase the speed of the calculation, but possibly generate faulty results.

The time that is elapsed by the computation process depends on the number of parameter steps and the steps it takes until an attractor is found. If none can be found, the algorithm takes the specified maxmimal number of steps. Additionally pre-steps can be used to get rid of small activities at the beginning, which may result in a faulty output. Taken together, in the worst case the algorithm takes a total of [parameter steps] $\times$ ([presteps] + [maximal number of steps]) steps. In the best case, the maximal number of steps is not reach, but an attractor is found in the first step. When two parameters are varied, the number of steps in the worst case is [parameter 1 steps] $\times$ [parameter 2 steps] $\times$ ([presteps] + [maximal number of steps]).

With the input received from the user, the neural network is parameterized and triggered and the calculation is started. The maintained results are inserted into an output matrix, which is then used by the visualizing components. As mentioned above, the matrix holds the respective pixel values, which are later translated into colors by the plotters. Hence the matrix is either two or three dimensional, depending of the type of the plot.

In the subsequent subsections I will describe the existing computation modules in more detail.

3.2.1 Bifurcation Diagram

As mentioned before, dynamical systems can exhibit a variety of behaviors. To analyze the connection between the parameters and the behavior of a system, it is helpful to record the long-term behavior for a series of parameter values. Thus a bifurcation diagram plots the output of system against the parameter settings. Due to the constriction to plot only the final behavior, the information about the system's transient is lost. A bifurcation diagram is largely constrained to the qualitative behavior, in that it shows the attractor types and values. Every black dot in the plot indicates that the respective output activation value (y-axis) is part of the attractor. That means that the system moves through this state, the frequency depending on the period of the attractor. If no attractor is found all visited states are plotted, but if one is found, the transient part is removed and only the remaining states are shown. See figure 4 for an exemplary bifurcation diagram. Here some parameter r is varied on the x-axis from 2 to 3.2 and the output is shown on the y-axis.

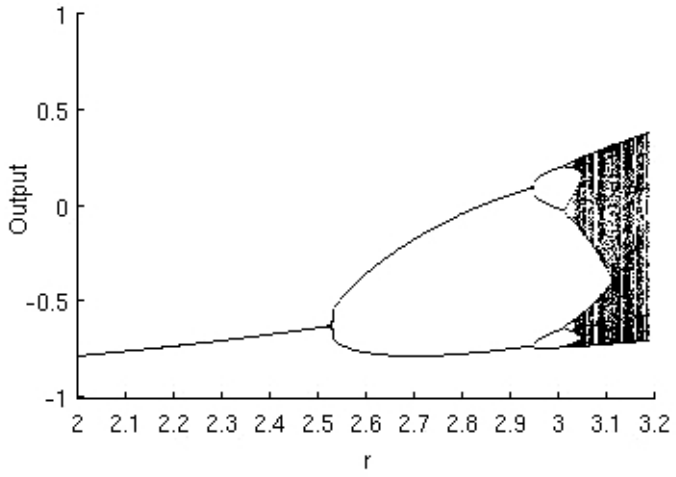

Figure 4: Bifurcation diagram

Often it is of interest, at what point changes in the behavior occur. At some parameter values, a small change in the parameter dramatically changes the system's qualitative behavior. Such a rapid change, for example from a

fix-point attractor to period-2 oscillator, is called a *bifurcation*. The parameter values at which it happens are called *bifurcation points*.[Strogatz, 2000, p. 44] Figure 4 shows an abundance of bifurcations, one occurs, for example, at about $r \approx 2.53$. At this bifurcation point a fixed point transitions into a period-2 oscillator. Thus, in this diagram parameter ranges are identifiable in which the system stays in one state ($r = 2$ to $r \approx 2.53$) and where it oscillates with period 2 ($r \approx 2.53$ to about $r \approx 2.95$). Then the period increases to four, then to eight, etc. Accordingly, this increase of the period during the change of a parameter is called *period doubling bifurcation*.[Devaney, 1990, p. 63] Finally, the high periods switch to very high periods or non-periodic behavior (around $r = 3.05$).

A bifurcation diagram does not help to discern chaotic and quasi-periodic behavior. This is a result of the information loss of the plot, since chaos and quasi-periodicity are alike in the result that both do not revisit any state twice.

Bifurcation diagrams have a variety of benefits. Firstly, a bifurcation diagram exhibits, what behaviors a system can produce and thus help getting a initial overview of the systems behavior. Secondly it is possible to find parameter domains where a behavior is stable, which is interesting in systems where exactness of the parameter setting is not guaranteed. Finally, one can determine where to find bifurcation points, when, for instance, an oscillation is supposed to follow on a steady behavior.

As mentioned earlier, biases can be used to model input to a network module and thus its variation simulates a changing input. Hence, it is possible to use bifurcation diagrams to find regions for the input at which a desired behavior is shown, and values, where it changes to another.

Due to the fact that the initial state can influence in which attractor the system ends, the outcome may look different depending on the system state before every parameter change.

To find hystereses, bifurcation diagrams can be plotted with the parameter running in both directions. Therefore, the parameter is first increased from minimum to maximum and then backwards or the other way round. During this process, hystereses can be discovered, when a different attractor

is shown at the same parameter value during both runs. In section 4, I will give an example to illustrate this.

The Dynamics Plotter provides help with the creation of these bifurcation diagrams. As it is concerned with neural networks, it enables the user to vary synapse strengths and neuron biases as the system parameters. I will show in the example section 4 how the user is able to set more than one synapse and/or one neuron to vary. By this means, the scope of available parameter changes can be increased.

Additionally to the common properties (see above), the user has to specify a few ones specific to the bifurcation diagram plotter. Among the boolean property if the algorithm runs bidirectional, it needs to be specified, if the neuron activities are reset after every parameter step or not. If chosen to do so, the activities are set to the level, they have been before the start of the calculation process. If not, the activities are preserved, such that if an attractor was found in the step before, it will determine the next step's activities. Due to the fact that attractors often move only slightly or not at all, this can lead to an increased calculation speed.[Nusse and Yorke, 1994, p. 232]

The general algorithm of the bifurcation diagram plotter is quickly explained. For every pixel in the x-dimension it will do one parameter step, starting with the specified minimum. After setting the parameters to the network, the network is initiated and takes the number of pre-steps specified. Subsequently it will start searching for attractors until the maximal number of steps allowed is reached or a repetition within the neuron activities is found. Therefore, after every step the network takes, all activities are saved and compared to the ones before. In case of a repetition, the algorithm determines the size of the period and which pixels in y-direction represent the new values. If no repetition is found, all values are plotted. After that is done for every parameter step, the procedure is repeated, starting with the maximal parameter value specified, if a bidirectional run is required by the user.

3.2.2 Isoperiodic Plot

Isoperiodic plots show areas in the parameter space that have the same period length. Parameters are therefore varied and the network dynamics are executed until an attractor is found. The period of the attractor is visualized by a dot with a color chosen for the particular period length. By this, attractors with the same period ('isoperiodic') have the same color. See figure 5 for an example of an isoperiodic plot. Here the white area denotes very long periods, chaos or quasi-periodicity (see above), whereas black stands for fixed-point behavior. Colors mark settings that lead to periodicity.

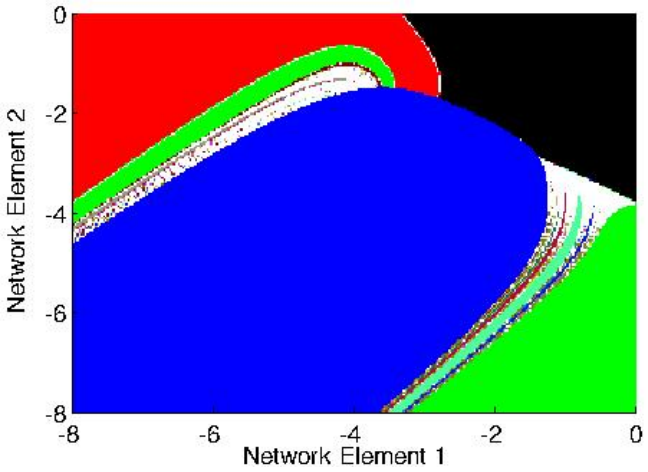

Figure 5: Isoperiodic plot

As with the bifurcation plot, on the x-axis neuron biases and/or synapse strengths are varied. However, here the same holds for the y-axis. More information is hence excluded in the isoperiodic plot, because it does not show the activity of the individual attractors. Instead the output is reduced to the periods of the attractors. Thereby, the isoperiod plots can give a compact overview of the isoperiodic regions. Additionally, scattered points of different color can hint at coexisting attractors.[Negrello et al., 2008, p. 92f]

With regard to the properties set by the user, the isoperiod plotter is similar to bifurcation diagram plotter. The highest possible period that

is found is specified by maximal steps that the algorithm does for every parameter change. Moreover, it is again possible to choose to reset the neuron activities before each calculation. Thus, the attractor (and thereby the period length) depends on the initial state.

Additionally, the computation modules of the bifurcation diagrams and isoperiodic plots do resemble each other. Again, the algorithm uses the new parameter settings and triggers the network. When an attractor is found, or the maximal number of steps is reached, the period is inserted into the output matrix (or a zero in case that no attractor is found). Due to the parameter changes on both axes, this takes considerably longer.

3.2.3 Basins of Attraction Plot

As mentioned in section 2.1, basins of attraction are sets of states from which a system converges in a particular attractor. Therefore the neuron activities are altered, whereas the parameters stay fixed.

To create a plot of the basins, the Dynamics Plotter calculates the attractor for all output activities of the neurons and assigns numbers to these attractors. For every pixel in the plot, the resulting attractor is checked whether it was found before. If so, the same number is assigned to that pixel as to the pixel, where the attractor appeared first. If no attractor can be found, the pixel is assigned a zero, indicating possible quasi-periodicity or chaos. See figure 6 for an illustration.

The properties specified by the user do very much resemble those of the isoperiodic plot with an addition of the possibility to limit the maximal period of an attractor that is shown. All higher periodic attractors are treated as if chaotic and shown in white on the plot. Thus the plot can be kept clearer, if there are many high-periodic attractors which are not of interest. Using the property to reset to the initial state is only reasonable if there are more than two neurons in the network. Otherwise the reseted output activities will be immediately overwritten. Hence this property applies only to neurons, whose output is not varied.

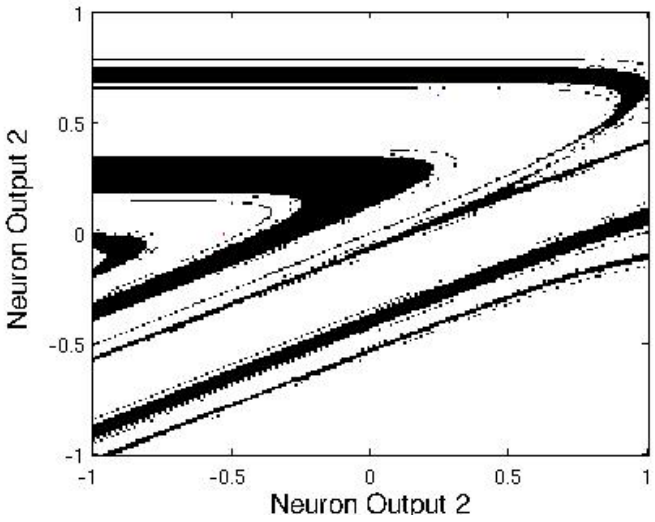

Figure 6: Basins of attraction plot

3.2.4 Transients Plot

The transient or trajectory plotter is the most simple module of the Dynamics Plotter. It examines the activity of one neuron and plots its output against the number of network steps from the initial network state. Figure 7 gives an example of a transient plot. It shows a neuron's activity settling down into its long-term behavior, obviously a fixed point. There is no parameter change in this process. The x-axis therefore simply displays the network steps. Due to this, the transient plotter only needs specific user input for the observed neuron and the number of steps that will be made.

As there already exists an activity plotter in the NERD environment (see section 2.3), it is debatable, how much of a use this plotter is. In comparison to the NERD plotter, this module of the Dynamics Plotter has two benefits, as it does plot the transient and thus the settling behavior. In contrast, the NERD plotter, due to the fact that it plots online, shows the actual live activity, which quickly is an attractor behavior. The second advantage of this transient plotter is the exportability of the results (compare section 3.3).

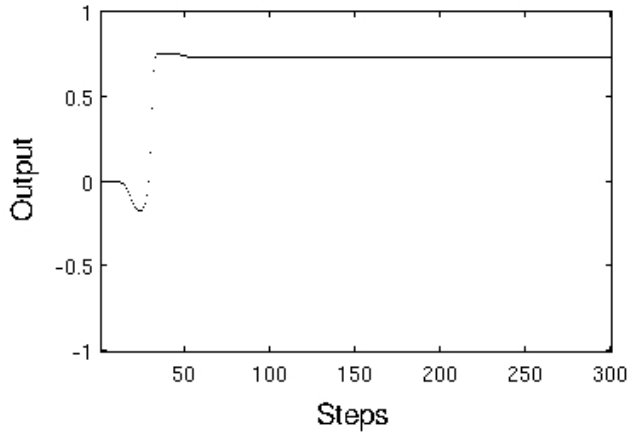

Figure 7: Transient plot

3.3 Visualization

The data created in the computation process before, is visualized either internally with the online plotter or can be exported to use other programs, such as Matlab, to generate diagrams from the data.

3.3.1 Online Plotter

The online plotter displays the resulting diagram directly in the NERD environment. A new window is created to show the plot and give additional information, such as the axes descriptions. Figure 8 presents an example of the online plotter, which here displays a bifurcation diagram. Besides the actual diagram, it shows the axes descriptions, the minima and maxima of each axis and also the name of the computation module that was used. A tooltip gives information about the value and the coordinates at the position of the cursor. The coordinates in the plot correspond to the values on the axes. Therefore in this figure, at a parameter value of about 1.38 and a neuron output value of approximately 0.67 the matrix entry is 1, which is translated into black color. I will give more detail about this in section 4.

As its name suggests, the online plotter is able to display the existent results during the calculation process. Alternatively, the data can be printed when the calculation process has finished. This gives a slight speed improve-

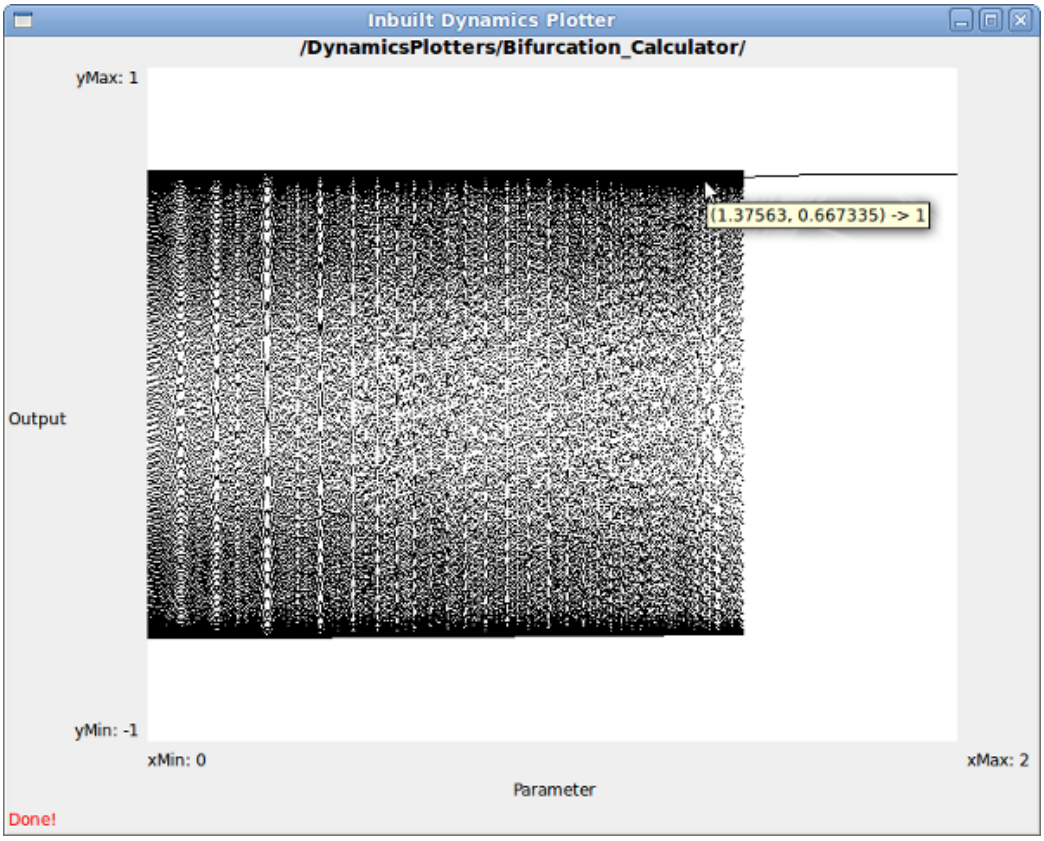

Figure 8: Internal online plotter

ment. If the plotter operates online it updates the diagram about twice a second, by using the meanwhile calculated data. Thus the user is able to observe the process of the plot creation, which can help, for example, to identify hystereses. More benefits of the internal plotter are the quickly maintained results and its straightforward handling.

3.3.2 Exporter

Exporters operate as interfaces between the Dynamics Plotter computation modules and external displaying programs or scripts. Therefore, the data matrix is formatted and saved to an output file.

The particular exporters are adjusted to the respective plotting program. Hence, an exporter for Matlab creates a file, which is readable by a Matlab script. It is likely, that the file is not readable by, for instance, QtOctave or Scilab. To use those programs a specified exporter is needed. This may be avoided by creating scripts that can read the output file designed for Matlab.

The export of the data enables the user to exploit powerful tools to interpret and adjust the plots. This can be of use, when creating plots for scientific papers or books. The export further ensures persistence of the data, due to the creation of files.

3.3.3 Matlab Script

Scripts like the existing one for Matlab are instructions for the respective program how to read, interpret and plot the output data. Hence the user only needs to specify the file that holds the data.

Running the Matlab script generates a query where to find the data file. When this is specified, the script imports the computation module name, axes descriptions and data from the file. Thereby a figure displaying the diagram, axes, axes descriptions and axes values is created. Further, the matrix entries are translated into colors and written to the figure.

Figure 9 shows the figure property editor of Matlab. With the help of the color panel on the right of the figure, one is able to change the colors and their range. Thus it is possible, for example, to give all values higher

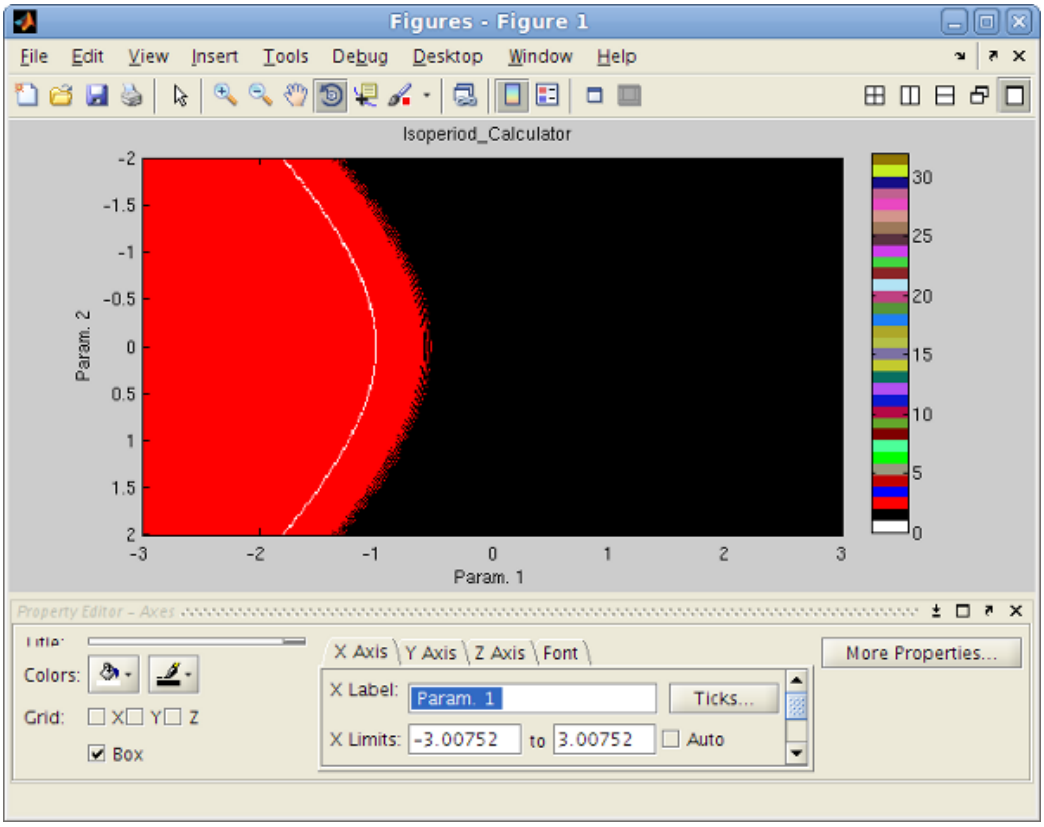

Figure 9: Matlab figure property window

than some number the same color. The plot is easily reversed and scaled and descriptions can be changed and formatted. There are many more options for all parts of the figure to be modified.

4 Application Example

This section gives an example of the analyses of a 2-neuron artificial neural network. I will first shortly present this network. Secondly, the process of creating these plots is demonstrated by explaining the important properties and parameterization of the network for each plot. In the end I give a short overview of the usage of the visualization tools.

The network used here to illustrate the application of the Dynamics Plotter consists of two neurons. Figure 10 illustrates this network. I will refer to the neurons as neurons 1 and 2 as figure 10 suggests, thus the bias of neuron 1 will be b_1. The parameters shown in the plot are set for the bifurcation plot, as presented in a moment.

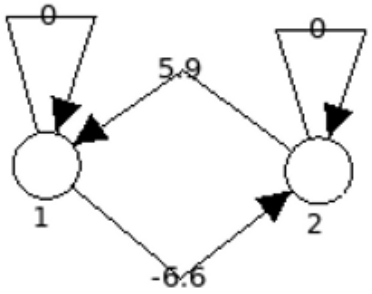

Figure 10: 2-neuron network

The neuron biases and synapse weights are set with the help of the NERD network editor (see section 2.3). Additionally the user can adjust the initial neuron output activities, to potentially find different attractors.

For the *bifurcation diagram*, the parameters are shown in table 3. Before starting the computation the properties for the bifurcation plotter have to be set. As mentioned above, some properties are used for more than one plot type. Table 1 gives an overview over those properties. There are properties, that are not in the table, such as the axes descriptions, but these are of no importance for the computation process. Further there are some properties, that are read-only, like the data matrix and are not listed either. In table 2 the bifurcation-specific properties are shown.

To create a bifurcation diagram, the user has to choose a size first. The PlotPixelsX-property determines the number of parameter steps it takes from the minimal value (MinimaOfVElems) of the parameter to the maximum (MaximaOfVElems). For the bifurcation diagram, either synapse strengths or neuron biases can be used as the parameter. It is possible to modify more than one network element or more than one of a kind. The user inserts the IDs of the neurons and synapses into the IdsOfVariedNetwElems-property as a string with the IDs separated by commas and/or vertical bars. Thus, it is possible to use the NERD network editor for quickly getting a list of IDs, which can be easily acquired and are saved to the cache as a string. Accordingly, a list minima and maxima of the network elements are given as a string, in the same order as the IDs-list. Similarly, the PlotPixelsY-property gives the number of output buckets, as outlined above, from minimum of the output (MinOutputRange) to the maximum (MaxOutputRange). The output here can be either the output of a single neuron or alternatively of multiple neurons, such as every neuron in the network. The user specifies the IDs (IdsOfObservedNeurons) of the observed neurons and for every attractor state the average of those is used. When all other properties are specified (see section 3.2 for more on those properties) the Activate-property can be set to *true* to start the computation.

Property	**Value Type**	**Description**
Activate	bool	*True* starts computation
MaxSteps	int	Maximal number of steps
PrerunSteps	int	Steps before search starts
Tolerance	double	Min. distance to distinguish two points
ResetToInitState	bool	*True* resets before every step
PlotPixelsX	int	Horizontal size
PlotPixelsY	int	Vertical size

Table 1: General properties

To create the example diagram[6] the 2-neuron network shown in figure 10

[6]This example was taken from [Pasemann, 2002] and reproduced to provide comparableness.

Property	Value	Description
IdsOfObservedNeurons	string	List of observed neurons' IDs
Bidirectional	bool	Runs also backwards, if *true*
MaxOutputRange	double	Maximal value on y-axis
MinOutputRange	double	Minimal value on y-axis
IdsOfVariedNetwElems	string	List of IDs of varied elements
MaximaOfVElems	string	List of maxima for varied elements
MinimaOfVElems	string	List of minima for varied elements

Table 2: Properties of the bifurcation diagram

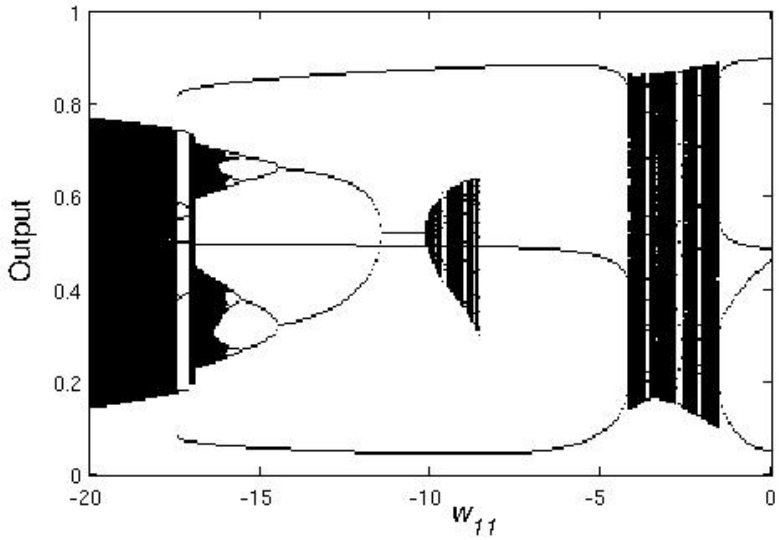

Figure 11: Bifurcation diagram for varied w_{11}

was parameterized with the fixed parameters given in table 3 and varying the self-connection w_{11} of neuron 1 from -20 to 0. The transfer function of the neurons was chosen to be the standard sigmoid. The computation module was set to do 500 pre-steps and 2000 maximal steps if no attractor was found. Tolerance was set to 10^{-8} and resetting to the initial activities was set to *false*. The algorithm ran forward and backward and the plot size was 300 times 200 pixels. The output value was the average of both neuron's outputs. See figure 11 for the result. See figure 12 for the result of the same parameters and property values, except that the algorithm was not set to run bidirectional. By comparison of both plots one can identify a hysteresis, for example a sudden jump from chaotic behavior to a period-3 attractor at approximately $w_{11} = -8.5$ in figure 12. As can be seen from figure 11 the reversal of the parameter does not lead to jump back to the chaotic behavior, but instead the system stays in the period-3 behavior until a value of about -17.4 is reached.[Pasemann, 2002]

Parameter	b_1	b_2	w_{11}	w_{12}	w_{21}	w_{22}
Value	-3.8	3	-	5.9	-6.6	0

Table 3: Parameters for the bifurcation diagram

To use the Dynamics Plotter to create an *isoperiodic plot*, again some specific properties must be set besides the general ones. Opposed to bifurcation plot, the isoperiodic plot uses parameters on both axes. Accordingly, these must be specified via their IDs and their minimal and maximal values need to be set. See table 4 for an overview over these properties. Apart from that, the process of creating both plots are very similar.

For the isoperiodic example plot, the 2-neuron network of 10 is differently parameterized. See table 5 for the fixed parameters. Synapse strengths w_{12} and w_{11} are varied from 0 to 8 and -18 to 0, respectively. As the transfer function the standard sigmoid was chosen.

In the creation of figure 13 again the plot size was 300 times 200 pixels.[7]

[7]The template for this plot can again be found in [Pasemann, 2002].

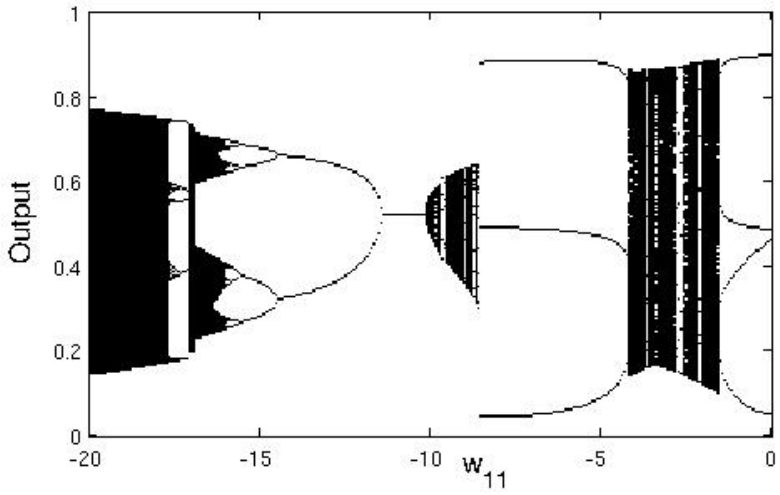

Figure 12: One-directional bifurcation diagram for varied w_{11}

Property	**Value**	**Description**
XIdsOfVNetwElems	string	List of IDs of x-elements
XMaximaOfVElems	string	List of maxima for x-elements
XMinimaOfVElems	string	List of minima for x-elements
YIdsOfVNetwElems	string	List of IDs of y-elements
YMaximaOfVElems	string	List of maxima for y-elements
YMinimaOfVElems	string	List of minima for y-elements

Table 4: Properties of the isoperiodic plot

Parameter	b_1	b_2	w_{11}	w_{12}	w_{21}	w_{22}
Value	-3	4	-	-	-6	0

Table 5: Parameters for the isoperiodic plot

This specifies the step size of the parameters w_{11} and w_{12}. The tolerance was set to 10^{-8} while the neuron activities were not reseted. After 3500 steps the algorithm searched in 50 steps for attractors. The digits in the different colors give the respective periods. These were added afterwards manually. White areas denote very long transients, high periods, quasi-periodicity or chaos.

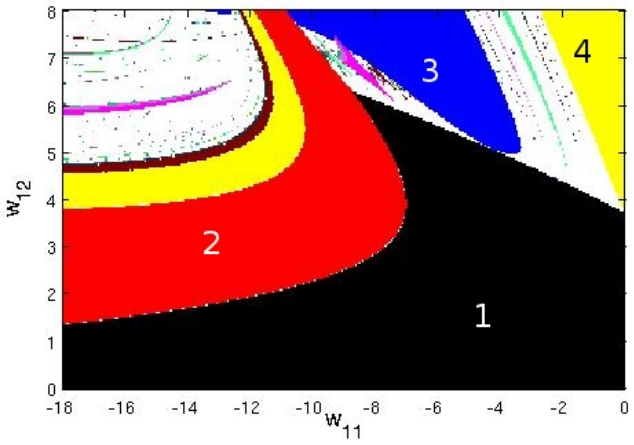

Figure 13: Isoperiodic plot with varied w_{12} and w_{11}

Similar properties are needed for a *basins of attraction plot* as for the plots above. Besides general properties, one or more neurons must be specified for each axis, whose output activity are varied. Further a maximal period can be specified to reduce the plotted attractor basins to those which are of a lower period. Table 6 gives an overview of the properties.

The network used for this plot is symmetric, without self-connections. See table 7 for the parameters, which are all fixed.

The exemplary basins of attraction plot (figure 14) was created with 100 pre-steps and a maximum of 100 steps for the attractor search. The maximal period was specified as 16, but did not influence the outcome (otherwise a white (zero) area would be noticeable). It was not reseted to the initial state and the tolerance was 10^{-6}.

Property	Value	Description
MaxPeriod	int	Max. period that is shown
XIdsOfVNeurons	string	List of IDs of x-neurons
XMaximaOfVNeurons	string	List of maxima for x-neurons
XMinimaOfVNeurons	string	List of minima for x-neurons
YIdsOfVNeurons	string	List of IDs of y-neurons
YMaximaOfVNeurons	string	List of maxima for y-neurons
YMinimaOfVNeurons	string	List of minima for y-neurons

Table 6: Properties of the basins of attraction plot

Parameter	b_1	b_2	w_{11}	w_{12}	w_{21}	w_{22}
Value	0.001	0.001	0	2	2	0

Table 7: Parameters for the basins of attraction plot

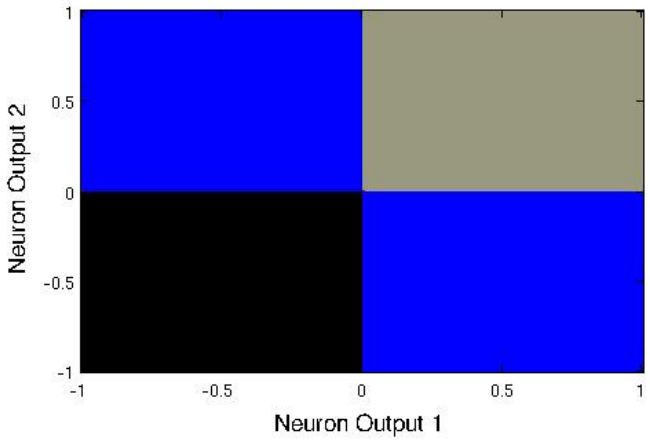

Figure 14: Basins of attraction plot with varied neuron outputs

The properties for the *transient plot* consist only of the neuron that shall be observed and the number of steps the algorithm takes (table 8), in addition to the general properties. For the creation of figure 15 the same fixed parameters for the network were used as for the basins of attraction plot. The initial output activity of both neurons was set to $x_1 = x_2 = 0.0001$. The resulting transient quickly (after about 15 steps) shows a fixed point behavior at an output value of approximately 0.96. Hence, the attractor for the gray area (first quadrant) of figure 14 is this attracting fixed point.

Property	**Value**	**Description**
IdOfObservedNeuron	string	ID of observed neuron
NoOfStepsX	int	Number of steps

Table 8: Properties of the transient plot

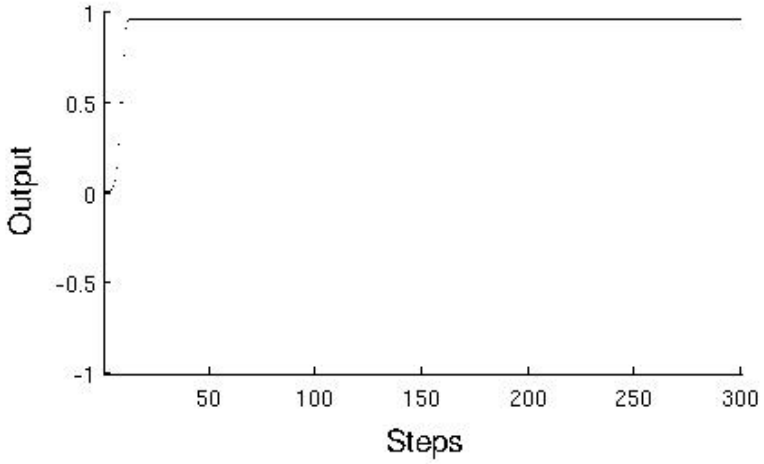

Figure 15: Transient plot with $x_1 = x_2 = 0.0001$

For all plots described above, the properties for the visualization must be set as well. These are identical for all and are summarized in table 9. The property PlotterProgram specifies how the data is visualized. The available possibilities can be chosen from a drop-down list or entered as a string. 'Matlab' activates the exporter for a Matlab-readable file, whereas 'Inbuilt

Plotter' calls the online plotter. The string can be combined to, for instance, 'Matlab and Inbuilt Plotter' to create a file and simultaneously use the online output. InbuiltPlotterOnline is boolean that can set to *true* to observe the plot in the online plotter window during the computation process. In OutputPath it is specified where the files, created by the exporters, are saved.

Property	**Value**	**Description**
PlotterProgram	string	Specification of plotting program
InbuiltPlotterOnline	bool	*True* for online plotting
OutputPath	string	Path to output file

Table 9: Properties for visualization

5 Conclusion

For the presentation of the Dynamics Plotter the preliminaries introduce the reader to the concepts of dynamical systems and artificial neural networks. Moreover the NERD environment is shortly described. With this in mind section 3 explains the design of the program. Additionally specific information is given for the particular plot types and thus a sufficient background to understand their characteristics is provided. In the example section 4 the process of the creation of each plot is illustrated with a description of the user inputs.

The Dynamics Plotter simplifies the analysis of dynamical behavior of neural networks. The combination with NERD enables the user to create networks and plot their behavior with only one program. Hence, with less effort more knowledge about networks can be obtained. I would consider the goal of this thesis as reached with reservations. The program is easily extensible and to a good extent simple to handle. For an easier use an extended graphical user interface would be desirable. An interactive way to specify new parameter ranges by selecting these on an existing plot with the cursor, is not yet implemented, for instance. A further simplification would be given by an interface displaying the existent plots to choose from.

This visualization tool differs from existing ones mainly by its integration in a more powerful program and the specialization on neural networks. During the programming process this was noticeable by the fact, that most references did not focus on reducing the number of calculation steps, which in contrast is important, if network steps take a great amount of time in comparison to the data calculation.

In future, users may need more than the existing plots, such as, for example, first return maps or three dimensional plots. These have yet to be implemented. Moreover, possibly an exporter for an open source program would be a good expansion, to allow usage of only open source software. Nevertheless the Dynamics Plotter provides a basis for these upgrades and enables the creation of frequently used plots with different ways of illustration. Henceforth

users can quickly visualize the behaviors of networks created with NERD and generate plots for a quick overview or even for scientific papers.

List of Figures

List of Tables

References

[Cessac and Samuelides, 2007] Cessac, B. and Samuelides, M. (2007). From neuron to neural networks dynamics. *The European Physical Journal - Special Topics*, 142:7–88.

[Devaney, 1990] Devaney, R. L. (1990). *Chaos, Fractals, and Dynamics*. Addison-Wesley, Menlo Park, Calif., 9. reprint edition.

[Ermentrout, 1990] Ermentrout, B. (1990). *PhasePlane, Version 3.0: The Dynamical Systems Tool*. Brooks/Cole Publishing Co.

[Haykin, 1998] Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 2nd edition.

[Negrello et al., 2008] Negrello, M., Huelse, M., and Pasemann, F. (2008). Adaptive neurodynamics. In *Applications of complex adaptive systems*, pages 85–111. IGI Global Publishing.

[Nusse and Yorke, 1994] Nusse, H. E. and Yorke, J. A. (1994). *Dynamics: Numerical Explorations*. Springer-Verlag Berlin and Heidelberg GmbH & Co. K.

[Osipenko, 2007] Osipenko, G. (2007). *Dynamical systems, graphs, and algorithms*. Lecture notes in mathematics : a collection of informal reports and seminars, ISSN 0075-8434. Springer, Berlin.

[Pasemann, 2002] Pasemann, F. (2002). Complex dynamics and the structure of small neural networks. *Network: Computation in Neural Systems*, 13:195–216.

[Pasemann et al., 2001] Pasemann, F., Steinmetz, U., Hülse, M., and Lara, B. (2001). Robot control and the evolution of modular neurodynamics. *Theory in Biosciences*, 120:311–326.

[Rempis et al., 2010] Rempis, C., Thomas, V., Bachmann, F., and Pasemann, F. (2010). Nerd - neurodynamics and evolutionary robotics development kit. *LNAI*, 6472:121–132.

[Strogatz, 2000] Strogatz, S. H. (2000). *Nonlinear dynamics and chaos*. Westview Press, Cambridge, Mass., 1. paperback printing, [reprinted] edition.

Declaration

I hereby affirm, that I wrote this bachelor thesis independently without unauthorized assistance. I have not made use of any other resources or means than those indicated.

Hamburg, November 17, 2010

_____

Till Faber