Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Engineering and
Networks Laboratory

Spring Semester 2008 Prof. Dr. E. Zitzler

Semester Project

Parameter Estimation in Differential Equation Models

Christian Schürch

Advisor: Tim Hohm

Acknowledgements

I would like to thank my advisor Tim Hohm for many helpful discussions.

Zürich, 30. Mai 2008

Christian Schürch

Abstract

In this project a special type of differential equation systems, the so called reaction diffusion systems with parameters are considered. We try to determine for which parameters the solution of a reaction diffusion system is a timely stable and spatially inhomogeneous pattern. This we do by estimating the boundaries between different solution regions. A solution region is a part of the parameter space of a reaction diffusion system, for which the solutions behave qualitatively similar. The boundaries are estimated with an Evolutionary Algorithm. This Evolutionary algorithm estimates for a parameter set how much one has to change these parameters to be on a boundary and tries to find the boundaries based on this estimate. The estimate is obtained with the help of data from a reaction diffusion system with known boundaries. At the end the Evolutionary Algorithm is applied to two reaction diffusion systems with unknown boundaries. It turns out, that the boundaries can reliably be found and hence the parameter values with pattern formation.

Zusammenfassung

In dieser Arbeit wird ein spezieller Typ von Differentialgleichungssystemen, die so genannten Reaktions-Diffusions-Systeme mit Parametern, betrachtet. Wir versuchen diejenigen Parameterwerte zu bestimmen, für die die Lösung des Reaktions-Diffusions-Systems ein zeitlich stabiles und räumlich inhomogenes Muster ist. Dazu schätzen wir die Grenzen zwischen verschiedenen Lösungsregionen. Eine Lösungsregion ist ein Teil des Parameterraumes eines Reaktions-Diffusions-Systems, für den sich die Lösungen qualitativ gleich verhalten. Es wird versucht, die Grenzen mit Hilfe eines Evolutionären Algorithmus zu finden. Dieser Algorithmus schätzt für einen Parameter Satz, wie stark man diese Parameter ändern muss um auf einer Grenze zu sein. Basierend auf dieser Schätzung versucht der Algorithmus die Grenzen zu finden. Die Schätzung erhält man mit Hilfe von Daten, welche bei einem Reaktions-Diffusions-System mit bekannten Grenzen gesammelt werden. Am Ende wird der Evolutionäre Algorithmus auf zwei Reaktions-Diffusions-Systeme mit unbekannten Grenzen angewandt. Es stellt sich heraus, dass die Grenzen zuverlässig gefunden werden können und somit auch die Parameterwerte mit Musterbildung.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The behaviour of a reaction diffusion system[1] depends on the values of its parameters. The goal of this semester project is to find an algorithm, which is able to locate the boundaries between different solution regions. A solution region is a part of the parameter space of a reaction diffusion system, for which the solutions behave similar. The solution region of our main interest is the region, where the solution of the differential equation system is a timely stable and spatially inhomogeneous pattern. Because you can choose in general infinite many values for the parameters, it is impossible to simulate the solution to a differential equation system for all possible values of the parameters. But one can solve the problem of finding the boundaries between the different solution regions analytically, by applying the stability analysis, which is explained in section 1.2. Unfortunately the stability analysis becomes very complicated for systems with many parameters, so we wish to have an algorithm, which does the same. Because there are many different reaction diffusion systems, which can produce stable inhomogeneous patterns, the task of finding an algorithm, which can find the wished parameters is difficult.

The three sections in this chapter are about three important and often used themes of this project, namely the biological pattern formation, the stability analysis and a kind of optimization algorithms, the so called Evolutionary Algorithms (EA).

In chapter 2 the fitness functions[2] used in this project are explained and the methods to build them are discussed. In chapter 3 a special EA is explained, the CMA and an algorithm, which is not an EA, is introduced. In chapter 4 the EA are tested on the simple system characterized by the equation system (1.1) - (1.2) and the results are displayed. In chapter 5 the EA are applied to two reaction diffusion systems with unknown boundaries.

[1]Reaction diffusion systems are a subclass of differential equation systems and are explained in the next subsection 1.1

[2]A fitness function is a part of every EA and is explained in section 1.3

Finally in chapter 6 the conclusions of this semester project are summarized.

1.1 Biological Pattern Formation

Many animals have a coat with a pattern on it, like for example a leopard, as shown in figure 1.1.

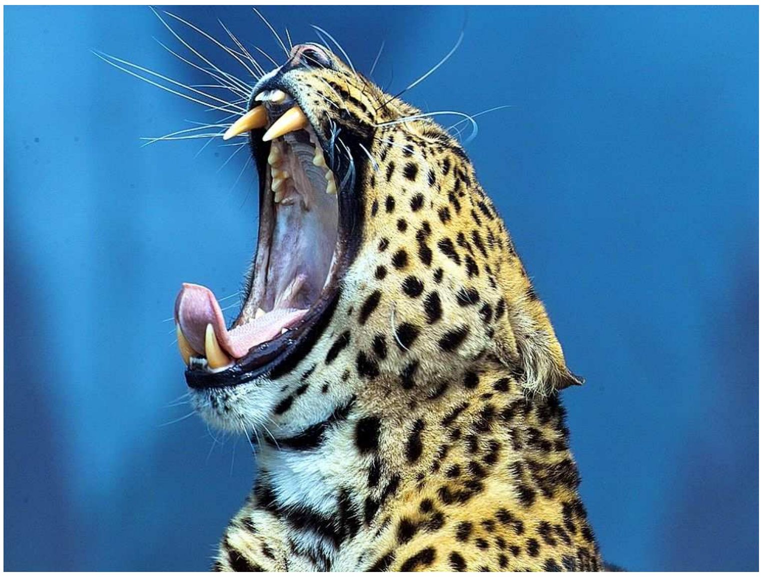


Figure 1.1: A leopard. This picture was downloaded from the link: http://www.big-cats.de/bild.php4?kat=leopard&id=b_leopard_gaehnen

But how can such a pattern be formed, although every cell of the skin has the same genetic material? One kind of models to explain this are reaction diffusion systems, which are a subclass of differential equation systems. These models are based on interactions of two or more chemicals and on their diffusion[3]. As an example look at the equation system (1.1) - (1.2).

$$\frac{\partial a}{\partial t} = D \cdot \Delta a + \frac{a^2}{h} - a + \sigma \tag{1.1}$$

$$\frac{\partial h}{\partial t} = \Delta h + \mu \cdot (a^2 - h) \tag{1.2}$$

In these equations, a is the concentration of one chemical and h is the concentration of the other chemical. Δ is the Laplace-Operator, which is one-, two- or three-dimensional depending on the application, t is the time and σ, μ and D are real parameters greater than zero. The homogeneous steady-state solution is the timely stable solution without diffusion. The

[3]In [2] and [3] reaction diffusion systems are treated more detailed.

concentrations of this solution are $a_0 = 1 + \sigma$ and $h_0 = a_0^2$, which can easily be obtained by setting $\frac{\partial a}{\partial t}$, $\frac{\partial h}{\partial t}$, Δa and Δh to zero. It is a surprising fact, that two chemicals with homogeneous spatial concentrations can form stable inhomogeneous patterns if one slightly perturbs the homogeneous spatially concentrations and therefore adds diffusion. Stable inhomogeneous patterns, are patterns, which do not change in time anymore, but the concentrations of the chemicals are not spatial constant. In our example (1.1) - (1.2), this corresponds to $\frac{\partial a}{\partial t} = 0$, $\frac{\partial h}{\partial t} = 0$, $\Delta a \neq 0$ and $\Delta h \neq 0$ and additionally $D \ll 1$ and $\mu > 1$.

If one adjusts the parameters σ, μ and D correct and simulate the equation system (1.1) - (1.2) sufficiently long for the two-dimensional space and for the initial conditions $a = a_0$ and $h = h_0$ with a small perturbation, you get figure 1.2.

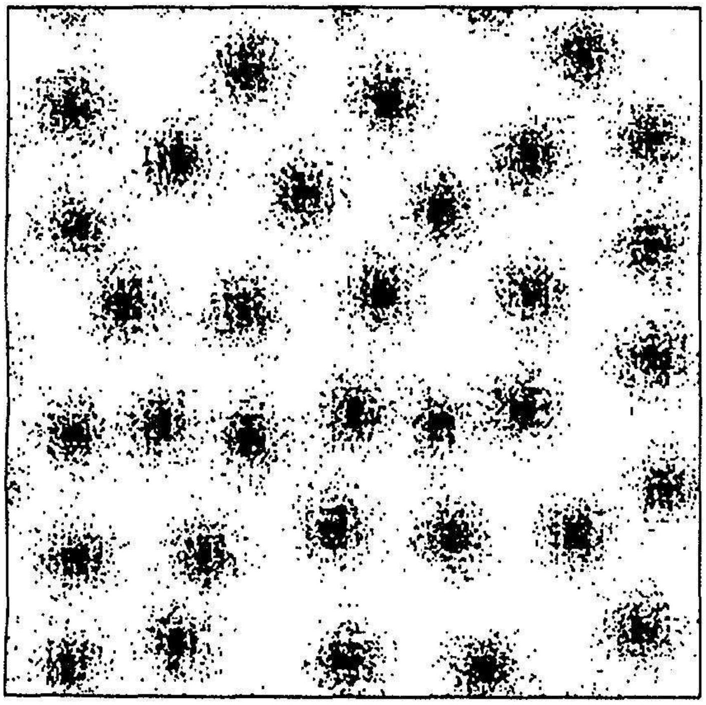


Figure 1.2: This stable inhomogeneous pattern can be formed with the equation system (1.1) - (1.2). The concentration of a is shown in dependence of a part of the two-dimensional space. It seems to look similar to the coat pattern of a leopard. This figure is taken from [3]

In this figure you see a stable inhomogeneous pattern, which arises just for some parameter values of σ, μ and D. The key question of this project is: For which parameter values can a reaction diffusion system produce stable inhomogeneous patterns?

1.2 Stability Analysis

In a stability analysis one linearizes the differential equation system, then one can analytically solve the linearized system. The solution for the linearized system suffices to decide, how the behaviour of the not linearized system is. So stability analysis is a useful tool to decide whether a set of parameters can produce stable patterns or not. If we look at the equation system (1.1) - (1.2) the free parameters are μ, D and σ. The stability analysis now tells us how a and h in dependence of μ, D and σ behave if we slightly perturb the concentrations a and h out of the homogeneous steady-state solution. In our example of the equation system (1.1) - (1.2) the stability analysis was already done [3], with the following results:

There are four solution regions, which are called G_a, G_b, I and H:

1. Region G_a: $0 < \mu < (\sqrt{\frac{2}{a_0}} - 1)^2$ and $0 < D < \infty$, perturbations grow exponentially
2. Region G_b: $(\sqrt{\frac{2}{a_0}} - 1)^2 < \mu < \frac{2}{a_0} - 1$ and $0 < D < \infty$, perturbations oscillate and grow exponentially
3. Region H: $\mu > \frac{2}{a_0} - 1$ and $\mu \cdot D > (\sqrt{\frac{2}{a_0}} - 1)^2$, perturbations are damped, so the homogeneous steady state is stable
4. Region I: $\mu \geq \frac{2}{a_0} - 1$ and $\mu \cdot D < (\sqrt{\frac{2}{a_0}} - 1)^2$, the homogeneous steady-state solution is unstable and the system develops a stable inhomogeneous pattern

In figure 1.3 you see a picture of the (μ, D) space divided up into the four different solution regions G_a, G_b, I and H. The region G'_b will be ignored, because the behaviour of the solutions in this region depends also on the initial conditions of the concentrations and not only on μ and D. This is a fineness which we do not want to consider and so we assume, that G'_b belongs to G_b.

Fore more informations on biological pattern formation or stability analysis, especially on how to derive a stability analysis, see [3].

1.3 Evolutionary Algorithms

An EA is an optimization algorithm, which imitates the evolution. In an EA you generate an initial population of individuals. Individuals can be very abstract things, like points in a space or a table of things which one has in his back-pack. These individuals will be evaluated according to a fitness function. The fitness function takes as input an individual and produces a real valued output. The fitness function measures the quality of an

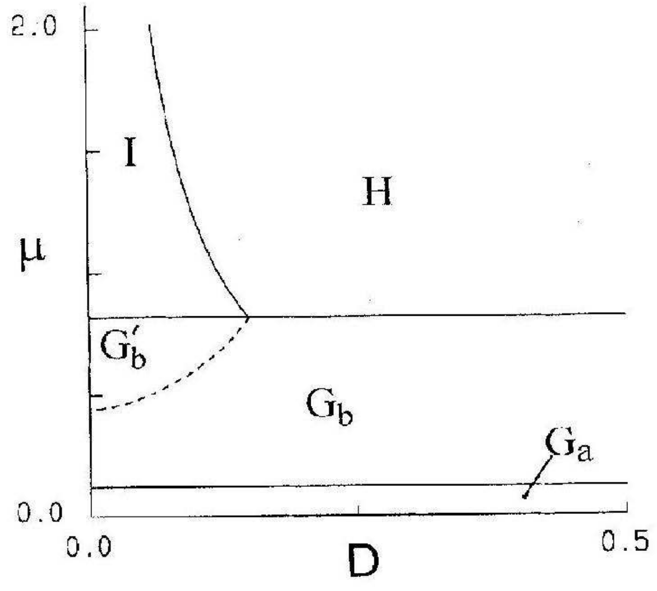


Figure 1.3: Image of the result of the stability analysis for the differential equation system (1.1) - (1.2) taken from [3] with $\sigma = 0.1$

individual. Sometimes an individual is better if it has a big fitness value and sometimes an individual is better if it has a small fitness value. This depends on the application, but because we can always change between the two by multiplying the fitness function by -1, they are equivalent. Without limiting the generality, we assume in this project, that an individual is better, if it has a smaller fitness value. Based on this fitness value some of the individuals of the initial population will die, some will survive, some will recombine and some will mutate.

Here recombination means, to take a set of individuals and to produce a set of offspring by recombining the information of the taken parent individuals. Mutation means, to take one individual and produce based on the taken individual one individual. The recombination and mutation operator have the purpose to create the necessary diversity within the population. Often the better individuals, based on the fitness function, will survive and will have more likely a child, hence the bad individual should die out. The goal of an EA is to find a set of individuals with a small or the minimal fitness value. A possible scheme of an EA is shown in figure 1.4.

In this scheme, you first generate an initial population. Then these individuals are evaluated according to a fitness function. Now a loop with a termination condition appears. The termination condition could be for example a running time limit of the algorithm or a rule like: If an individual with fitness value smaller than 0.1 was found, terminate the program. In

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Figure 1.4: A possible scheme of an EA taken from [1]

the loop, parents are selected first. An individual is more likely selected as a parent, if it has a small fitness value. Then the chosen parents are grouped in pairs and to every pair the recombination operator is applied, which results in one child per pair. To these children, in figure 1.4 called offspring, the mutation operator is applied. Now we have a population made of the initial population, the children of the selected parents and the mutated children of the selected parents, whereat the not mutated children are often not considered anymore. After the mutation, a second selection stage occurs. In this selection stage the population is split into two parts. One part consists of the surviving individuals, which form the population for the next loop run. The other part consists of the dead individuals. The smaller the fitness value of an individual, the more likely it belongs to the surviving individuals. We not always take the best individuals, because then it is more likely to stuck in a local minimum. If we let survive also some worse individuals, perhaps they lead to another local minimum or even the global minimum. So an EA can be used to find a minimum or a small value of a function. In general an EA does not have to find the minimum value of the fitness function, it can be stuck in a local minimum or the termination condition ends the algorithm before the minimum value of the fitness function was found. In this project an algorithm should be found, which is able to locate the boundaries between different solution regions of a reaction diffusion system with parameters. A possible approach is to use an EA. In this EA the individuals are points in the parameter space. A fitness function, which is the focus of chapter 2, tells us how distant the nearest boundary is. The goal is to have a parameter set for which the distance to a boundary is small or minimal.

For a more comprehensive introduction to EA, see [1].

Chapter 2

Fitness Function

In this chapter the fitness functions used in this project are explained and the methods to build them are discussed. In subsection 2.2.1 a classification function is introduced, which is not a fitness function. Nevertheless it is introduced in this chapter, because we want to have a different approach.

The EA of this project should locate the boundaries between different solution regions for reaction diffusion systems. The individuals are points in the parameter space. As mentioned in chapter 1 an individual with a small fitness value is a good individual. So the fitness function has to have a small value on the boundaries and the farther away from the boundaries, the bigger value it should have. An easy function, which has these properties, is the distance function:

For every individual in the parameter space, the fitness function is the euclidean distance from the individual to the nearest boundary.

So the fitness function is zero for a point on the boundary and greater than zero for a point not on the boundary. As example we take equations (1.1) - (1.2) with $\sigma = 0$. Therefore $a_0 = 1 + \sigma$ is equal to 1 and $h_0 = a_0^2$ is equal to 1. Hence the solution regions are given by:

1. Region G_a: $0 < \mu < (\sqrt{2} - 1)^2 = 0.172$ and $0 < D < \infty$
2. Region G_b: $0.172 < \mu < \frac{2}{a_0} - 1 = 1$ and $0 < D < \infty$
3. Region H: $\mu > 1$ and $\mu \cdot D > 0.172$
4. Region I: $\mu \geq 1$ and $\mu \cdot D < 0.172$

Because σ is set to zero, the only free parameters are μ and D. So the individuals are points in the (μ, D) space. The fitness value f for an individual (μ, D) is the value:

$$f(\mu, D) = \min_{(\mu_b, D_b) \in B} \sqrt{(\mu - \mu_b)^2 + (D - D_b)^2} \tag{2.1}$$

$$\text{with } B = (\mu_b, D_b) \in \mathit{Boundaries} \tag{2.2}$$

The function $f(\mu, D)$ is only determined, if we know the location of the boundaries. In this example we know the boundaries, because of the already done stability analysis. In general the task is to find the boundaries, because we do not know their location. Then we can not build the distance function. But we can estimate the distance to the nearest boundary, which is explained in the next paragraph.

A part of this function is drawn with MATLAB in figure 2.1:

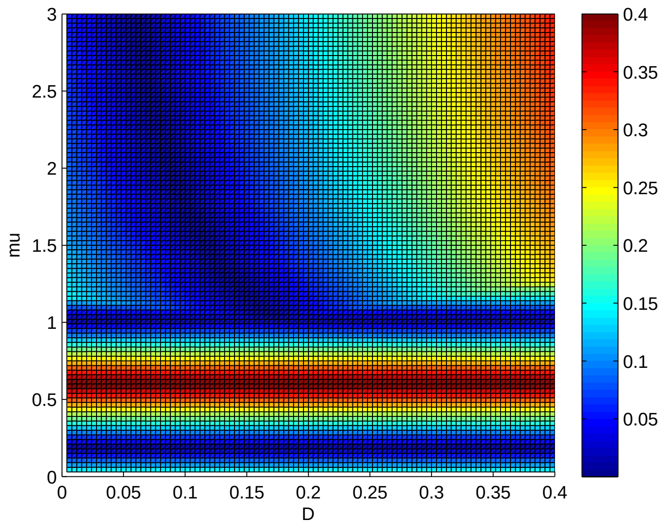


Figure 2.1: The distance from a point in the (μ, D) plane to the nearest boundary.

We would like to have a fitness function, which works for different systems, for example the system described by equations (1.1) - (1.2) as well as for a varied system with five parameters, therefore the fitness function used in the example is not enough general. Another problem is the location of the boundaries. In the example the boundaries are known, because the stability analysis is already done. But in general the boundaries are not known and the task is to find the boundaries. To deal with these problems, we split the fitness function in two parts. The first part of the fitness function, is a function from the parameter set to a real valued vector called *prop*. This function calculates for the given parameter set the solution of the differential equation system numerical using MATLAB and extracts some properties from the solution, the *prop* vector. The second and more difficult part of the fitness function is a function from the vector *prop* to the output called *distance*, which tries to estimate the distance to the nearest boundary based on the properties of the solution. Because we extract always the same properties from every reaction diffusion system, the second part of the fitness function gets independent from the differential equation system. Now the question comes up, which properties should be taken from a solution? Clearly, we need properties, which allow us to estimate the distance to a boundary. The answer to this question used in this project is simple: Take as much properties as possible, but only if you think, they are helpful in estimating the distance to a boundary. This answer is very dusty, but in the next section you will see how the answer is interpreted. Another important question is: How can we make a function from the vector *prop* to *distance*, which estimates the distance to the nearest boundary? The approach used here is to simulate the simple differential equation system (1.1) - (1.2) for many different parameter choices. With these simulations we can determine for every simulated parameter point the *prop* vector. Because the stability analysis for this system is already done, we know the distance between every parameter point and the nearest boundary. So we know for every simulated parameter point the pair *prop* and *distance*. The task now is to make a good function from *prop* to *distance* with the help of the *prop distance* pairs.

2.1 Data Generation

To generate data, i. e. *prop distance* pairs, we simulate the simple differential equation system (2.3) - (2.4) for many different parameter choices, with the slightly perturbed homogeneous steady-state solution as starting condition.

$$\frac{\partial a}{\partial t} = D \cdot \Delta a + \frac{a^2}{h} - a \tag{2.3}$$

$$\frac{\partial h}{\partial t} = \Delta h + \mu \cdot (a^2 - h) \tag{2.4}$$

The reaction diffusion system (2.3) - (2.4) is the reaction diffusion system given by (1.1) - (1.2) with $\sigma = 0$. The simulations are done with MATLAB. MATLAB has some Ordinary Differential Equation Solvers implemented, for example the function *ode45*. We use these solvers in this project. These ODE solvers can solve differential equations of the form,

$$\frac{\partial y}{\partial t} = f(t, y) \tag{2.5}$$

with y a scalar or a vector. At first sight, this seems not to be an appropriate solver for our equation system (2.3) - (2.4), due to the Laplace-Operator Δ and the spatial dependence of the concentrations. But because we can choose the dimension of the vector y as we want, we will see, that these ODE-Solvers are a suitable choice. First we have to decide whether we want a one-, two- or three-dimensional model. For simplicity we take a one-dimensional model, therefore the Laplace-Operator is one-dimensional as well: $\Delta = \frac{\partial^2}{\partial x^2}$. We call this one-dimensional spatial coordinate x, therefore the concentrations a and h are functions of the coordinate x and the time t. In simulations the time and the space need to be discretized. The time coordinate t is discretized in the MATLAB ODE Solvers, but the x coordinate not. We discretize the x coordinate on a one-dimensional grid with mesh spacing δx. Any point is then defined by an index i: $x_i = i \cdot \delta x$, with $0 \leq i \leq N$. The time derivative is approximated in the ODE solvers as well. The Laplace-Operators are approximated as follows:

$$\frac{\partial^2 a(x_i,t)}{\partial x^2} \approx \frac{a(x_{i+1},t) + a(x_{i-1},t) - 2 \cdot a(x_i,t)}{\delta x^2} \tag{2.6}$$

Now we define the k-th element of the vector y, $0 \leq k \leq 2 \cdot N + 1$:

$$y_k(t) = \begin{cases} a(x_k,t) & \text{if } 0 \leq k \leq N \\ h(x_{k-(N+1)},t) & \text{if } N+1 \leq k \leq 2 \cdot N + 1 \end{cases} \tag{2.7}$$

With equations (2.6) and (2.7), we can write a spatial discrete version of equations (2.3) - (2.4):

$$\begin{aligned} \frac{\partial a(x_i,t)}{\partial t} &= D \cdot \frac{a(x_{i+1},t) + a(x_{i-1},t) - 2 \cdot a(x_i,t)}{\delta x^2} + \\ &+ \frac{a(x_i,t)^2}{h(x_i,t)} - a(x_i,t) \qquad (2.8) \\ \frac{\partial h(x_i,t)}{\partial t} &= \frac{h(x_{i+1},t) + h(x_{i-1},t) - 2 \cdot h(x_i,t)}{\delta x^2} + \\ &+ \mu \cdot (a(x_i,t)^2 - h(x_i,t)) \qquad (2.9) \end{aligned}$$

There is just one little problem. In equations (2.8) - (2.9), i goes from 0 to N, so we access the elements $a(x_{-1},t)$, $a(x_{N+1},t)$, $h(x_{-1},t)$ and $a(x_{N+1},t)$, which are not existent. In order to specify these values, we need boundary conditions. A often used boundary condition is the periodic boundary condition, which is here used as well. With a periodic boundary condition, you can think of a function $a(x_i,t)$ as periodically repeated in space. If we arrive at one end of a or h, i. e. $a(x_{-1},t)$, $a(x_{N+1},t)$, $h(x_{-1},t)$ and $a(x_{N+1},t)$, we begin at the other end:

$$\begin{aligned} a(x_{-1},t) &= a(x_N,t) \qquad (2.10) \\ a(x_{N+1},t) &= a(x_0,t) \qquad (2.11) \\ h(x_{-1},t) &= h(x_N,t) \qquad (2.12) \\ h(x_{N+1},t) &= h(x_0,t) \qquad (2.13) \end{aligned}$$

Because equations get very complicated, we do not explicitly write equations (2.10) - (2.13) in the following equations (2.14) - (2.16), but we keep it in mind.

Rewriting equations (2.8) - (2.9) in terms of y leads us to:

$$\begin{aligned} \frac{\partial y_i(t)}{\partial t} &= D \cdot \frac{y_{i+1}(t) + y_{i-1}(t) - 2 \cdot y_i(t)}{\delta x^2} + \\ &+ \frac{y_i(t)^2}{y_{i+N+1}(t)} - y_i(t) \qquad (2.14) \\ \frac{\partial y_{i+N+1}(t)}{\partial t} &= \frac{y_{i+N+2}(t) + y_{i+N}(t) - 2 \cdot y_{i+N+1}(t)}{\delta x^2} + \\ &+ \mu \cdot (y_i(t)^2 - y_{i+N+1}(t)) \qquad (2.15) \end{aligned}$$

These two equations are equivalent to:

$$\frac{\partial y_i(t)}{\partial t} = \begin{cases} D \cdot \frac{y_{i+1}(t)+y_{i-1}(t)-2\cdot y_i(t)}{\delta x^2} + \frac{y_i(t)^2}{y_{i+N+1}(t)} - y_i(t) & \text{if } 0 \leq i \leq N \\ \frac{y_{i+1}(t)+y_{i-1}(t)-2\cdot y_i(t)}{\delta x^2} + \mu \cdot (y_{i-(N+1)}(t)^2 - y_i(t)) & \text{if } N+1 \leq i \leq 2 \cdot N + 1 \end{cases} \qquad (2.16)$$

Equation (2.16) has now the form of equation (2.5), so the ODE solvers of MATLAB can be applied. If we would use a two- or three-dimensional model, everything becomes heavier in notation, but the principle stays the same.

In this project the equation system (2.3) - (2.4) was simulated for 2500 parameter choices of μ and D, namely the parameters $\in P$:

$$\begin{aligned} P &:= (\mu, D) \in M \times E \qquad (2.17) \\ M &:= \{0.06, 0.12, \ldots, 3\} \qquad (2.18) \\ E &:= \{0.008, 0.016, \ldots, 0.4\} \qquad (2.19) \end{aligned}$$

Because one has limitations in memory and processing time, one can not simulate a differential equation system for an infinite number of parameters, thats why we limit the number of simulations. We saved the whole course of the solution offered by MATLAB, in order to be able to extract more properties if needed. We need more or less 2.5 GByte memory for the 2500 simulated points. So the limitation to 2500 points comes from the huge memory needed. The range of the μ and D values, i. e. 0.06 to 3 and 0.008 to 0.4, are chosen such that every of the four solution regions appears as good as possible equally often, which can be seen from figure 2.2.

MATLAB offers different ODE Solvers: *ode45*, *ode113*, *ode15s*, *ode23*, *ode23s*, *ode23t* and *ode23tb*. Because reaction diffusion systems are known to be stiff differential equation systems [4], *ode45*, *ode113* and *ode23* are not appropriate. In our case the solver *ode23s* needs much more time than the solvers *ode15s*, *ode23t* and *ode23tb*. So the simulations are done with the ODE solvers *ode15s*, *ode23t* and *ode23tb*, but the differences are not very big.

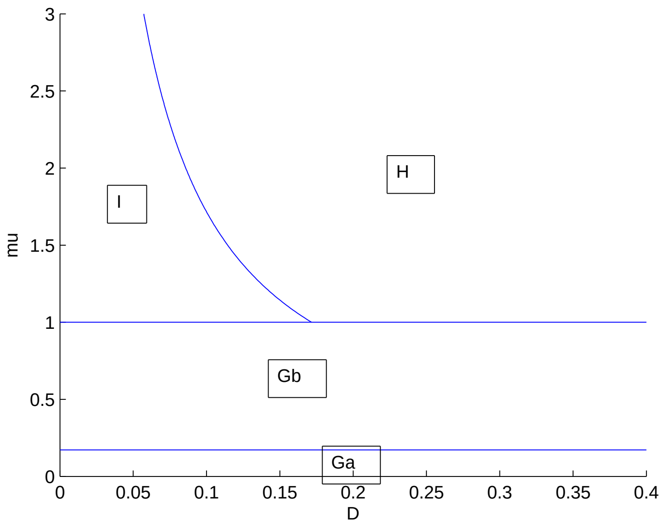


Figure 2.2: The boundaries in the (μ, D) plane

The other parameters in the MATLAB program, which one needs to determine before the simulation, are:

$endtime$	=	1000, how long the differential equation system is simulated.
N	=	100, the same N as used above.
$zufall$	=	0.1, the starting conditions are $a_{start}(x_i) = a_0 + zufall \cdot r_i$ and $h_{start}(x_i) = h_0$, where every r_i is a random variable uniformly distributed over the interval [0, 1].
δx	=	0.5, the same δx as used above.

These values were chosen, such that a compromise is reached between the reliability and the precision of the solution and the memory space used and the running time of the algorithm.

In the following table the properties extracted from a solution are listed and explained:

1. Property 1: The number of time points chosen by the ODE solver of MATLAB

2. Property 2: The time used for simulating the reaction diffusion system

3. Property 3: The maximum of the concentration $a(x_i, t)$ at time $t = endtime$

4. Property 4: The minimum of the concentration $a(x_i, t)$ at time $t = \textit{endtime}$
5. Property 5: The mean of the concentration $a(x_i, t)$ at time $t = \textit{endtime}$, i.e. $\frac{1}{N+1}\sum_{i=0}^{N} a(x_i, \textit{endtime})$
6. Property 6: The mean of the absolute value of Δa at time $t = \textit{endtime}$, i.e. $\frac{1}{N+1}\sum_{i=0}^{N} |\Delta a(x_i, \textit{endtime})|$
7. Property 7: The mean of the absolute value of $\frac{\partial a}{\partial t}$ at time $t = \textit{endtime}$, i.e. $\frac{1}{N+1}\sum_{i=0}^{N} |\frac{\partial a(x_i, \textit{endtime})}{\partial t}|$
8. Property 8: The maximum of the concentration $h(x_i, t)$ at time $t = \textit{endtime}$
9. Property 9: The minimum of the concentration $h(x_i, t)$ at time $t = \textit{endtime}$
10. Property 10: The mean of the concentration $h(x_i, t)$ at time $t = \textit{endtime}$
11. Property 11: The mean of the absolute value of Δh at time $t = \textit{endtime}$
12. Property 12: The mean of the absolute value of $\frac{\partial h}{\partial t}$ at time $t = \textit{endtime}$
13. Property 13 to 22: Properties 3 to 12 at the time $t =$ The time of the k-th time point, with k the rounded value of $\frac{r}{10}$ and r the number of time points chosen by the ODE solver of MATLAB
14. Property 23 to 32: Properties 3 to 12 at the time $t =$ The time of the k-th time point, with k the rounded value of $\frac{r}{2}$ and r the number of time points chosen by the ODE solver of MATLAB

We hope these 32 properties are enough to estimate the distance to the nearest boundary.

2.2 Ideas And Methods On How To Realize A Good Fitness Function

We are going to discuss the different approaches to build a fitness function. As explained above, the fitness function is split into two parts. In the first part the differential equation system is simulated and some properties are extracted. This part should be clear, so the focus of the following subsections is on the second part of the fitness function, the function from the *prop* vector to the output *distance*. If there are any simulation results shown, they are done with the MATLAB ODE solver *ode15s*.

In the following sections we use the notation: $prop_i$ is the property vector of the i-th simulated (μ, D) point and $distance_i$ the corresponding distance. $prop[k]$ is the k-th element of the $prop$ vector.

2.2.1 Classification

First we want to make a function, which is not a fitness function, but a classification function. In figure 2.2 we see the four different regions: G_a, G_b, H and I. The classification function has an input $prop$ and a $\{1, 2, 3, 4, 5\}$-valued output. The classification function should have the value 1, if the input $prop$ is typical for the solution region G_a, 2 for G_b, 3 for I, 4 for H and 5 if it is not typical for any solution region. The function should be very easy. In order to make such a function, we look at figures 2.3 - 2.5.

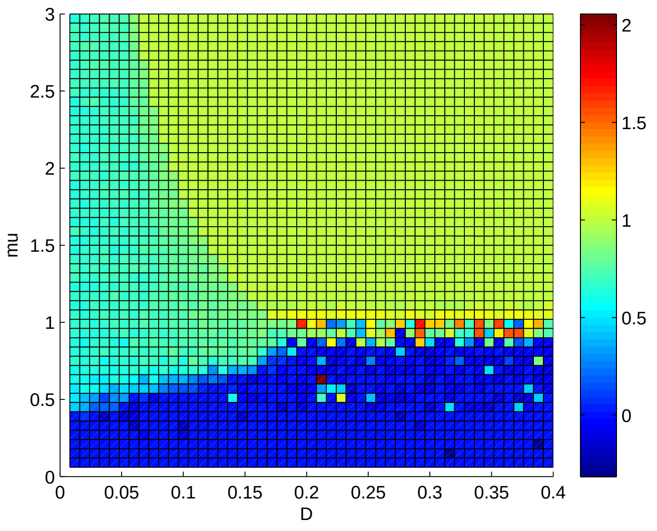


Figure 2.3: Property 5 drawn for the 2500 simulated (μ, D) points.

In figure 2.5, region G_a is different from the rest. So we can use property 29 to determine if a $prop$ vector belongs to region G_a with the following rule:

Rule 1: The classification function takes the value 1, if the 29-th element of the $prop$ vector is greater than 3.

Sure, the value 3 is a little bit arbitrary, we could also take 4. In figure 2.4, region H is different from the rest. By noting this, the following rule is used:

Rule 2: The classification function takes the value 4, if the 24-th element of the $prop$ vector is greater than 0.9 and rule 1 is not fulfilled.

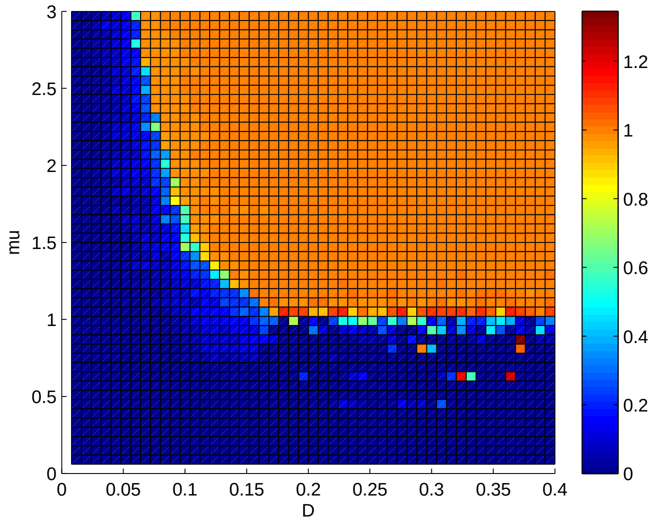


Figure 2.4: Property 24 drawn for the 2500 simulated (μ, D) points.

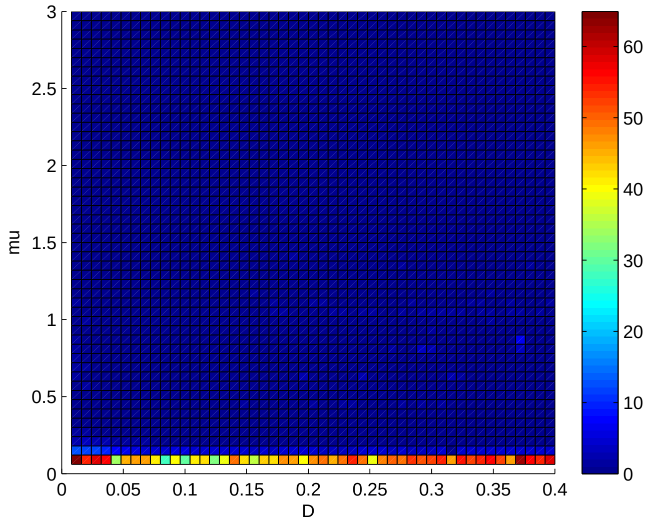


Figure 2.5: Property 29 drawn for the 2500 simulated (μ, D) points.

In rule 2, the additional part "...rule 1 is not fulfilled." is due to the wish, that just one rule is fulfilled. With figure 2.3, we can determine rules for classifying the remaining regions:

Rule 3: The classification function takes the value 2, if the 5-th element of the *prop* vector is smaller than 0.1 and rules 1 and 2 are not fulfilled.

Rule 4: The classification function takes the value 3, if the 5-th element of the *prop* vector is smaller than 0.9 and bigger than 0.1 and rules 1, 2 and 3 are not fulfilled.

Probably there are *prop* vectors, which do not fulfill any of the 4 rules. For these *prop* vectors, we say they are not typical for any solution region, so we assign to them the value 5:

Rule 5: The classification function takes the value 5, if the rules 1, 2, 3 and 4 are not fulfilled.

With these rules, we can draw the classification function for the simulated parameter points:

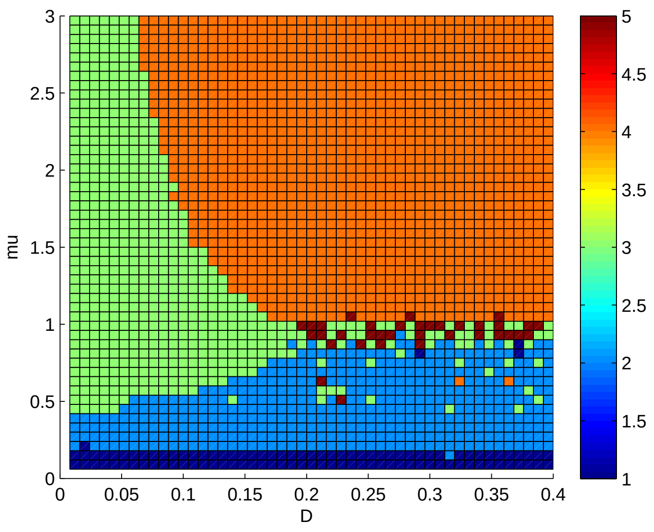


Figure 2.6: The classification function drawn for the simulated (μ, D) points.

This classification function is very easy, it depends just on 3 properties, but it is not perfect. Of course there are many different ways to make a classification function, but figure 2.6 indicates that the classification function used here is quite good. In chapter 3 an algorithm is introduced to find the boundaries between different solution regions with the help of this classification function.

2.2.2 Least Squares

Least Squares is a method to find a linear function from a row vector x to a scalar output y with minimum mean square error. If you have observed some input output relations $(x_i, y_i) 1 \leq i \leq M$ and you want to make a linear function, i.e. $\hat{y} = (1, x) \cdot h$ with h a constant column vector with one more element than x, then you can determine the vector h such that the mean square error $MSE = \frac{1}{M} \sum_{i=1}^{M} (y_i - (1, x_i) \cdot h)^2$ is minimized. The vector h is then given by equation (2.20), which is shown in [5]. The 1 in $\hat{y} = (1, x) \cdot h$ is to make the data mean free. With this 1 we have a smaller or equal MSE than without the 1.

$$\begin{aligned} A^T \cdot A \cdot h &= A^T \cdot g && (2.20) \\ \text{with } g &= (y_1, y_2, \ldots, y_M)^T && (2.21) \\ \text{and } A &= \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \\ 1 & x_m \end{bmatrix} && (2.22) \end{aligned}$$

If the matrix $A^T \cdot A$ is regular, then the h, which minimizes the MSE is uniquely determined by $h = (A^T \cdot A)^{-1} \cdot A^T \cdot g$. If $A^T \cdot A$ is singular, then there is no unique solution for h, but every solution has the same MSE.

The problem defined above is the same as for our fitness function. We have 2500 input output relations of our fitness function. If we assume that the fitness function is linear, then Least Squares can be applied, with $x_i = prop_i$ and $y_i = distance_i$. The estimated distance is then $\hat{distance} = prop \cdot h$ with h determined through equation (2.20). By doing this we get a minimum mean square error $MSE = 0.0074$. If we evaluate this function on the 2500 simulated (μ, D) points, we get figure 2.7.

This figure should look as similar as possible to figure 2.1. But some properties have to correspond in figures 2.1 and 2.7 and some others can differ without being critical. As an example it is very important, that there is a minimum at the location of the boundaries or very close to the boundaries. If this is not the case, then the task of finding a minimum of the fitness function and the task of finding the boundaries are not the same. This is not perfectly the case, but figure 2.7 has a tendency to be minimal at the boundaries. Another important property, which has to correspond in both figures, is, that there are no plateaus. If there were plateaus, it could be, that an algorithm stucks in a plateau. But there is a plateau in the solution region H. So this approach is not perfect.

If we want to decrease the MSE further, we can do a trick. Instead of using the *prop* vector for the x vector, we could additionally use transformations of the *prop* vector, for example $x = (\cos(\pi \cdot prop), \sin(\pi \cdot prop),$

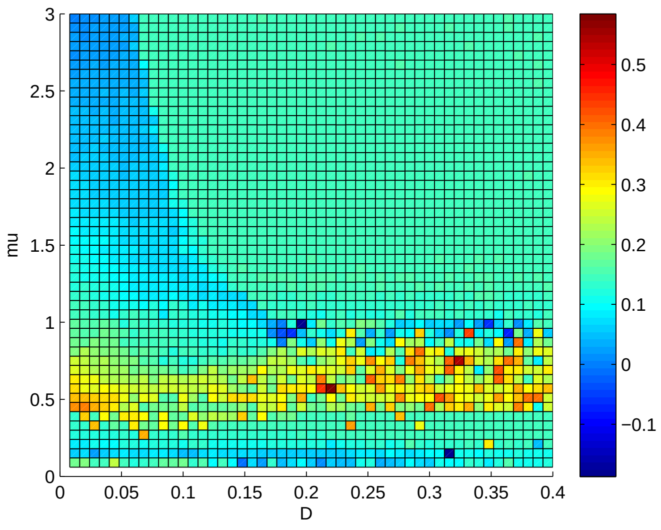


Figure 2.7: The linear function $\widehat{distance} = prop \cdot h$ drawn for the simulated $(\mu,\ D)$ points.

$\ln|prop|, prop^2, prop)$. In this and the following expressions, the functions cos, sin, $\ln||$ or $()^2$ with a vector as input argument have to be interpreted as component wise functions, i.e. $\cos(\pi \cdot prop) = (\cos(\pi \cdot prop[1]), \cos(\pi \cdot prop[2]), \ldots, \cos(\pi \cdot prop[32]))$. So we get a vector x with 160 elements. As a motivation for this trick, we can look at the function $y = \cos(\pi \cdot x)$, x a scalar. If we want to use a linear function for estimating y, i.e. $y = (1, x) \cdot h$, then we never bring the MSE to zero. If we use $y = (1, x, \cos(\pi \cdot x)) \cdot h$, we bring the MSE to zero with $h = (0, 0, 1)^T$.

We can calculate h for $x = (\cos(\pi \cdot prop), \sin \pi \cdot prop, \ln|prop|, prop^2, prop)$ by using (2.20). If we evaluate the function $\hat{distance} = (\cos(\pi \cdot prop), \sin(\pi \cdot prop), \ln|prop|, prop^2, prop) \cdot h$ on the simulated $(\mu,\ D)$ points, we get figure 2.8 and a minimum mean square error $MSE = 0.0022$.

In this figure we see an improvement. The plateau in the H region is gone and the minimum on the boundaries is more prominent. So we would expect, that this fitness function is better than the fitness function without transformations. The problems according to such interpretations are the points in the $(\mu,\ D)$ space for which we not simulated the reaction diffusion system (2.3) - (2.3). Because with Least Squares, we just minimize the MSE for the known input output pairs, we not consider the MSE for other points. So in general our fitness function could have a MSE for the known input output pairs of zero, even so it has a very big MSE for other points. In this subsection we hope that this is not the case and in the next subsection 2.2.3 we give a possibility to deal with this problem.

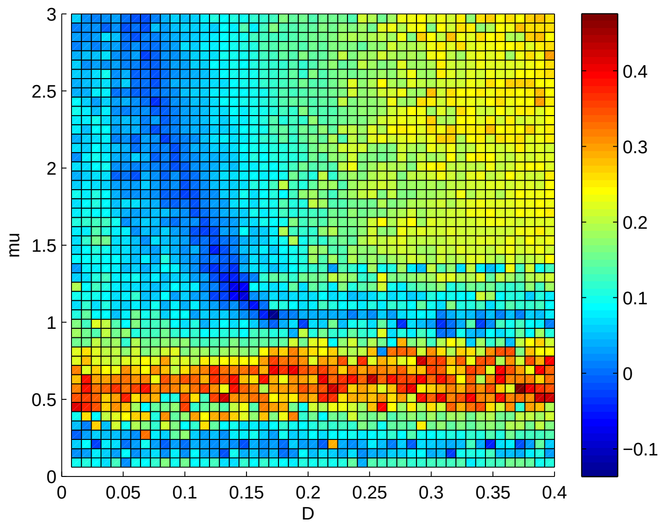


Figure 2.8: The linear function $\widehat{distance} = (\cos(\pi \cdot prop), \sin(\pi \cdot prop), \ln|prop|, prop^2, prop) \cdot h$ drawn for the simulated (μ, D) points.

The linear function with transformations is quite complicated. The h vector has 161 elements. If two columns of the matrix A are more or less proportional, we could leave out one of the columns without increasing the *MSE*. Therefore we can decrease the number of elements of h and hence the complexity. A measure for the proportionality of two columns is the correlation coefficient. The correlation coefficient c of two data columns f and g is defined as:

$$c(f,g) = \frac{\sum_{k=1}^{M}(f[k]-\overline{f})\cdot(g[k]-\overline{g})}{\sqrt{\sum_{k=1}^{M}(f[k]-\overline{f})^2}\cdot\sqrt{\sum_{k=1}^{M}(g[k]-\overline{g})^2}} \tag{2.23}$$

$$\text{with } \overline{f} = \frac{1}{M}\cdot\sum_{k=1}^{M} f[k] \tag{2.24}$$

$$\text{and } \overline{g} = \frac{1}{M}\cdot\sum_{k=1}^{M} g[k] \tag{2.25}$$

c is always between -1 and 1. If c is close to 1 or -1, f and g are quite proportional. If c is close to zero, f and g are not proportional.

A possible approach to decrease the complexity is the following procedure:

1. Determine the correlation coefficient c for all possible pairs of columns

of the matrix A (the first column of A is not considered).

2. While the maximum of the absolute value of all c is bigger than a threshold t, do points 3 and 4
3. Determine the pair of columns of A with the biggest absolute value of c and the new A is the old A without one of the two columns.
4. Determine the correlation coefficient c for all possible pairs of columns of the matrix A

So we eliminate bit by bit the columns of A which have a big $|c|$ with another column until there are no pairs of A with a $|c| > t$. We use this approach with a threshold $t = 0.4$. The number 0.4 was chosen such that more or less the same number of columns remained as the number of elements of the *prop* vector. With the procedure we decrease the number of columns of A from 161 to 32, which is a factor of about 5. The remaining columns are: $\cos(\pi \cdot prop[1])$, $\sin(\pi \cdot prop[1])$, $\sin(\pi \cdot prop[7])$, $\sin(\pi \cdot prop[13])$, $\sin(\pi \cdot prop[14])$, $\sin(\pi \cdot prop[16])$, $\sin(\pi \cdot prop[17])$, $\sin(\pi \cdot prop[19])$, $\sin(\pi \cdot prop[20])$, $\sin(\pi \cdot prop[21])$, $\sin(\pi \cdot prop[22])$, $\sin(\pi \cdot prop[23])$, $\sin(\pi \cdot prop[24])$, $\sin(\pi \cdot prop[26])$, $\sin(\pi \cdot prop[27])$, $\sin(\pi \cdot prop[28])$, $\sin(\pi \cdot prop[30])$, $\sin(\pi \cdot prop[31])$, $\sin(\pi \cdot prop[32])$, $\ln|prop[14]|$, $\ln|prop[22]|$, $prop[7]$, $prop[10]$, $prop[12]$, $prop[17]$, $prop[21]$, $prop[22]$, $prop[24]$, $prop[27]$, $prop[30]$, $prop[32]$. Now we can compute the h vector which minimizes the *MSE* for these 31 columns plus the all one column. If we evaluate this function on the simulated (μ, D) points, we get figure 2.9 and a minimum mean square error $MSE = 0.0073$.

This function is not much better than the first Least Squares function of figure 2.7. The *MSE* is more or less equal to the *MSE* of the function of figure 2.7. The plateau is still there, but the boundaries seems to be a little bit better than before.

The last function presented here, should be a very easy one. It should have the all one vector plus four or less out of the 160 transformed properties. To find the four transformed properties which have the minimum *MSE*, we could calculate for every 4-tuple out of the 160 transformed properties the *MSE* and choose the 4-tuple with the minimal *MSE*. There exists 26294360 4-tuples, which are too much. So we have to use a different approach. The idea for this approach was taken from [6]:

1. Start with none of the 160 transformed properties, so $C = \{\}$
2. Do the next steps, while the number of elements in C is ≤ 4
3. Include the transformed property to the current model for which the actual linear function has the minimal *MSE*, so $C_{new} = C_{old} \cup$ (The chosen transformed property)

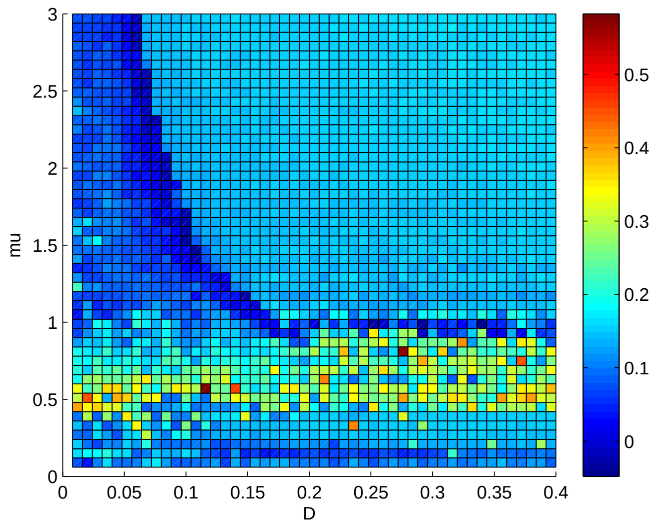


Figure 2.9: The linear function with transformations, but just 32 of the 161 columns of A are used. You see this function drawn for the simulated (μ, D) points.

So in every run, we increase the number of used transformed properties by one. We do not need to calculate the MSE 26294360 times. If we do this, the four remaining transformed properties are: $\cos(\pi \cdot prop[19])$, $\sin(\pi \cdot prop[24])$, $ln|prop[29]|$ and $ln|prop[31]|$

If we evaluate this function on the simulated (μ, D) points, we get figure 2.10 and a minimum mean square error $MSE = 0.0077$.

Although this function uses just 4 of the 160 transformed properties it is quite similar to the function of figure 2.9. The boundary between the regions H and G_b and the boundary between the regions G_a and G_b seems to be less prominent than in figure 2.9.

2.2.3 Neural Networks

A Neural Network (NN) is also a possible method for finding a function with some known input output relations. An NN consists of p inputs, J outputs and one hidden layer. In general a NN can have more than one hidden layer, but we focus here on NN with one hidden layer. The number of outputs J is in this project always one, namely the distance to the nearest boundary. The inputs are the elements of the $prop$ vector, sometimes with transformations and sometimes not all of them are used. A NN with one hidden layer and one output can be written as:

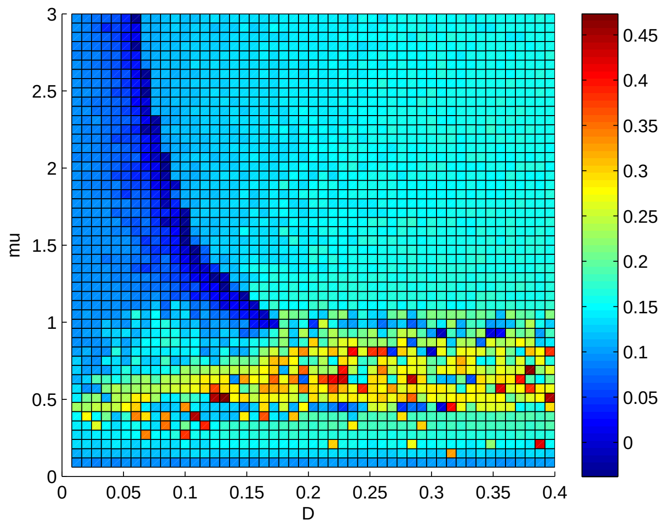


Figure 2.10: The linear function with transformations, but just 5 of the 161 columns of A are used. You see this function drawn for the simulated (μ, D) points.

$$g(x) = f_0(\alpha + \sum_{h=1}^{q} w_h \cdot \phi(\beta_h + \sum_{j=1}^{p} w_{jh} \cdot x_j)) \tag{2.26}$$

A often used function for ϕ is the sigmoid function:

$$\phi(x) = \frac{e^x}{1 + e^x} \tag{2.27}$$

The function $f_0(.)$ is typically the identity function, but for both you can also use other functions. The α, β_h, w_h, w_{jh} are constants. An illustration of an NN is shown in figure 2.11.

Our goal is to have a NN, which is similar to the distance function from figure 2.1. The task is to adjust the constants α_1, α_2, w_h, $w_j h$, such that the *MSE* of the given input and output pairs is small. This can be done using the nftool of MATLAB. There you have a supplementary feature. You can use some of the known input output pairs as testing or validation data. So you can for example use 1500 data points for training the NN, 500 for testing the created NN and the remaining 500 data points for validation. With this possibility to test the used NN, we are able to measure how close the NN is to the wished function for points which are not used to train the NN. So we can decide, whether the NN just memorizes the training data or it really learns the wished function. A further parameter of a NN is the number of hidden nodes in the hidden layer. The letter q in equation (2.26) denotes

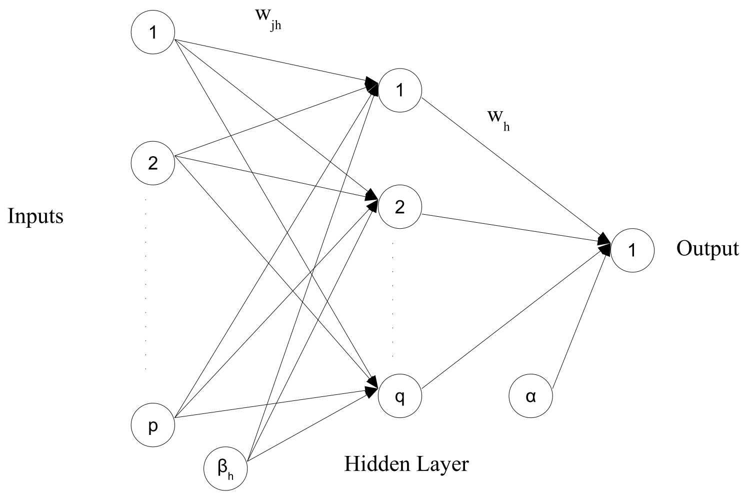


Figure 2.11: An illustration of a NN with one hidden layer, one output and p inputs.

the number of hidden nodes. Often the more hidden nodes one uses, the more probably the NN just memorizes the training data. The less hidden nodes one uses, the bigger is the *MSE* on the training data. So we have to find a tradeoff. In this project, we started with a little q and increased q until the *MSE* on the training data and the *MSE* on the testing data began to diverge. Then the biggest q for which the two *MSE* did not diverge, was taken.

If you would like to have more detailed informations about NN, see [5]. The nftool is explained in the MATLAB Help. One remark regarding the nftool tool should be made here. The nftool uses in the training step some random numbers, so the results presented in this subsection can not be reproduced exactly.

Now we are ready to create the first NN by using the nftool. Input data are the 32 properties of the 2500 simulated points. Target data are the distances to the nearest boundaries, which you see in figure 2.1. 20% of the data are used for testing and 20% for validation. The number of hidden nodes is 20. The *MSE* of the whole 2500 data points is 0.0054. If we evaluate the NN on the 2500 simulated points, we get figure 2.12.

The problem of this NN is the plateau in the H region. The rest seems to be acceptable. An advantage of the NN presented here are the fact, that they are tested during the training. So we expect, that the NN can predict new points better than the Least Squares functions.

We can build a second NN. Input data are the 160 transformed properties of the 2500 simulated points. Target data are the distances to the nearest boundaries, which you see in figure 2.1. 20% of the data are used for testing

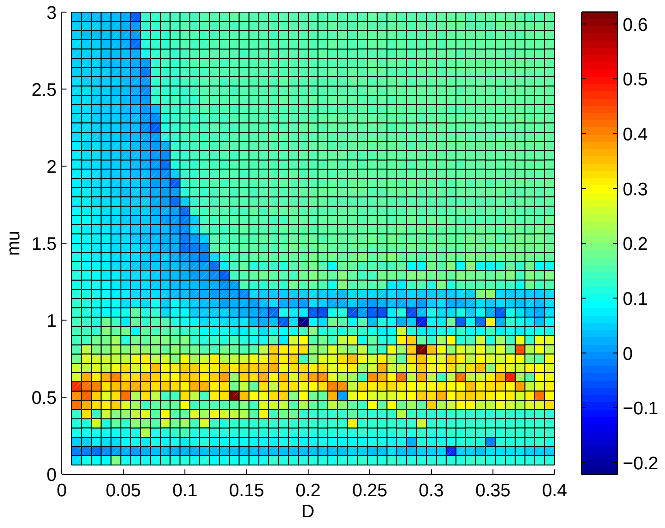


Figure 2.12: The NN drawn for the simulated (μ, D) points.

and 20% for validation. The number of hidden nodes is 10. The *MSE* of the whole 2500 data points is 0.0021. If we evaluate the NN on the 2500 simulated points, we get figure 2.13.

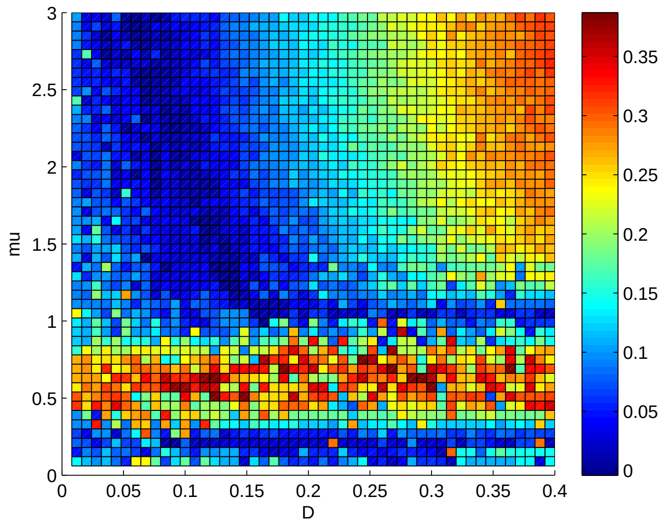


Figure 2.13: The NN with all 160 transformed properties drawn for the simulated (μ, D) points.

This NN has a smaller *MSE* and theres no plateau in the *H* region. On the boundaries the NN is minimal.

The third and the last NN used here, is an NN, for which we use the same 31 transformed properties as in the Least Squares function of figure 2.9. 20% of the data are used for testing and 20% for validation. The number of hidden nodes is 15. The *MSE* of the whole 2500 data points is 0.0027. If we evaluate the NN on the 2500 simulated points, we get figure 2.13.

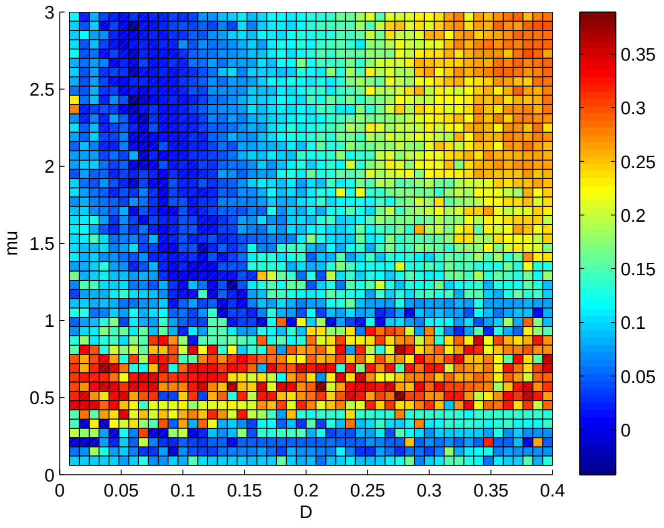


Figure 2.14: The NN with 31 of the 160 transformed properties drawn for the simulated (μ, D) points.

This NN has no plateau in the H region as well. The minimum on the boundaries is not so prominent as for the NN of figure 2.13. Between the H and the I region the minimum is quite large and not so deep, which could be a problem.

2.2.4 CART

CART is an abbreviation for Classification and Regression Tree. The model function for CART is

$$g_{tree}(x) = \sum_{r=1}^{R} \beta_r 1_{[x \in A_r]} \tag{2.28}$$

where $P = \{A_1, A_2, \ldots, A_R\}$ is a partition of $\mathbb{R}^p$. The function $1_{[x \in A_r]}$ is one if $x \in A_r$ and zero if $x \notin A_r$. So the function $g_{tree}(x)$ is piecewise constant. The sets A_r are restricted to be axes parallel rectangles. If this is the case, we can draw a tree, like for example in figure 2.15. The trees used in this project are regression trees and not classification trees.

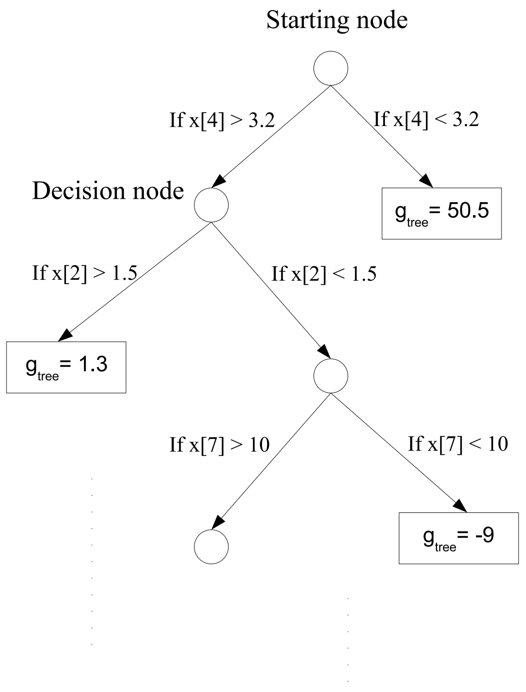


Figure 2.15: An example of a tree. One begins on the starting node and goes through the tree until one arrives at a rectangle. The value in this rectangle is then the value of the function $g_{tree}(x)$.

One can interpret a CART much better than a NN or a Least Squares function.

With MATLAB one can build a CART by using the function classregtree. You have to give some input output pairs of the wished function. Optional you can handle a parameter called splitmin. This parameter is very useful, if you want to limit the complexity of the tree. Because the function classregtree divides a node further only if there exists in this node more observation points than the number splitmin. So in every decision node are at least splitmin observations.

For more details about CART, please look at [6]. If you are interested in the function classregtree, please look at the Help of MATLAB.

Now we can build the first CART with the MATLAB function classregtree. The input output pairs are of course the pairs *prop* and *distance* for the 2500 simulated points. The optional argument splitmin is not used[1]. The result is shown in figure 2.16.

[1]The default value used by MATLAB is 10

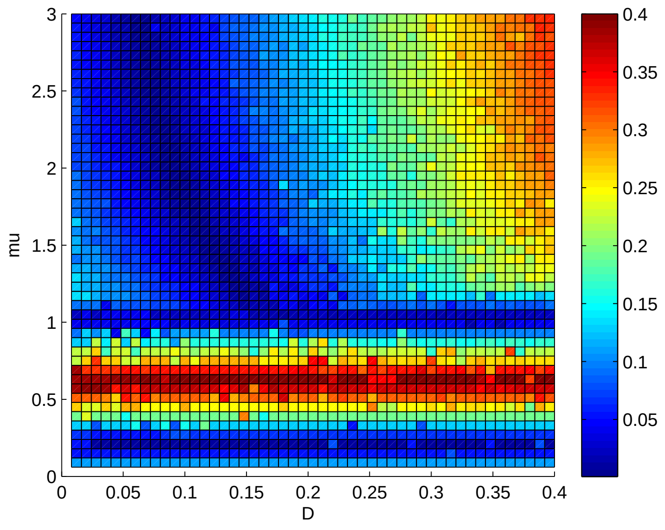


Figure 2.16: The CART, built with the function classregtree of MATLAB, drawn for the simulated (μ, D) points.

The *MSE* of the CART is $1.7741 \cdot 10^{-4}$, which is very small compared to other *MSE*. This is not very surprising, because essentially with a CART you can force the *MSE* to zero by dividing the regions A_r further[2]. This function is quite perfect. A probably big disadvantage of this function, is the behaviour on new samples. It could be, that this CART just memorizes the 2500 input output pairs. This CART has 421 decision nodes, which is too much to do interpretations. It would be nice to have a tree with less decision nodes. To find such a tree, we can need the optional parameter splitmin. If we set splitmin to 250, we get 18 decision nodes. This can be interpreted. This CART has an *MSE* of 0.0032 and it is shown in figure 2.17.

There is no plateau in the H region and the function is quite small on the boundaries. A picture of the tree is shown in figure 2.18.

We can interpret the tree of figure 2.18. First we show the same tree, but instead of writing numbers to the points, we write the region where the numbers appear and instead of writing the numbers of the properties, we write abbreviations:

1. Abbreviation of property 1: # time points

2. Abbreviation of property 2: sim. time

[2]If there are two input output pairs with the same input vector but a different output it is impossible to have an *MSE* of zero.

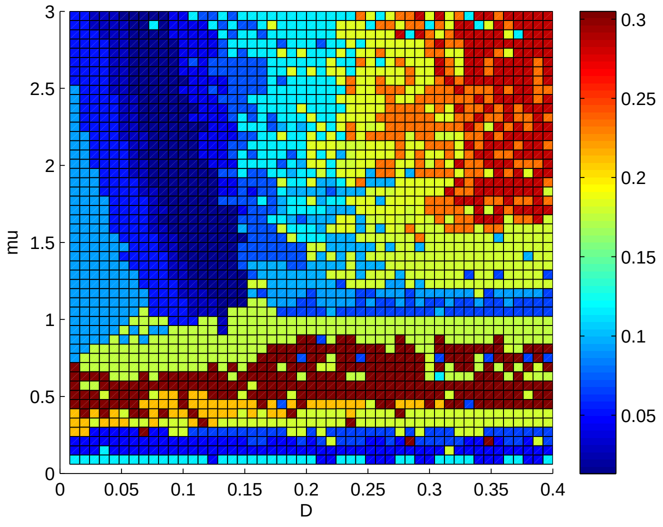


Figure 2.17: The CART, with the optional parameter splitmin set to 250, drawn for the simulated (μ, D) points.

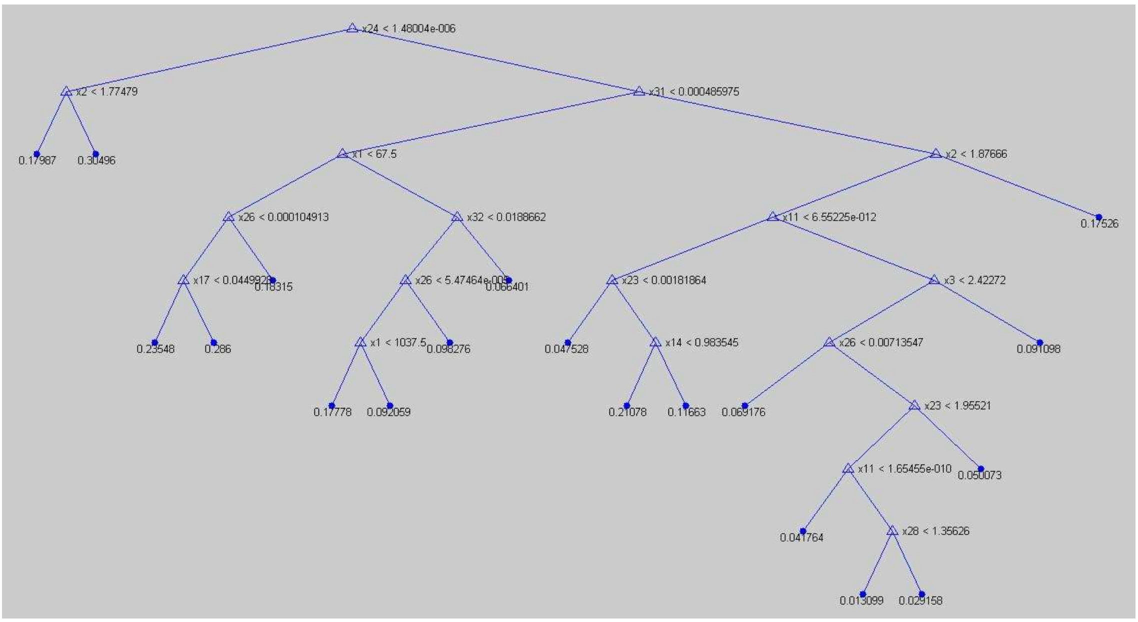


Figure 2.18: The tree of the CART, with the optional parameter splitmin set to 250. The triangles are the decision nodes and the points have the same meaning as the rectangles in figure 2.15. One goes to the left if the condition is true and right if the condition is false.

3. Abbreviation of properties 3 to 32: Every abbreviation consists of three terms: *what*, *of what* and *when*. *what* can be one of the following: {*max, min, mean*}. *of what* can be one of the following: {a, h, $|\Delta a|$, $|\Delta h|$, $|da/dt|$, $|dh/dt|$} and *when* can be one of the following: {*end, start, middle*}, where *end* refers to the time of properties 3 to 12, *start* to the time of properties 13 to 22 and *middle* to the time of properties 23 to 32. For example *mean*, $|dh/dt|$, *start* corresponds to property 22.

You see this in figure 2.19. Sometimes this is difficult, because some numbers occur in more than one region or are near a boundary.

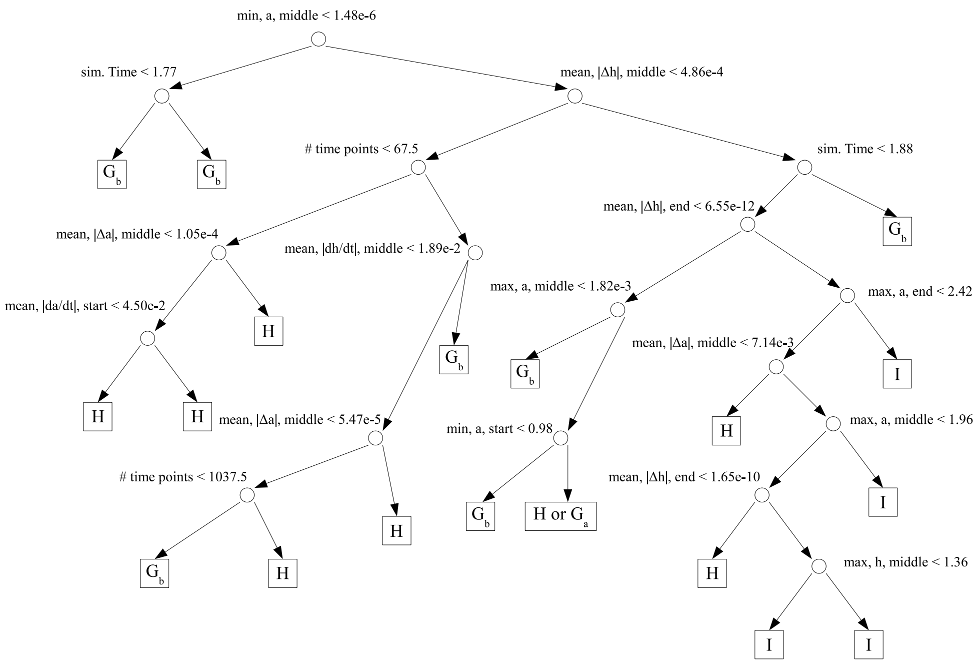


Figure 2.19: The tree of the CART, with the optional parameter splitmin set to 250. Instead of writing numbers, we write regions and we use abbreviations for the properties.

The following interpretation uses only tendencies, so it is not exact. To belong to the I region, it needs a diffusion, $\Delta a > 0$ and $\Delta h > 0$. To belong to the region G_b, a rather small value of the maximum or minimum of the concentration a has to appear. The H region appears in many different parts of the tree. We can extract two rules. A point belongs to the H region, if it has a small diffusion, $\Delta a \approx 0$ and $\Delta h \approx 0$ or if the number of time points (or simulation time) is small. This is reasonable, because in the H region there should be no diffusion. We see also, that one can decide whether a point in the H region is far away from the boundaries or not. The less time points needed and the less diffusion one has, the farer apart one is from the boundary. This is quite important, because the H region is often big and you need a criterion where a different solution region is located.

2.3 500 Additional Points For Testing

In the last tree subsections, we built 9 functions. Now we want to test these functions. We simulated another 500 points with μ and D uniformly drawn from the intervals [0, 6] and [0, 0.8]. Because we know the distance to the nearest boundary for every point in the (μ, D) space, we can compute the *MSE* for these 500 points for every function. The results are shown in table 2.1 together with the *MSE* on the 2500 training points and the mentioned disadvantages.

Function:	Abbreviation	MSE_{500}	MSE_{w105}	MSE_{2500}	Disadvantages
Figure 2.7	LS1	$9.4182 \cdot 10^3$	0.0707	0.0047	P, B and T
Figure 2.8	LS2	$4.9287 \cdot 10^{13}$	0.0612	0.0022	T
Figure 2.9	LS3	$1.0060 \cdot 10^3$	0.0641	0.0073	P and T
Figure 2.10	LS4	0.0681	0.0682	0.0077	P and T
Figure 2.12	NN1	0.0657	0.0656	0.0054	P
Figure 2.13	NN2	0.0326	0.0327	0.0021	No
Figure 2.14	NN3	0.0291	0.0291	0.0027	No
Figure 2.16	CART1	0.0306	0.0306	$1.7741 \cdot 10^{-4}$	T
Figure 2.17	CART2	0.0413	0.0414	0.0032	T

Table 2.1: A summary with different *MSE* values for the 9 fitness functions. The abbreviations in column Disadvantages mean: P = Plateau somewhere, B = The minimum on the boundaries not reached, T = No testing during training

The column MSE_{w105} is the *MSE* without the 105-th simulated point. This is showed due to the fact, that some functions have trouble with this 105-th point and has a very big squared error. The μ and D coordinates of this point are: $\mu = 3.383 \cdot 10^{-4}$ and $D = 0.355$. So the point lies in the G_a region. A possible explanation for the big error induced by this point, are the small number of points simulated in the G_a region. Now we can make a ranking list according to the MSE_{500}, MSE_{w105} or MSE_{2500} of table 2.1, which is shown in table 2.2.

The functions CART1, NN3 and NN2 are at least two times under the top three and never under the worst three. The functions NN1, LS1, LS3 and LS4 are at least two times under the worst three and never under the top three. The disadvantages in table 2.1 strengthen the impression. In the following chapters we will no longer consider the functions NN1, LS1, LS3 and LS4.

Rank:	MSE_{500}	MSE_{w105}	MSE_{2500}
1.	NN3	NN3	CART1
2.	CART1	CART1	NN2
3.	NN2	NN2	LS2
4.	CART2	CART2	NN3
5.	NN1	LS2	CART2
6.	LS4	LS3	LS1
7.	LS3	NN1	NN1
8.	LS1	LS4	LS3
9.	LS2	LS1	LS4

Table 2.2: A ranking list of the 9 fitness functions for MSE_{500}, MSE_{w105} and MSE_{2500}

Chapter 3

Evolutionary Algorithm

We will briefly discuss the idea behind a popular EA, the so called Evolution Strategy with Covariance Matrix Adaption (CMA). This EA will be used in the following chapters. The second section of this chapter treats an algorithm, which is used combined with the classification function from subsection 2.2.1. This algorithm is not an EA, we use it to have an alternative approach.

3.1 CMA

The CMA can be used to minimize (or maximize) a function f from $\mathbb{R}^n$ to $\mathbb{R}$. In the case of a fitness function from chapter 2, f is a function from $(\mu, D) \in \mathbb{R}^2$ to $\mathit{distance} \in \mathbb{R}$. The idea behind CMA is to create new points by sampling a multivariate normal distribution. A normal distribution is uniquely determined through its mean and covariance matrix. The covariance matrix is divided into two parts, the normalized covariance matrix and the step size. The basic equation for sampling new points, for generation number g, is:

$$x_k^{(g+1)} \sim \mathcal{N}(m^{(g)}, {\sigma^{(g)}}^2 C^{(g)}) \tag{3.1}$$

$x_k^{(g+1)}$ denote the new points created by sampling a normal distribution with mean $m^{(g)}$ and covariance matrix ${\sigma^{(g)}}^2 C^{(g)}$. ${\sigma^{(g)}}^2$ denotes the step size parameter and $C^{(g)}$ the normalized covariance matrix. The CMA chooses the mean $m^{(g)}$ as a weighted sum of the old points $x_i^{(g)}$, whereat the better points get bigger weights. The step size and normalized covariance matrix are adjusted new for every generation number g, such that it is more likely to create new points with a small fitness value. For example, we have two points in the (μ, D) space (2, 0.4) and (3, 0.8) with corresponding fitness values 1 and 5. Then the CMA would create new points by sampling a normal distribution with iso-probability contours shown in figure 3.1.

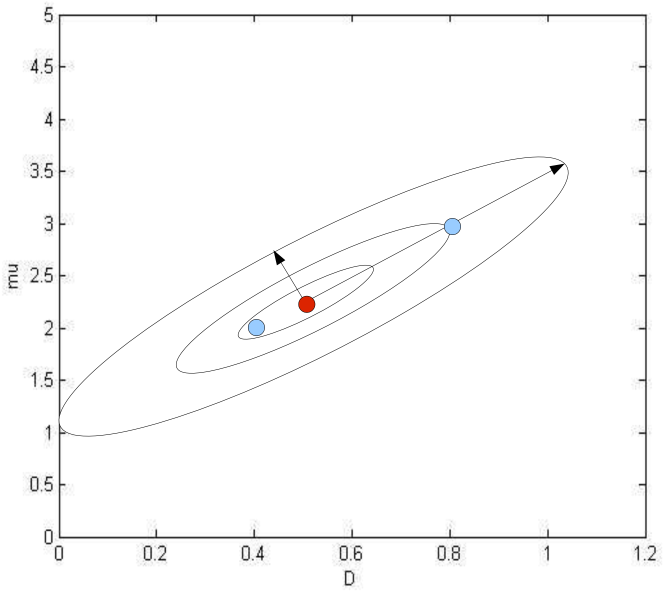


Figure 3.1: Iso-probability contours of a normal distribution possibly chosen for the example with two points (2, 0.4), (3, 0.8) and their fitness values 1 and 5. The red point is the mean of the normal distribution. The two arrows are given by the covariance matrix and determine the shape of the iso-probability contours.

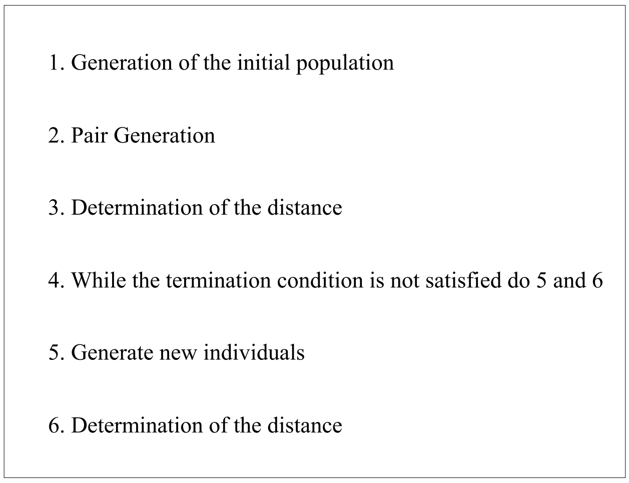
The mean of the normal distribution is closer to the point (2, 0.4), because this point has a smaller fitness value and therefore a bigger weight. The shape of the iso-probability contours and hence the prefered search direction are determined by the normalized covariance matrix. We see in figure 3.1, that the prefered search direction is the connecting line between the two points. The step size has no influence on the shape of the iso-probability contours, but it determines how close they are or in other words how big the search space is.

For further explanations and details, see [7] or [8].

3.2 An Algorithm For The Classification Function

We call the algorithm explained here classification algorithm. The structure of this algorithm is shown in figure 3.2.

For simplicity, we discuss the classification algorithm only for the (μ, D) problem, but one can use the algorithm also for other problems. Before we explain the algorithm, we would like to introduce the needed parameters of the classification algorithm: $r_\mu, r_D \in \{r \in \mathbb{R} : r > 0\}$, *initpopsize* $\in$



1. Generation of the initial population
2. Pair Generation
3. Determination of the distance
4. While the termination condition is not satisfied do 5 and 6
5. Generate new individuals
6. Determination of the distance

Figure 3.2: The structure of the classification algorithm

$\{n \in \mathbb{N} : n > 1\}$, *tolerance* $\in \{r \in \mathbb{R} : r > 0\}$. The following list discusses the steps in figure 3.2:

1. We create *initpopsize* points in the $(\mu,\ D)$ space randomly. These points $(\mu_i,\ D_i)$ are drawn uniformly from the subspace $\{[0, r_\mu] \times [0, r_D]\}$.

2. We simulate for every point $(\mu_i,\ D_i)$ the reaction diffusion system (2.3) - (2.4), extract the *prop* vector and plug it into the classification function. Now we try to find pairs of the *initpopsize* points with different classification values. This we do iteratively by choosing randomly two points out of the initial points and compare their classification value. If they have a different classification value, then we save the pair and remove them from the initial population. If they have the same classification value, nothing happens. This we repeat until not more than one point remains or all the remaining points have the same classification value. We call the set of the found pairs as P.

3. For every pair $(\mu_k,\ D_k),\ (\mu_j,\ D_j) \in P$, we calculate the euclidean distance $d(\mu_k, D_k, \mu_j, D_j) = \sqrt{(\mu_k - \mu_j)^2 + (D_k - D_j)^2}$. If the euclidean distance of a pair is smaller than *tolerance*, the mean of this pair $mean = \left(\frac{\mu_k+\mu_j}{2}, \frac{D_k+D_j}{2}\right)$ is saved, interpreted as a point on the boundary and $P_{new} = P_{old}$without this pair. If the euclidean distance of a pair is not smaller than *tolerance*, nothing happens ($P_{new} = P_{old}$).

4. The termination condition is: The number of pairs $\in P$ is zero.

5. We substitute every pair $(\mu_k,\ D_k)$, $(\mu_j,\ D_j) \in P$ by a new pair $(\mu_l,\ D_l)$, $(\mu_m,\ D_m)$. First we compute the difference vector *diff* $= (\mu_k - \mu_j, D_k - D_j)$. Then we create two random numbers $rand_1$ and $rand_2$ uniformly drawn from the interval [0, 0.5]. We compute two new points in the $(\mu,\ D)$ space: $point_1 = (\mu_k, D_k) - rand_1 \cdot diff$ and $point_2 = (\mu_j, D_j) + rand_2 \cdot diff$. For $point_1$ and $point_2$ we simulate the reaction diffusion system, extract the *prop* vector and plug it into the classification function. If the classification value of the points $point_1$ and $point_2$ have different values, then $(\mu_l, D_l) = point_1$ and $(\mu_m, D_m) = point_2$. Else if the classification value of the points $(\mu_k,\ D_k)$ and $point_2$ have different values, then $(\mu_l, D_l) = (\mu_k, D_k)$ and $(\mu_m, D_m) = point_2$. Else if the classification value of the points $point_1$ and $(\mu_j,\ D_j)$ have different values, then $(\mu_l, D_l) = point_2$ and $(\mu_m, D_m) = (\mu_j, D_j)$. Else $(\mu_l, D_l) = (\mu_k, D_k)$ and $(\mu_m, D_m) = (\mu_j, D_j)$. So we try to make new pairs, which are closer to each other than before, but only if they still have a different classification value. For an illustration, look at figure 3.3.

6. Here we do the same as in step 3.

Although the classification algorithm is not an EA, there are some parallels.

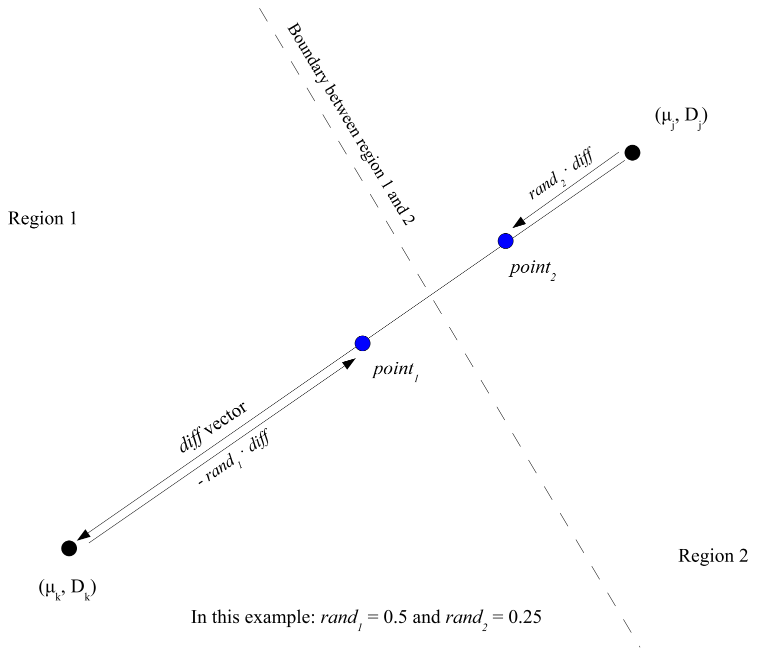


Figure 3.3: An illustration of the fifth point of the classification algorithm is shown. The difference vector *diff* goes from the point $(\mu_j,\ D_j)$ to the point $(\mu_k,\ D_k)$. The point $(\mu_k,\ D_k)$ lies in region 1 and the point $(\mu_j,\ D_j)$ lies in region 2. $point_1$ lies in region 1 and $point_2$ in region 2, so they lie in different regions. Therefore $(\mu_l, D_l) = point_1$ and $(\mu_m, D_m) = point_2$.

Chapter 4

CMA Applied To The Simple System

We apply the CMA to the reaction diffusion system of equations (2.3) - (2.4). We discuss the behaviour of the different fitness functions of chapter 2. Because we know the location of the boundaries, we can evaluate the performance of the different fitness functions. Then we look at the robustness of the fitness functions with respect to little changes on the system and the simulation parameters.

4.1 Normal System

In this subsection we run the program cmaes.m[1] for every of the 9 fitness functions 10 times. The program cmaes.m takes 4 parameters: the fitness function, an initial starting point, initial coordinate wise search standard deviation and options. Because $D < 0$ or $\mu < 0$ makes no sense, we only allow points with $D > 0$ and $\mu > 0$. This we do by setting the fitness value of all points with $D < 0$ or $\mu < 0$ to 10. The initial starting point is always a vector $\in \mathbb{R}^2$ uniformly distributed over the range [0, 6] for the first component μ and uniformly distributed over the range [0, 0.8] for the second component D. In the description of the cmaes.m is written, that setting the initial coordinate wise search standard deviation to one third of the initial search region is appropriate. So we set it to 2 for μ and to 0.3 for D. Because the expected minimum of the fitness functions is zero, we set the option StopFitness to a very little value greater zero. We choose StopFitness $= 10^{-8}$ for the functions LS2, NN2 and NN3, which gives quite good results. So the program cmaes.m stops if it found a point with fitness value smaller than 10^{-8}. For the CART functions 10^{-8} is never reached, so it makes no sense to set the option StopFitness to 10^{-8}. For the function

[1]This program can be downloaded from the link: http://www.bionik.tu-berlin.de/user/niko/cmaes_inmatlab.html

CART1, we set the option StopFitness to $7 \cdot 10^{-2}$ and for CART2 to $2 \cdot 10^{-2}$. It could be, that it takes very long time to find a point with fitness value less than 10^{-8}, $7 \cdot 10^{-2}$ respectively $2 \cdot 10^{-2}$, so we set the option MaxFunEvals to 1000. This means, that after 1000 fitness function evaluations the algorithm stops. As mentioned above, we want to evaluate the performances. This we do graphically, by drawing the resulting points together with the boundaries and additionally we calculate the error, which is the mean of the distances to the nearest boundary:

$$\begin{aligned} error &= \frac{1}{M}\sum_{i=1}^{M} \min_{(\mu_b, D_b)\in B} \sqrt{(\mu_i - \mu_b)^2 + (D_i - D_b)^2} && (4.1)\\ \text{with } B &= (\mu_b, D_b) \in Boundaries, \text{ and } (\mu_i,\, D_i) \text{ are the points found} \\ & \quad \text{by the cmaes.m} && (4.2)\\ \text{and } M &= \text{The number of points found by the program} \\ & \quad \text{cmaes.m (mostly 10)} && (4.3) \end{aligned}$$

The points found by the program cmaes.m are drawn together with the boundaries in figures 4.1- 4.5. The error is summarized in table 4.1.

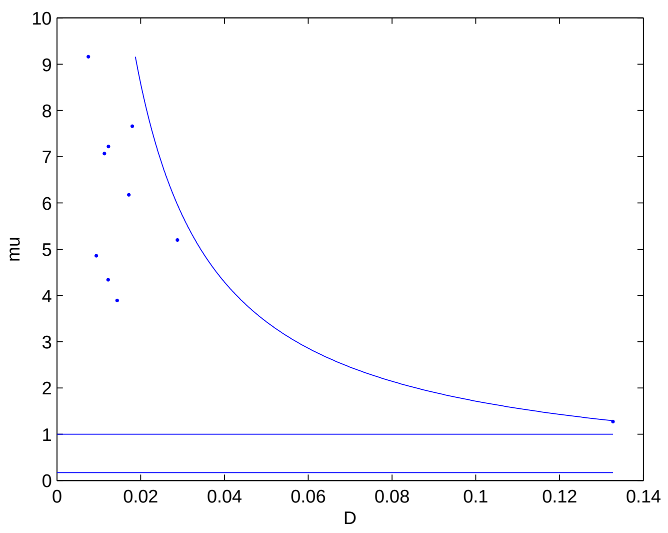


Figure 4.1: The points found by the program cmaes.m for the fitness function LS2 and the boundaries.

We begin with the fitness function LS2. The found points are shown in figure 4.1. The points are close to the boundary between the H and the I region and the error is pretty small. Unfortunately the other boundaries were not found. We can explain this by looking at figure 2.8, there the blue region is larger between the H and the I region than between other regions.

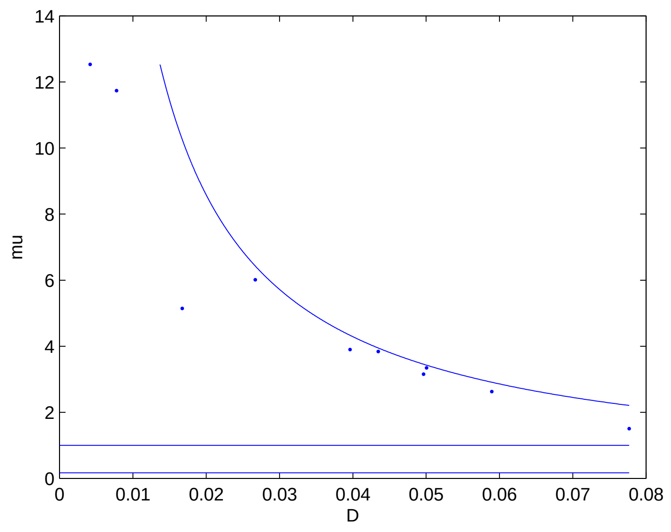


Figure 4.2: The points found by the program cmaes.m for the fitness function NN2 and the boundaries.

Now we use the fitness function NN2. The resulting points are drawn in figure 4.2. The error is even smaller than before. The same boundaries were found as with the LS2. By looking at figure 2.13, we see that the fitness value is smaller on the found boundary than on the others. This could be a possible explanation.

By using the NN3, the points found are also close to the boundaries and all boundaries were found, this we can see in figure 4.3. This could be a coincidence or an advantage of the function NN3. In figure 2.14 we see, that the fitness values close to all boundaries are more or less the same. This could be a reason for the discovery of different boundaries. The error is larger than before, but this is due to the point the most right. If we ignore this point, the error is between the one of LS2 and NN2.

Using the function CART1, the error is so far the biggest one. Nevertheless figure 4.4 shows a good result except for three points. But if we get such a result in the next chapter, we would be very happy. The bottom boundary is never found, but the boundary between regions H and G_b is found four times.

The last function used, CART2, has then compared to the others a big error. For the green points in figure 4.5, the fitness limit $2 \cdot 10^{-2}$ was not reached. The program cmaes.m stopped with a message warnequalfunvals. This means, that the CMA searched in an area, where the fitness values are the same. If the fitness values are the same, the algorithm can not find a search direction and stops with the message warnequalfunvals. CART functions are piecewise constant functions, which is an explanation for the

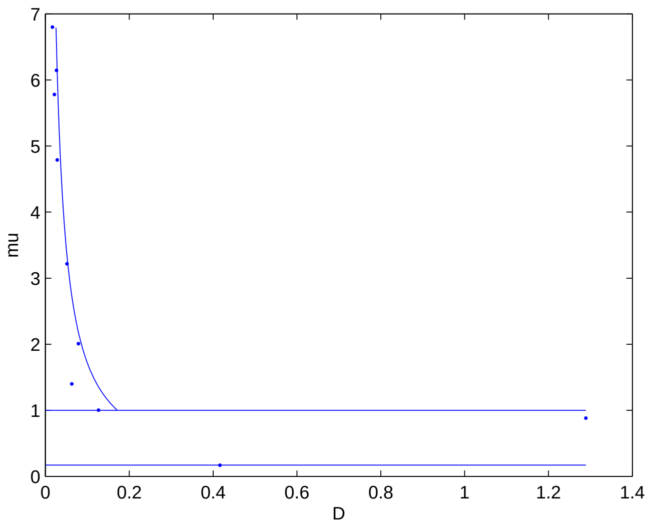


Figure 4.3: The points found by the program cmaes.m for the fitness function NN3 and the boundaries.

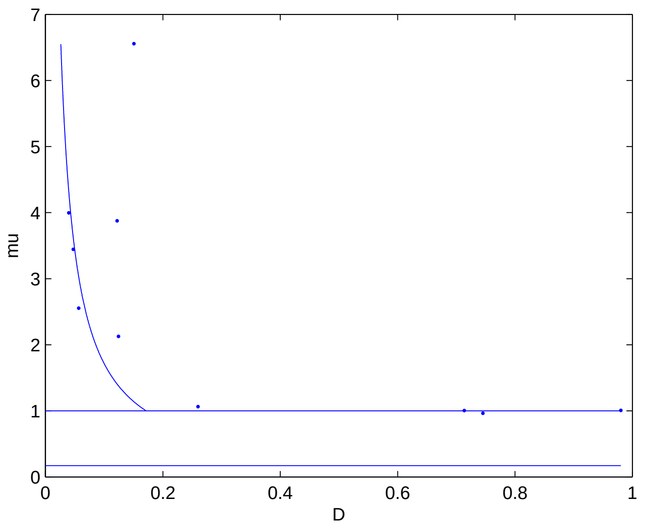


Figure 4.4: The points found by the program cmaes.m for the fitness function CART1 and the boundaries.

appearance of the message warnequalfunvals. We see in figure 4.5 two points far away from the boundaries. If we leave the two bad points out, the error is 0.0257, which is comparable to the other errors.

In general the 5 tested functions do their job. They can locate the

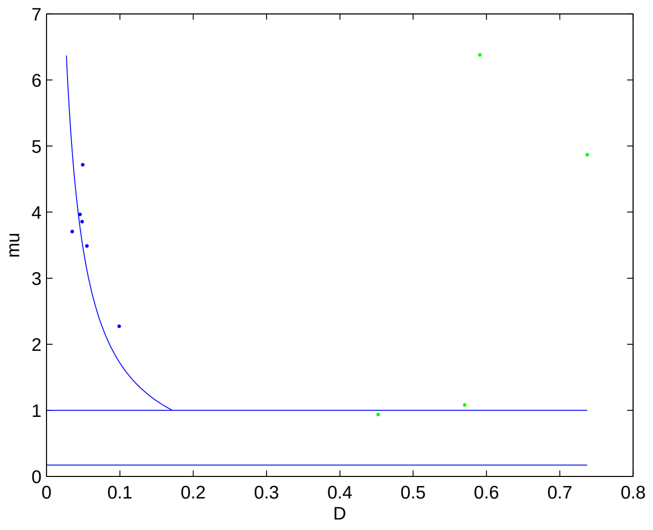


Figure 4.5: The points found by the program cmaes.m for the fitness function CART2 and the boundaries.

Function:	error
LS2	0.0140
NN2	0.0089
NN3	0.0217
CART1	0.0376
CART2	0.1472

Table 4.1: The error calculated for 5 fitness functions

boundary between the regions H and I better than the others. The boundary between the regions G_a and G_b is just once found by the function NN3. This could be a coincidence or the function NN3 can better locate this boundary. In the next subsection we will only use the function NN3, because it was throughout good. Surely this is a little bit arbitrary, but we have to decide us for one, because we do not want to force complexity and keep comparability.

We also want to evaluate the performance of the classification algorithm introduced in section 3.2. We start with an initial population size of 100 and a tolerance of 10^{-3}. The 100 initial starting points are always vectors $\in \mathbb{R}^2$ uniformly distributed over the range [0, 6] for the first component μ and uniformly distributed over the range [0, 0.8] for the second component D. The number of resulting points is 22 and these points are drawn together with the boundaries in figure 4.6. The resulting error is 0.0189.

This algorithm can find the boundaries well, except for the bottom

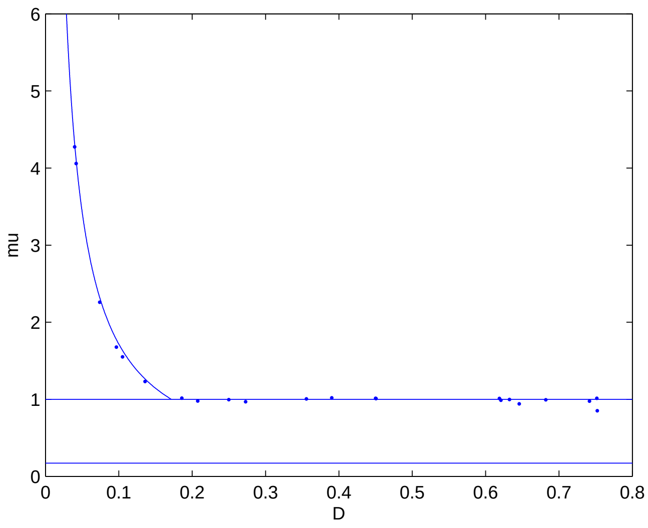


Figure 4.6: The points found by the classification algorithm

boundary. The error is comparable to the error of the fitness functions used with the CMA.

4.2 An A Little Bit Different System

Now we run the CMA for different simulation parameters, i.e. the parameters *endtime*, N, *zufall* and δx, and σ-values, to test the robustness of our approach. The used reaction diffusion system is the same as in section 4.1, except for the different σ-value. We have to consider, that the boundaries change with σ. We test the robustness only on a few examples and not rigorously and we will only use the fitness function NN3. First we let all parameters as written in chapter 2, except the parameter *endtime* is once set to 100 and once to 10000. The CMA finds the points drawn in figures 4.7 and 4.8 with an error of 0.0162 for *endtime* $= 100$ and 0.0451 for *endtime* $= 10000$.

We see the figures are quite similar to figure 4.3, except for the 3 points beyond the boundaries in figure 4.8. The errors are as well close to the error of NN3 in the last section. So we can say, that the approach is not very sensitive to changing the parameter *endtime*. Clearly if we choose *endtime* too small, there will be problems.

Now we let all parameters as written in chapter 2, except the parameter N is once set to 50 and once to 200. The CMA finds the points drawn in figures 4.9 and 4.10 with an error of 0.0132 for $N = 50$ and 0.0110 for $N = 200$.

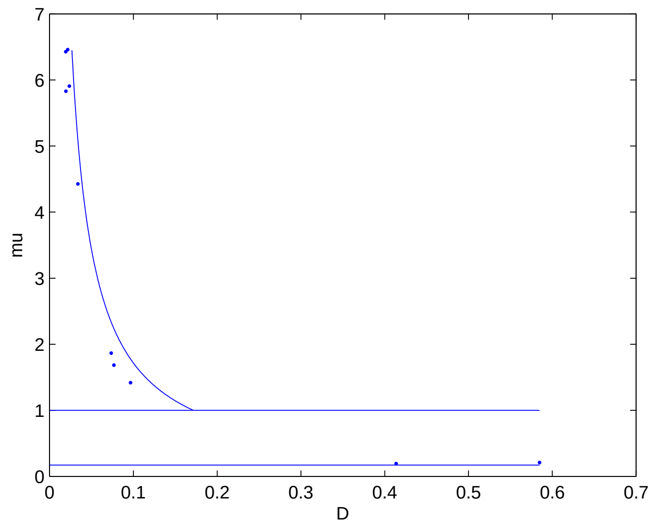


Figure 4.7: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with *endtime* = 100

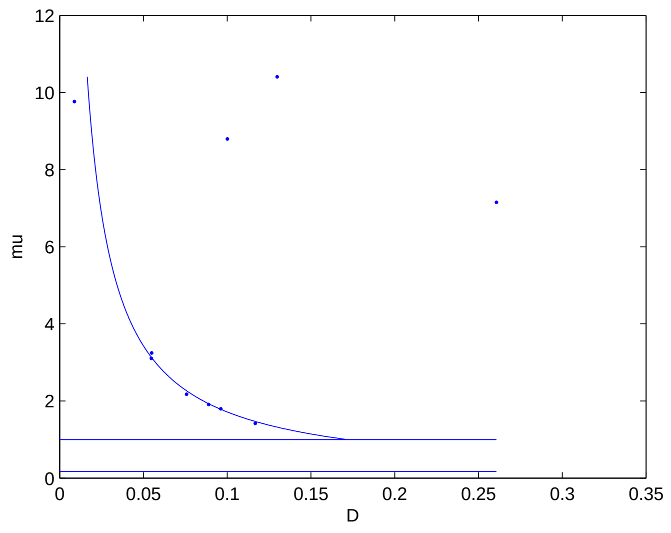


Figure 4.8: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with *endtime* = 10000

The errors and figures for changing N are as well quite similar to the ones in the last section. Hence changing N is not critical.

Further we let all parameters as written in chapter 2, except the parameter *zufall* is once set to 0.001 and once to 10. The CMA finds the points

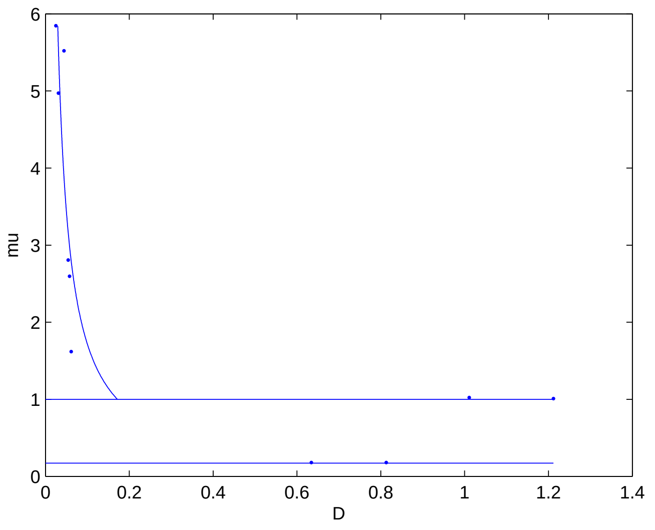


Figure 4.9: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with $N = 50$

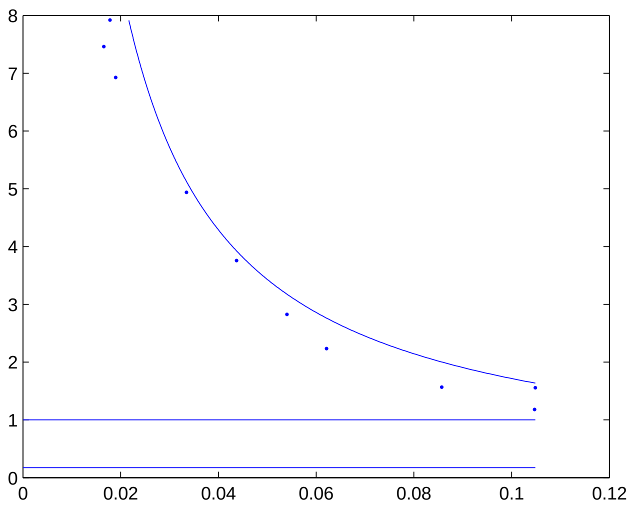


Figure 4.10: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with $N = 200$

drawn in figures 4.11 and 4.12 with an error of 1.1583 for $zufall = 0.001$ and 0.6186 for $zufall = 10$.

Changing the parameter *zufall* can result in a very bad result. The boundaries were not reliable found and the error is big. So we conclude, that

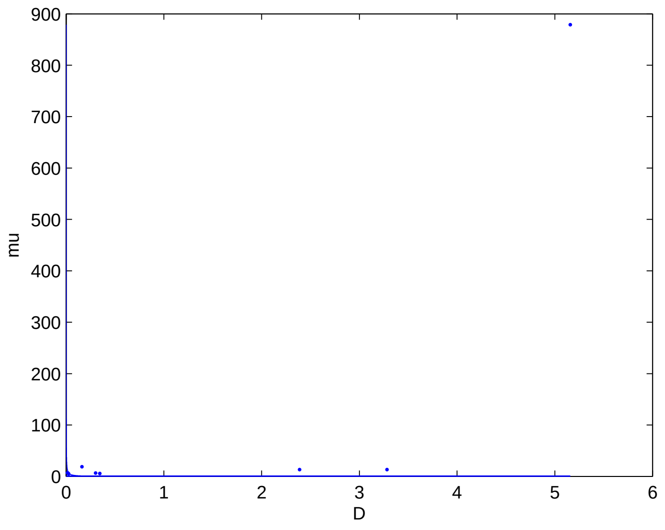


Figure 4.11: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with *zufall* = 0.001

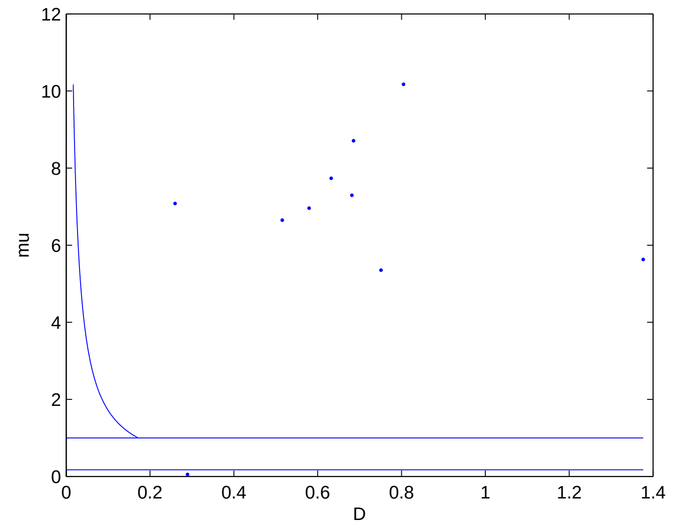


Figure 4.12: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with *zufall* = 10

changing *zufall* too much is critical. If *zufall* is too small, it could happen, that also in other regions than H the homogeneous steady-state solution is reached, because of the too little perturbation. Therefore the solution near the boundaries is typical for the H region and hence the boundaries were not found. If *zufall* is big, it could be, that in region H the concentrations do not go back into the homogeneous steady-state solution. So minima were found in the H region.

Now we let all parameters as written in chapter 2, except the parameter δx is once set to 0.25 and once to 1. The CMA finds the points drawn in figures 4.13 and 4.14 with an error of 0.0198 for $\delta x = 0.25$ and 0.0239 for $\delta x = 1$.

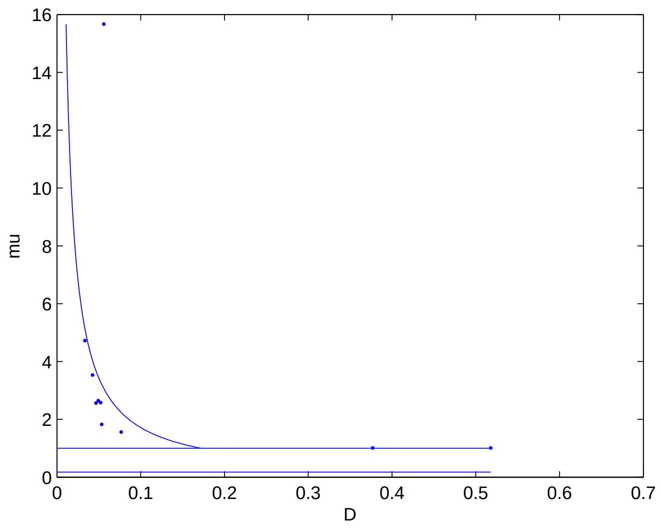


Figure 4.13: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with $\delta x = 0.25$

The results of the CMA with a different δx than in the last section, are comparable to the result in the last section. So again changing δx is not critical.

Last we change the value of σ once to 0.2 and once to 0.7. The simulation parameters are equal to the values assigned in chapter 2. The CMA finds the points drawn in figures 4.15 and 4.16 with an error of 0.0214 for $\sigma = 0.2$ and 0.3730 for $\sigma = 0.7$. Note the changed boundaries in figures 4.15 and 4.16.

The change in σ is probably the most interesting test in this chapter. The results are getting worse the bigger σ. If σ changes, the boundaries change and the homogeneous steady-state concentrations $a_0 = 1 + \sigma$, $h_0 = a_0^2$ change. The change of a_0 and h_0 probably irritates the NN. A possible approach to deal with this problem, is to scale some elements of the *prop* vector. We divide the elements 3 to 7, 13 to 17 and 23 to 27 through a_0 and the elements 8 to 12, 18 to 22 and 28 to 32 through h_0 before passing on to the fitness function. The points found are shown in figure 4.17 for

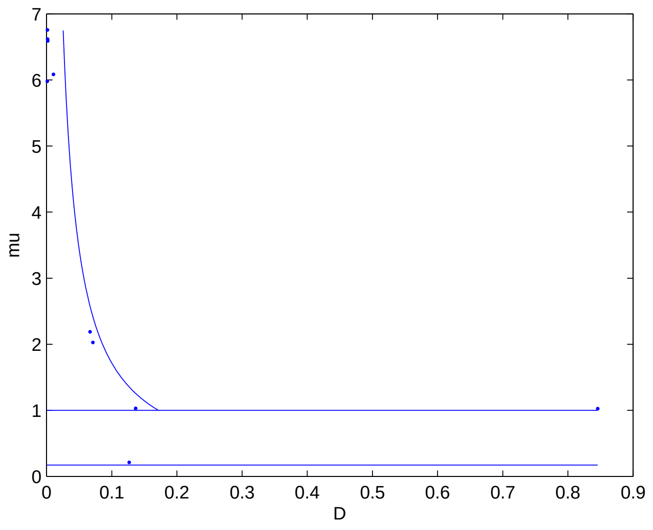


Figure 4.14: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with $\delta x = 1$

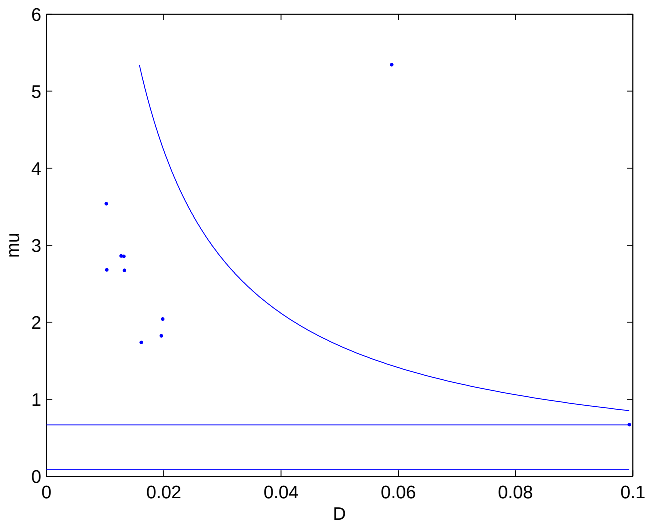


Figure 4.15: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with $\sigma = 0.2$

$\sigma = 0.2$ and in figure 4.18 for $\sigma = 0.7$. The errors are 0.0517 for $\sigma = 0.2$ and 0.0230 for $\sigma = 0.7$. For $\sigma = 0.7$ the results are better with the scaling and for $\sigma = 0.2$ they are a bit worse. Mostly one can calculate the homogeneous steady-state concentrations and the scaling can be done.

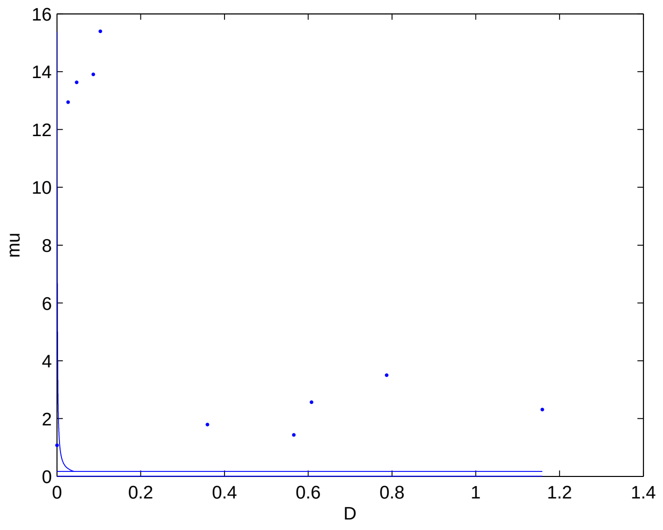


Figure 4.16: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with $\sigma = 0.7$

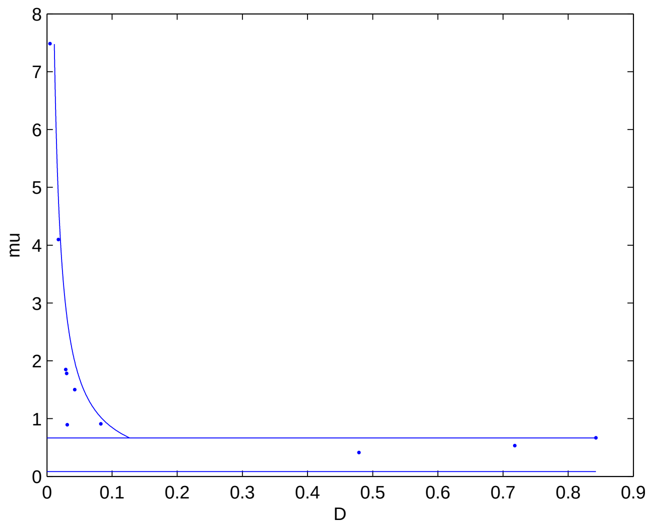


Figure 4.17: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with $\sigma = 0.2$ and scaled properties

Additionally we want to see how the classification algorithm behaves for different σ-values. We use the same two values 0.2 and 0.7 as above. The points found by the algorithm are shown in figures 4.19 and 4.20. The error is 0.0367 for $\sigma = 0.2$ and 0.0133 for $\sigma = 0.7$.

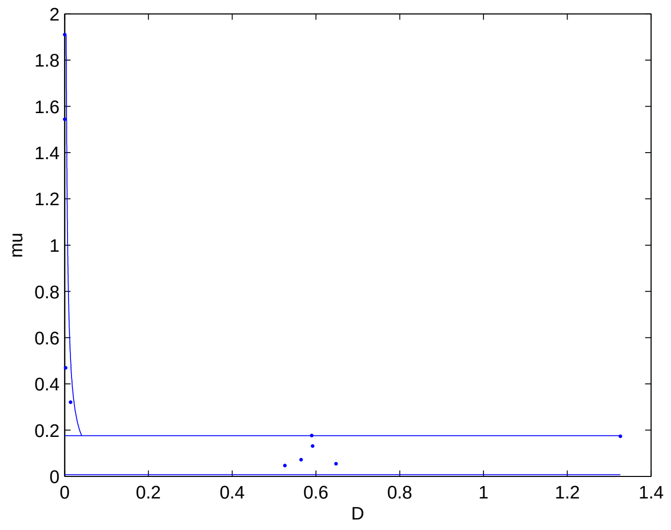


Figure 4.18: The points found by the program cmaes.m for the fitness function NN3 and the boundaries, with $\sigma = 0.7$ and scaled properties

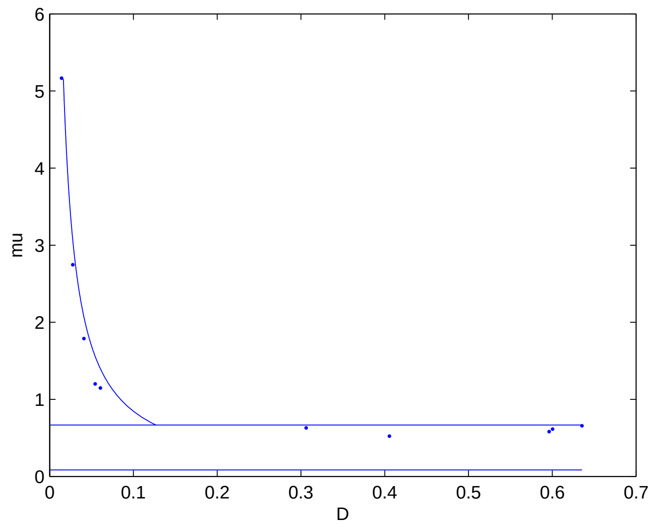


Figure 4.19: The points found by the classification algorithm, with $\sigma = 0.2$

The results are good, but the classification algorithm has a disadvantage. Often the H region is big and the other regions are small. If we choose initial points randomly, which is reasonable because we do not know where to search, it becomes less likely to find points in another region than H. The classification algorithm needs points in another region than H, else he can

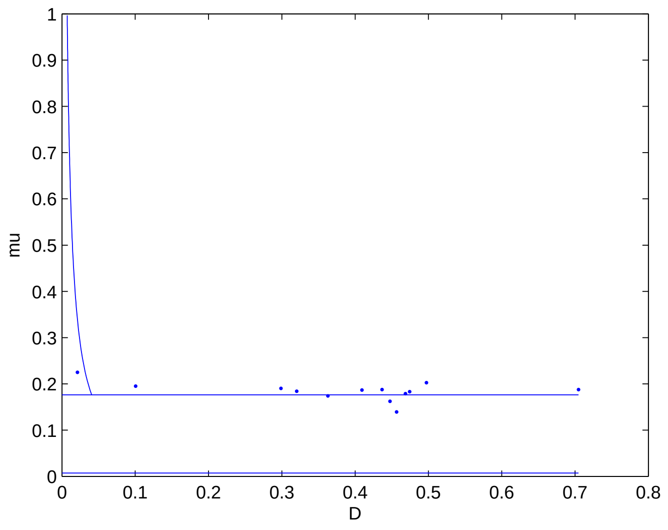


Figure 4.20: The points found by the classification algorithm, with $\sigma = 0.7$

not find any boundary. The CMA can also work with exclusively points in the H region, because not every point in the H region has the same fitness value and the CMA can follow the gradient. Due to this big disadvantage and the good results of the CMA, we will not consider the classification algorithm anymore.

Chapter 5

CMA Applied To A More Difficult System

In this chapter we apply the CMA to reaction diffusion systems different to the one given by equations (2.3) - (2.4). In the first section the CMA is applied to a reaction diffusion system with 3 parameters related to the one of equations (2.3) - (2.4). The reaction diffusion system discussed in the second section has 2 parameters and reminds of the system given by equations (2.3) - (2.4), but relies on a different dynamic. We do not know the boundaries of these two reaction diffusion systems.

5.1 CMA Applied To A System With 3 Parameters

The equations of the reaction diffusion system investigated in this section is:

$$\frac{\partial a}{\partial t} = D \cdot \Delta a + \frac{a^2}{(1+\kappa \cdot a^2) \cdot h} - a \tag{5.1}$$

$$\frac{\partial h}{\partial t} = \Delta h + \mu \cdot (a^2 - h) \tag{5.2}$$

So we have an additional parameter κ. This system is described in [3], but with more parameters, for example D_a, D_h, κ_a and so on. By finding the right dimensions for t, x, a and h and setting σ_h to zero, we can get a system with only 3 parameters D, μ and κ. The procedure is explained in [3] for $\kappa_a = 0$. For $\kappa_a \neq 0$ we get the additional parameter $\kappa := \frac{\kappa_a \cdot \mu_h^2 \cdot \rho_a^2}{\mu_a^2 \cdot \rho_h^2}$. The parameters have to be positive.

We do not know the boundaries for this system, so we can not evaluate the performance. We need very many points to reliably locate the boundaries, due to the fact that a boundary in a three dimensional space is a plane and we need more points to outline it compared to a line. This is clearly a disadvantage and for a big number of parameters it could be, that we can not simulate enough points to reliably locate the boundaries. First we will calculate the homogeneous steady-state solution of the reaction diffusion system of equations (5.1) - (5.2). It is given by the two equations:

$$\begin{aligned} 0 &= \frac{a_0^2}{(1+\kappa \cdot a_0^2)\cdot h_0} - a_0 && (5.3)\\ 0 &= \mu \cdot (a_0^2 - h_0) && (5.4)\end{aligned}$$

Equation (5.4) for $\mu \neq 0$ is equivalent to:

$$h_0 = a_0^2 \tag{5.5}$$

If we plug equation (5.5) into equation (5.3), we get a polynomial in a_0:

$$\kappa \cdot a_0^3 + a_0 - 1 = 0 \tag{5.6}$$

The polynomial $y(a_0) = \kappa \cdot a_0^3 + a_0 - 1$ is -1 for $a_0 = 0$ and κ for $a_0 = 1$. So the polynomial $y(a_0)$ is smaller than zero for $a_0 = 0$ and bigger than zero for $a_0 = 1$, if $\kappa > 0$. Therefore because $y(a_0)$ is a continuous function, $y(a_0)$ has a zero between 0 and 1. This zero can be calculated with the MATLAB function roots. The derivative of $y(a_0)$ is $y'(a_0) = 3 \cdot \kappa \cdot a_0^2 + 1$ and is greater than zero for $a_0 \in \mathbb{R}$ and $\kappa > 0$. So the function $y(a_0)$ is strictly monotone growing. Therefore the zero between 0 and 1 is the only real valued zero. Now we are ready to run the CMA. We let κ constant and run the CMA to find the boundaries with this constant κ. Then we repeat this procedure for different κ and therefore get boundaries for different discrete κ values. We can do this only for a finite number of κ values, which is not sufficient to determine all boundaries. But it facilitates the problem a lot and to decide whether our approach can find the boundaries, this procedure suffices. The chosen discrete κ values are: 0.1, 0.3, 0.5 and 1. The fitness function is again the NN3 function. To avoid points with $\mu < 0$ or $D < 0$ or $\kappa < 0$, we set the fitness value to 10 for such points, as we did in the last chapter. The initial starting point is always a vector $\in \mathbb{R}^2$ uniformly distributed over the range [0, 6] for the first component μ, uniformly distributed over the range [0, 0.8] for the second component D. The initial coordinate wise search standard deviation is 2 for μ, 0.3 for D. StopFitness was chosen to be 10^{-8} and the option MaxFunEvals was chosen to be 1000. Because the steady-state solution is not the same as for the system (2.3) - (2.4), we scale the properties. We simulated for every κ value 60 points. The points, which did not reach the fitness value 10^{-8}, were sorted out. Also a little number of points, which are very far apart of all other points were sorted out. The remaining points are drawn together in figure 5.1.

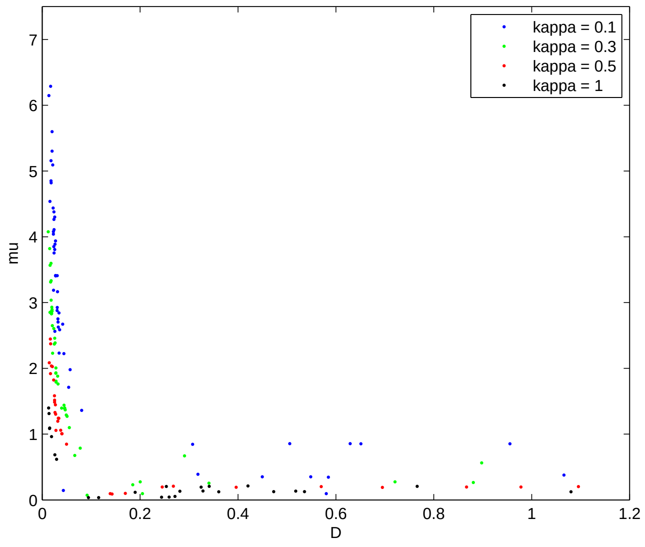


Figure 5.1: The remaining points found by the CMA for the reaction diffusion system (5.1) - (5.2) are shown.

According to this plot, we can extract the boundaries. We do this by eye. There are not very much points in the region, where $\mu < 1$. So it is difficult to extract the boundaries there. Nevertheless we try it. The extracted boundaries are shown together with the points in figure 5.2.

The shape of the extracted boundaries is similar to the shape of the boundaries of figure 2.2. Of course if we never saw the figure 2.2, probably we would not draw the boundaries as in figure 5.2. In figure 5.2 we would expect one additionally red and one additionally black straight line, such that there are for every κ two straight lines. Probably the two missing lines would appear, if we would simulate more points. In the previous chapter we observed, that the boundary between the regions I and H are found better. So we expect the extracted boundaries for $\mu > 1$ are the boundaries between the I and the H region. Now we have to check, whether the estimated boundaries are really boundaries. This is the case, if the behaviour of the reaction diffusion system is different on the one side of a boundary than on the other. We do this in two steps. First we check the estimated boundaries for $\kappa = 0.3$. Then we choose one point in the (μ, D) plane, which is for one κ on the one side of the boundary and for another κ on the other side of the boundary. Then we simulate the solution of this point in the (μ, D) plane for every discrete κ value and we hope, that the behaviour of the solution changes with κ. We begin with checking the boundaries for $\kappa = 0.3$. We simulate the solution of the reaction diffusion system for the points shown

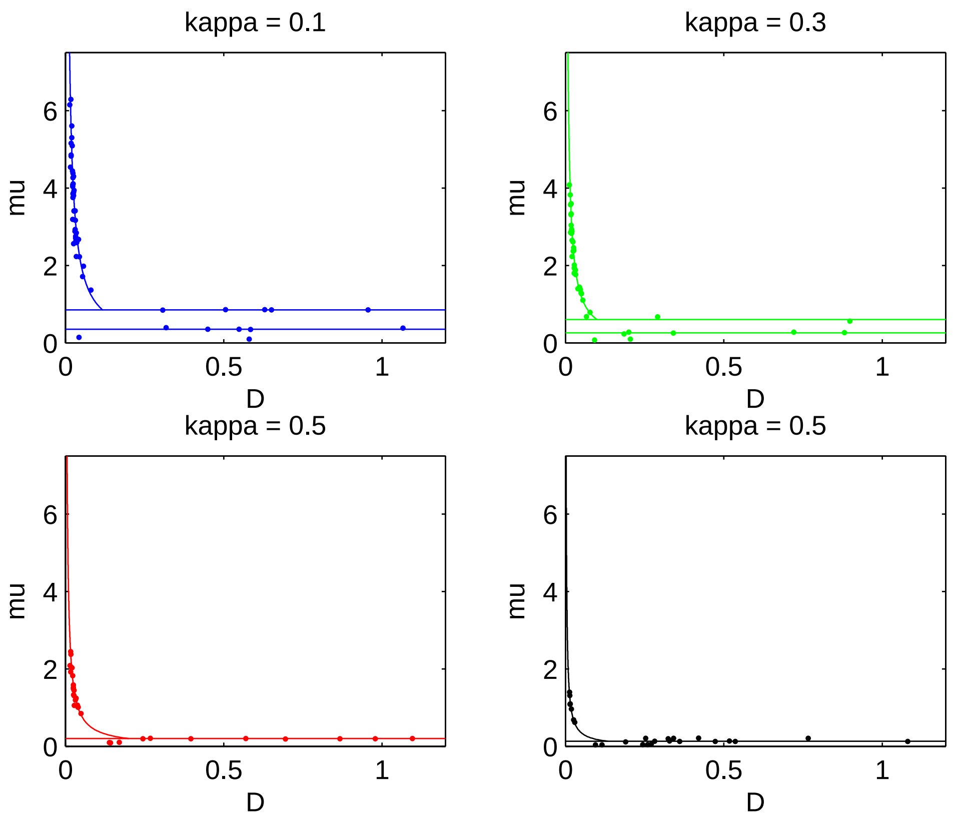


Figure 5.2: The points drawn together with the extracted boundaries.

in figure 5.3 and determine the behaviour of the solutions by eye.

The solution of the 5 points are shown in figure 5.4.

We call the top plot in figure 5.3 solution 1 and the lowest solution 5. Ideally would be, that solution 1 behaves different than all other solutions. Solution 2 and 3 behave equally but different than all others. Solution 4 behaves different than all other solutions and solution 5 behaves different than all other solutions. Solution 1 is clearly a timely stable and spatially inhomogeneous pattern. Solutions 2 and 3 are homogeneous steady-state solutions. Solutions 4 and 5 are neither stable inhomogeneous patterns nor homogeneous steady-state solutions. To show, that solution 4 and 5 behave different, we recall the regions G_a and G_b from chapter 1. In region G_a perturbations grow exponentially and in region G_b perturbations oscillate and grow exponentially. If we draw the time course of the solutions 4 and 5 for one space point 5.12, we can see oscillations only in the course of solution 4.

Hence we verified the boundaries for $\kappa = 0.3$. We also have to check, whether the boundaries are correct for $\kappa = 0.1, 0.5, 1$. We do not do this

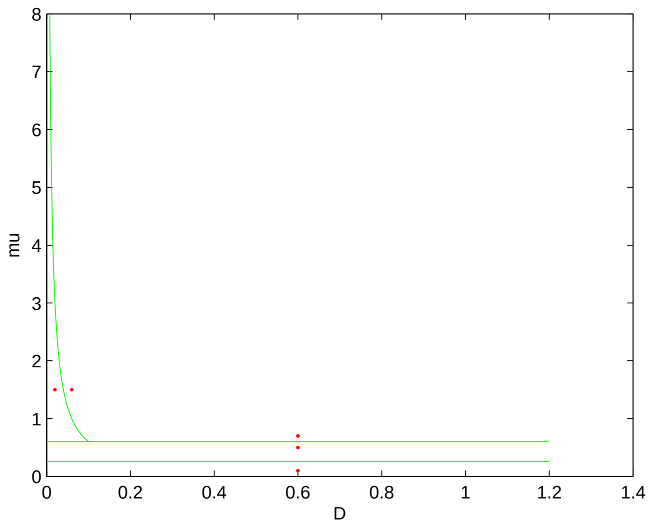


Figure 5.3: The estimated boundaries for $\kappa = 0.3$ drawn together with 5 points near the estimated boundaries.

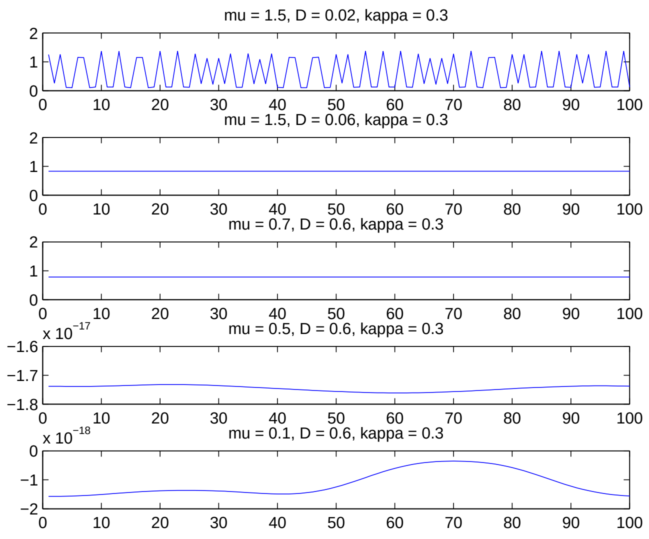


Figure 5.4: The concentration of a is shown in dependence of the 1-dimensional spatial coordinate for the 5 points in figure 5.3.

rigorously. We fix one point in the $(\mu,\ D)$ plane, the point $(1.5,\ 0.0325)$. This point is for $\kappa = 0.1, 0.3$, according to the boundaries in figure 5.2, in the region with pattern formation and for $\kappa = 0.5, 1$ in the region, where

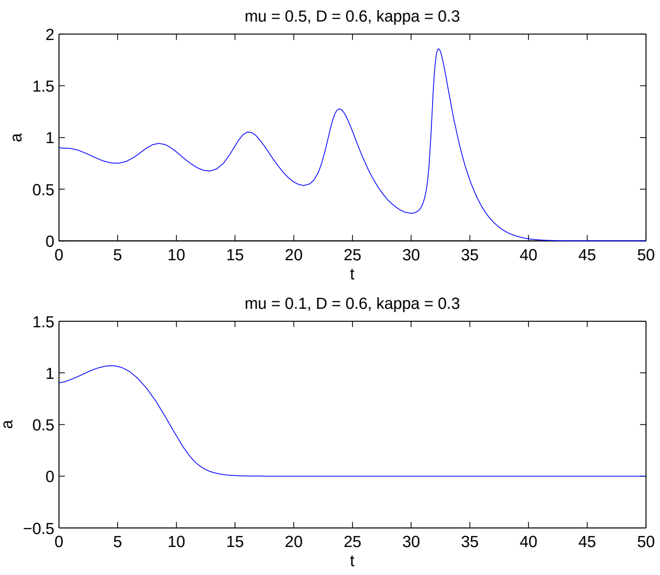


Figure 5.5: The concentration of a for one space point ($a(4 \cdot \delta x, t)$) is shown in dependence of time. There are oscillations in the upper plot and no oscillations in the bottom plot.

the solution is the homogeneous steady-state solution. If the boundaries are correct, we should observe such a behaviour. The solutions of the reaction diffusion system (5.1) - (5.2) for the 4 points is shown in figure 5.6.

The first two solutions in figure 5.6 are timely stable and spatially inhomogeneous pattern and the last two are homogeneous steady-state solutions. Therefore the found boundaries are correct.

5.2 CMA Applied To A System With 2 Parameters

The last reaction diffusion system considered in this project, is the following:

$$\frac{\partial a}{\partial t} = D \cdot \Delta a + s \cdot a^2 - a \tag{5.7}$$

$$\frac{\partial s}{\partial t} = \Delta s + \mu \cdot (1 - s \cdot a^2) \tag{5.8}$$

The system discussed in the last section is for $\kappa = 0$ the same system as the one given by equations (2.3) - (2.4), so they are quite similar. The reaction diffusion system of this section is not similar to the system given by equations (2.3) - (2.4), because it relies on a different dynamic. So the results in this section are the most interesting ones of the whole project.

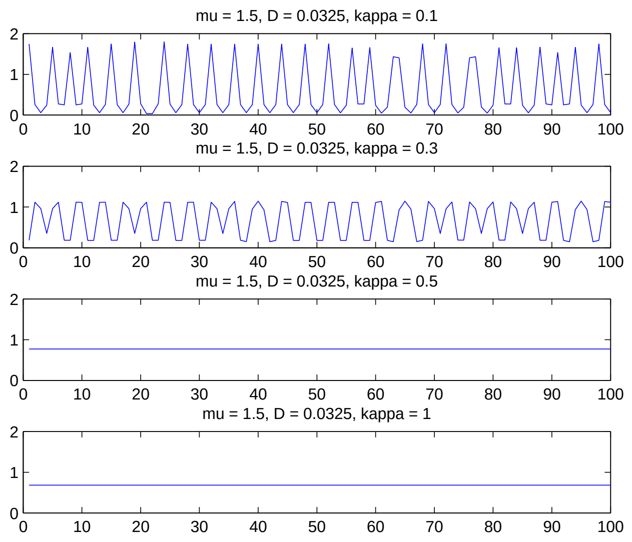


Figure 5.6: The concentration of a is shown in dependence of the 1-dimensional spatial coordinate for the 4 points $(\mu, D, \kappa) \in \{(1.5, 0.0325, 0.1), (1.5, 0.0325, 0.3), (1.5, 0.0325, 0.5), (1.5, 0.0325, 1)\}$.

We calculate the homogeneous steady-state solution, which is given by the following two equations:

$$0 = s_0 \cdot a_0^2 - a_0 \tag{5.9}$$
$$0 = 1 - s_0 \cdot a_0^2 \tag{5.10}$$

From equation (5.10) follows:

$$s_0 = \frac{1}{a_0^2} \tag{5.11}$$

If we plug s_0 into equation (5.9), we get $a_0 = 1$ and therefore $s_0 = 1$.

The second concentration s behaves different than h. So very likely the fitness functions done so far are not suitable for such a problem, because they use properties extracted from the concentration h. Because we do not know the boundaries of this reaction diffusion system, we can not repeat the same procedure done earlier. But we have some of the 32 properties, which can be used here as well. All those, which are derived from a and properties 1 and 2. We can make a new fitness function, which takes into account only the mentioned properties. The only things we need, are the 2500 simulated points of chapter 2. We do the same procedure as for the function NN3, but instead of using all 32 properties with transformations, we only use the mentioned 17 properties with transformations. The threshold t is set to be 0.4. By doing so, 22 transformed properties remain. The resulting function drawn for the 2500 simulated points is shown in figure 5.7. We call this function NN3s.

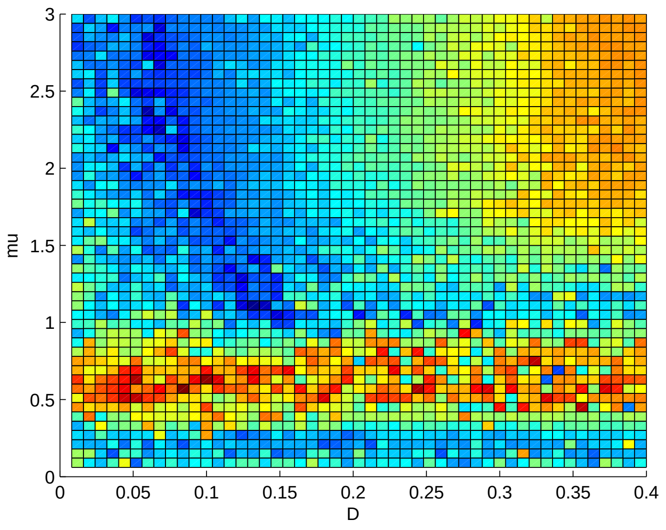


Figure 5.7: The NN with 22 of the 85 transformed properties drawn for the simulated (μ, D) points

Now we use the CMA with the fitness function of figure 5.7. We simulate 84 points and set the fitness value of all points with $D < 0$ or $\mu < 0$ to 10. The initial starting point is always a vector $\in \mathbb{R}^2$ uniformly distributed over the range [0, 6] for the first component μ and uniformly distributed over the range [0, 0.8] for the second component D. The initial coordinate wise search standard deviation is set to 2 for μ and 0.3 for D. The options StopFitness is set to 10^{-8} and MaxFunEvals to 1000. Because the homogeneous steady-state solution a_0 is the same as for the system (2.3) - (2.4), the problem of different homogeneous steady-state solutions do not occur. So we do not need to scale any properties. The points found are shown in figure 5.8.

By looking at figure 5.8, we can essentially extract 3 lines. Because we are satisfied with an approximated course of the boundaries, we determine them by eye. The extracted boundaries are shown in figure 5.9 together with the 84 points.

We have to check whether the extracted boundaries are really boundaries of the reaction diffusion system of equations (5.1) - (5.2). This is the case, if the behaviour of the system is different on the one side of a boundary than on the other. We do not verify this rigorously. We choose the 5 points of figure 5.10 and determine the behaviour of the 5 solutions by eye.

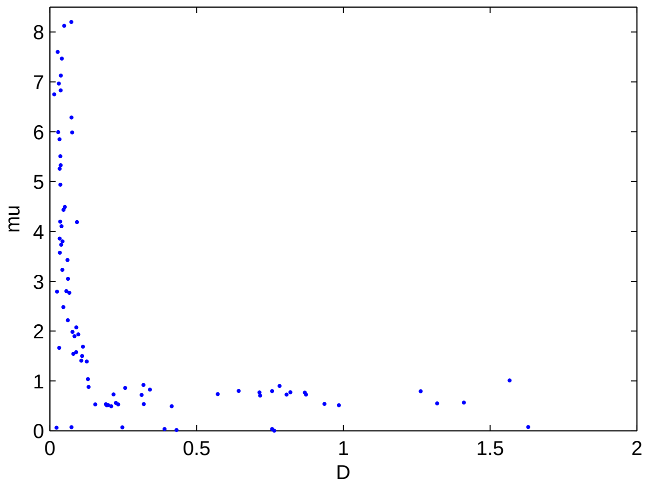


Figure 5.8: The points found by the program cmaes.m for the fitness function NN3s.

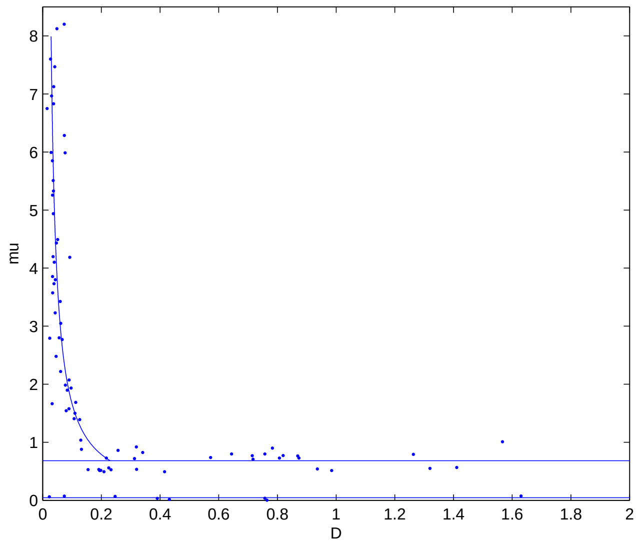


Figure 5.9: The extracted boundaries drawn together with the 84 found points.

The solutions are shown in figure 5.11. Solution 1 is clearly a timely stable and spatially inhomogeneous pattern. Solutions 2 and 3 are homogeneous steady-state solutions. Solutions 4 and 5 are neither stable inhomogeneous patterns nor homogeneous steady-state solutions. If we draw the

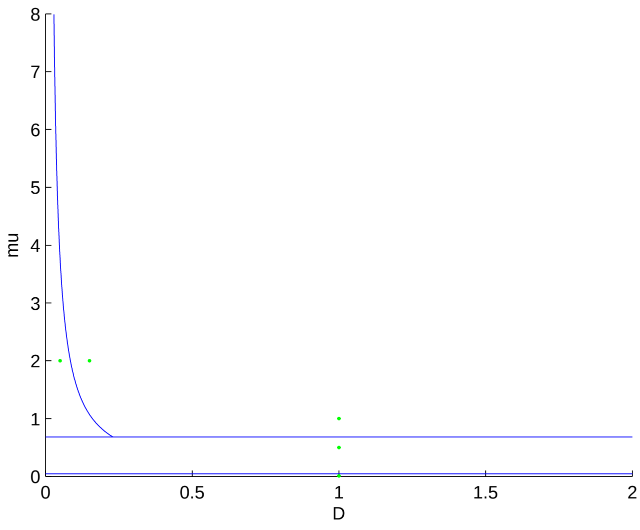


Figure 5.10: The chosen 5 points and the extracted boundaries.

time course of solution 4 and 5 for one space point 5.12, we see that they correspond to region G_b and G_a. So we can conclude, that the boundaries were found well.

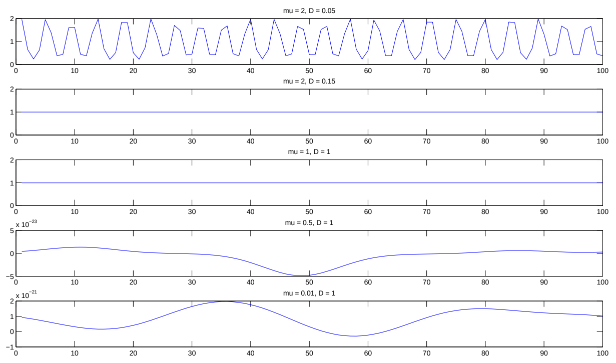


Figure 5.11: The concentration of a is shown in dependence of the 1-dimensional spatial coordinate for the 5 points.

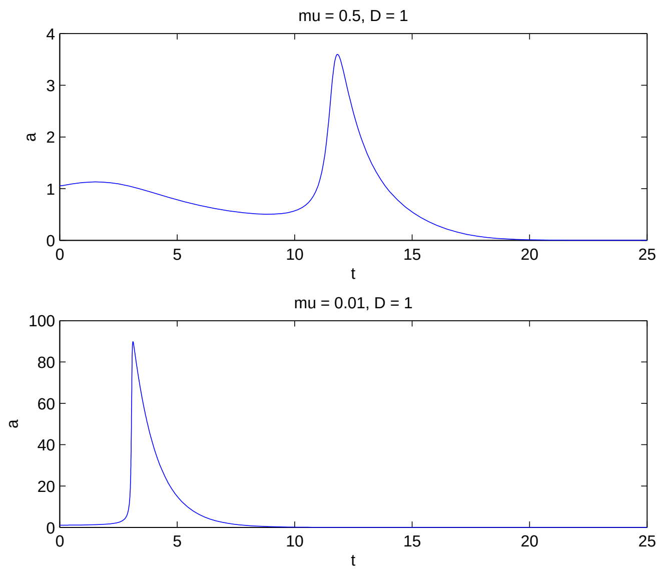


Figure 5.12: The concentration of a for one space point $(a(4 \cdot \delta x, t))$ is shown in dependence of time. There are oscillations in the upper plot and no oscillations in the bottom plot.

Chapter 6

Conclusions

We were searching for the parameters, for which a reaction diffusion system produces timely stable and spatially inhomogeneous patterns. This we have done by solving an equivalent problem, namely to find the boundaries between different solution regions of a reaction diffusion system. To find the boundaries, we used a special Evolutionary Algorithm, the CMA. A big difficulty of an EA is to find an appropriate fitness function, which is the main part of this project. The fitness function was created with the help of data from a reaction diffusion system with known boundaries. To this data statistical procedures were applied. The procedures used are Least Squares, Neural Networks and Classification and Regression Trees. It turned out, that Neural Networks are best suited. The CMA was then applied to two reaction diffusion systems with unknown boundaries. The boundaries were reliably found, but in general only if one scales the properties. If a reaction diffusion system has too much parameters, we would need too many points to locate the boundaries. For three parameters we can let the i-th ($i \in \{1, 2, 3\}$) parameter constant and search the boundaries in the plane spanned by the other two parameters. This simplifies the task, but we can only find the boundaries for a finite number of values of the i-th parameter. For l parameters we can let $l-2$ parameters constant and search the boundaries in the plane spanned by the other two parameters. Let us assume, that we choose for every of the $l-2$ constant parameters j values and we wish to find the boundaries for all combinations. So we wish to find the boundaries for j^{l-2} realizations of the constant parameters. Because we choose for every of the $l-2$ constant parameters the same number of values, the resolution is always the same. The quantity j^{l-2} grows exponentially with the number of parameters l, which is not desirable and limits the number l. In other words, the complexity of the problem of finding the boundaries grows exponentially with the number of parameters l. Another limiting factor of l is the fact, that we can only draw points in $\mathbb{R}^i$ for $i \in \{1, 2, 3\}$. Of course we can draw the plane spanned by the other two parameters for every realization of the constant parameters, but to keep the overview is very difficult and the number of such drawings grows exponentially with l. To find the boundaries for 5 parameters is already a difficult task and we think this is more or less the limit. Now we would like to give an outline of what one can do in future:

1. One can try to find the boundaries of a system without both chemicals a and h. So we can only use the properties 1 and 2. Do the properties 1 and 2 suffice to estimate the distance to the boundaries?

2. For some parameter values in the G_b region the simulation of the reaction diffusion system takes very long. Can we estimate for these parameter values the distance to the boundary based on shorter simulations?

Bibliography

[1] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Natural Computing Series. Springer, Berlin, Germany, 1. edition, 2003.

[2] A. Turing. The chemical basis for morphogenesis. *Philos Trans R Soc Lond, B*, 237:37–72, 1952.

[3] A. J. Koch and H. Meinhardt. Biological pattern formation: from basic mechanisms to complex structures. *Rev mod Phys*, 66(4):1481–1510, 1994.

[4] S. J. Ruuth. Implicit-explicit methods for reaction-diffusion problems in pattern formation. *J Math Biol*, 34(2):148–176, 1995.

[5] H.-A. Loeliger. *Signal and information processing: Modeling, filtering, learning. Lecture notes, Signal and Information Processing Laboratory, ETH Zürich.* FS 2007.

[6] P. Bühlmann. *Computational Statistics. Lecture notes, Seminar für Statistik, ETH Zürich.* Summer 2006.

[7] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[8] N. Hansen and S. Kern. Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In X. Yao et al., editor, *Conference on Parallel Problem Solving from Nature (PPSN VIII)*, volume 3242 of *LNCS*, pages 282–291, Berlin, Germany, 2004. Springer.