

Parameter Estimation in a Percolation model with coloring



MASTER'S THESIS

Albert-Ludwigs-University of Freiburg
Institute of Mathematics

Author:
Felix Beck

Supervisor: Dr. Prof. Peter Pfaffelhuber
Advisor: Dr. Bence Mélykúti

Place, date

Signature

Acknowledgments

I would like to express my gratitude to Dr. Bence Mélykúti, who assisted me and supported the implementation of parts of the MATLAB code. Furthermore, I would like to thank Maja Temerinac-Ott, who also supported the implementation of parts of the MATLAB code.

Statement of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without due acknowledgement in this thesis. This dissertation has not been submitted elsewhere in any other form for the fulfilment of any other degree or qualification.

Place, date

Signature

ABSTRACT. In this thesis we create a statistical method to estimate the contamination rate and the true concentration of DNA in a digital polymerase chain reaction. We give a brief introduction to percolation theory and explain the algorithms of the method of moments, the generalized method of moments and the method of simulated moments (MSM), which are statistical methods for parameter estimation. Besides, we define a mathematical model to apply the MSM for our specific setup. We use MATLAB for the implementation of the model and to receive our MSM estimators. We test our program on synthetic and real laboratory experiments and evaluate our MSM estimators.

ABSTRACT. In dieser Arbeit erstellen wir eine statistische Methode zur Schätzung der Kontaminationsrate und der wahren DNA-Konzentration in einer digitalen Polymerase-Kettenreaktion. Wir geben eine kurze Einführung in Perkolationstheorie und erklären die Algorithmen für die method of moments, die generalized method of moments und die method of simulated moments (MSM), welche statistische Methoden zur Parameterschätzung sind. Des Weiteren definieren wir ein mathematisches Modell, um die MSM an unsere Bedürfnisse anzupassen. Wir benutzen MATLAB für die Implementierung des Modells und um unsere MSM Schätzer zu erhalten. Wir testen unser Programm an synthetischen und echten Laborexperimenten und bewerten unsere MSM Schätzer.

Contents

	Page
Chapter 1. Introduction	1
1. Motivation and biochemical background	1
2. Experimental setup and contamination problem	2
Chapter 2. Percolation Theory	5
1. Introduction to Percolation	5
2. Bond Percolation	8
Chapter 3. The Method of Simulated Moments	11
1. Method of Moments	11
2. Generalized Method of Moments	12
3. Method of Simulated Moments	14
Chapter 4. Applying the MSM	20
Chapter 5. MATLAB algorithms	28
1. Algorithm for the recognition of the grid and colors	28
2. Implementation of the MSM	38
Chapter 6. Results	49
Chapter 7. Summary and outlook	59
Appendix	60
Appendix A. MATLAB Code I: Recognition of the grid and spots	61
Appendix B. MATLAB Code II: Estimation by MSM	81
Bibliography	109

CHAPTER 1

Introduction

1. Motivation and biochemical background

The aim of this thesis is to create a statistical, analytical method for the estimation of the contamination rate and the true concentration of DNA in a digital polymerase chain reaction (digital PCR).

A PCR is a common technique used in genetic laboratories to generate millions of copies of a DNA sequence. For this to happen, an enzyme called DNA-Polymerase is needed. The PCR is a chain reaction in the sense that the result of one cycle is directly used for the next cycle, leading to exponential growth. There are many areas of application of PCR, including identification of genetic disorders and viral diseases, DNA fingerprinting, parental testing and even forensics.

Currently the most accurate method to quantify individual DNA sequences is the digital PCR: diluted DNA is distributed over thousands of separated cavities so that most cavities receive either zero or one DNA molecule (*digital*). During the PCR, the DNA molecules get replicated (*amplified*) in each cavity. By counting the cavities with amplification, one could theoretically measure the initial concentration of DNA. However, there is a problem with this determination of DNA concentration: During the PCR, it can happen that some cavities are not insulated perfectly. In this case, their content could contaminate neighboring cavities. This means that even if some cavities did not receive a DNA molecule initially, they might have a content after the PCR caused by *contamination*.

Based on the *method of simulated moments* (MSM, Section 3.3), we will calculate estimators for the contamination rate in a digital PCR and the true concentration of DNA.

This thesis is organized as follows:

In Section 1.2, we describe the experimental setup for the digital PCR and explain the problem of contamination that arises during the laboratory experiments. Chapter 2 gives a short introduction to percolation theory. In Chapter 3 we describe the method of moments and the generalized method

of moments. Based on those two estimation methods, we introduce the method of simulated moments, which we use to estimate the DNA concentrations and the contamination rate. Furthermore, some asymptotic properties of the MSM are stated and proven. A mathematical model to apply the method of simulated moments to our experimental setup and contamination problem is defined in Chapter 4. Chapter 5 is a brief description of how we implement our model in MATLAB to receive our MSM estimators. It also gives instructions on how to work with our MATLAB program as a user. In Chapter 6, we test our program on synthetic and laboratory experiments and evaluate our estimators. A summary and some suggestions for possible improvement are stated in Chapter 7.

2. Experimental setup and contamination problem

In this section we briefly describe the laboratory experiments conducted by the group of Dr. Günter Roth (Center for Biological Systems Analysis, University of Freiburg) that form the basis for our estimations and we state the contamination problem.

[Hof+12a] provides a setup and protocol for amplification of DNA molecules with a starting concentration of ≤ 1 molecule (digital PCR). Via a technique explained in [Hof+12b] it is possible to graft PCR primers onto various lab-on-a-chip substrates like glass or PDMS. Combining those two protocols, we will focus on the following experimental setup:

A picowell array consists of multiple thousands of wells arranged according to hexagonal tiling. Three different DNA samples (DNA1, DNA2, DNA3) are given into a picowell array, diluted so that most wells will receive either zero or one DNA molecule, i.e. we have a binary state (typically called *digital* state). In the next step, the DNA samples are amplified with a PCR and grafted onto a (e.g. glass) slide. By using three different fluorescent probes and scanning the slides with each probe, we can create an image of the wells where each well has a distinct color (Figure 1):

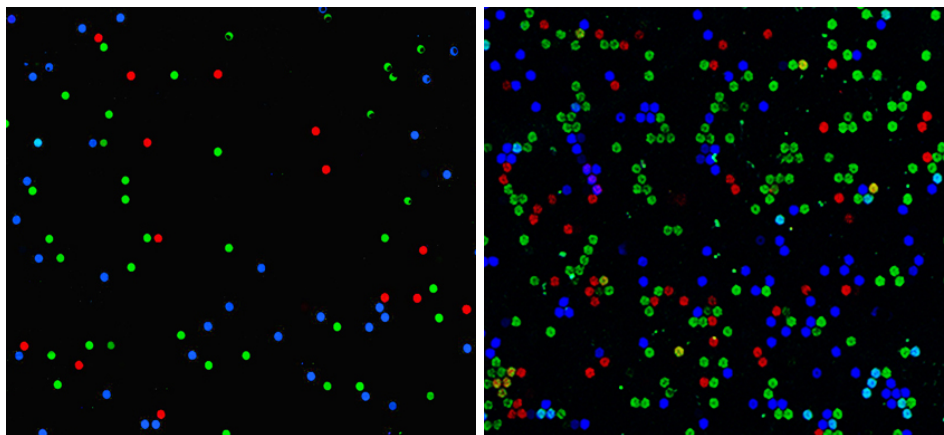
black	well is empty
red	well contains DNA1
green	well contains DNA2
blue	well contains DNA3
yellow	well contains DNA1 and DNA2
magenta	well contains DNA1 and DNA3
cyan	well contains DNA2 and DNA3
white	well contains DNA1, DNA2 and DNA3

As stated in Section 1, one main goal is to find the true DNA concentrations. However, a problem appears if we want to determine these concentrations by counting the colored wells: Some wells are not insulated perfectly and during the experiment, their content can contaminate neighboring wells. In the pictures we get from the experiments, it is not known if e.g. two neighboring wells with the same color were seeded with the same DNA samples or if one of them was initially empty and later contaminated by the other well (Figure 1).

According to Dr. Roth, no literature exists that deals with the estimation of contamination rates.

By implementing the method of simulated moments (Section 3.3) in MATLAB, we find estimators $\hat{\lambda}^1, \hat{\lambda}^2, \hat{\lambda}^3$ for the three true DNA concentrations as well as an estimator $\hat{\mu}$ for the probability of contamination between two neighboring wells. The MSM is applied to images we get from the laboratory experiments and to synthetic experiments with known true values to test the accuracy of the estimators.

To get a better understanding of the contamination model, we will have a short introduction to percolation theory in Chapter 2.



(a) Almost no clusters observable, meaning there is little sign of contamination. (b) Many clusters of e.g. green and blue wells. It is very likely that clusters occurred due to contamination.

Figure 1

Two pictures of digital copied DNA: three different DNA-sequences (red, green, blue) copied onto a slide. Wells with mixed colors arise from overlapping DNA-sequences.

CHAPTER 2

Percolation Theory

This chapter is based on [Gri99, Chapter 1].

1. Introduction to Percolation

In 1957, Broadbent and Hammersley presented the first 'percolation model', which was intended to investigate questions of the following type:

We immerse a porous stone into water. What is the probability that the centre of the stone is wetted?

Let us consider their model in two dimensions:

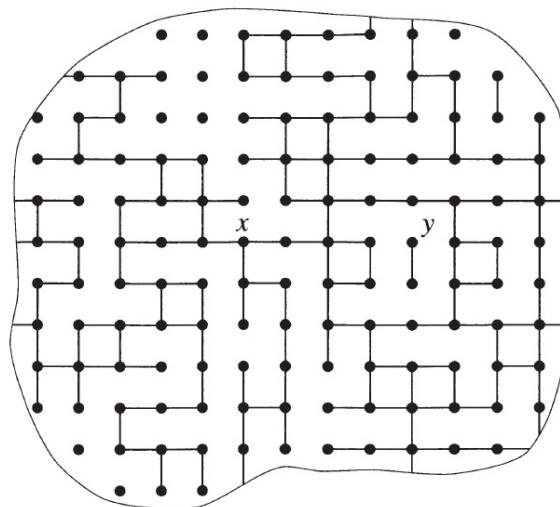
Let \mathbb{Z}^2 be the plane square lattice and $\mu \in \mathbb{R}$ with $0 \leq \mu \leq 1$. We investigate all the edges of neighboring vertices in \mathbb{Z}^2 . Each edge is to be *open* with probability μ and *closed* with probability $1 - \mu$, independently of all other edges. Applying the model to the above example, the passageways inside the stone are represented by the edges in \mathbb{Z}^2 and a passageway is broad enough for water to flow through if an edge is open. Thus μ is the expected proportion of passageways that allow water to pass. The stone itself can be seen as a large, finite subsection of \mathbb{Z}^2 . Let $i \in I$ be a vertex inside the stone. i is wetted if and only if there exists a connection of open edges (called *open path*) that connects i to a vertex on the boundary of the stone.

One main objective of percolation theory is to investigate the existence and size of 'open paths'.

Figure 2 shows a visualization of the stone model where the closed edges are deleted, i.e. we have a random subgraph of \mathbb{Z}^2 .

For large stone sizes, the probability that the centre i of the stone is wetted behaves similarly to the probability of the existence of an infinite open path in \mathbb{Z}^2 , which i is part of. This means that the large-scale penetration of a stone by water is connected to the existence of paths consisting of infinitely many open edges.

Of course, the occurrence of such infinite open clusters depends on the value of μ . For small μ , the different clusters are rather small and isolated. The sizes of the different clusters increase with the value of μ and for μ large enough every vertex is connected to any other vertex by a series of open

**Figure 2**

Possible structure of a two-dimensional porous stone where closed edges were deleted. The open edges are represented by the lines, the vertices by the spots.

In this case, vertex x is connected to the outside of the stone by open edges.

Therefore x would be wetted whereas vertex y remains dry.

Image source: [Gri99, p. 2].

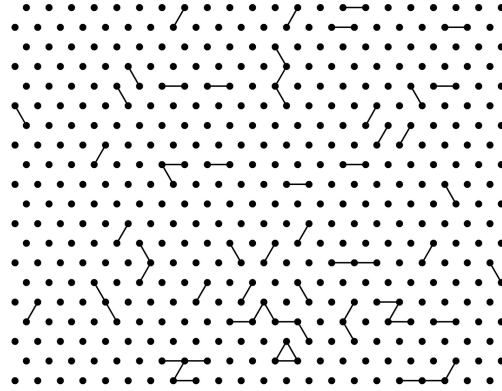
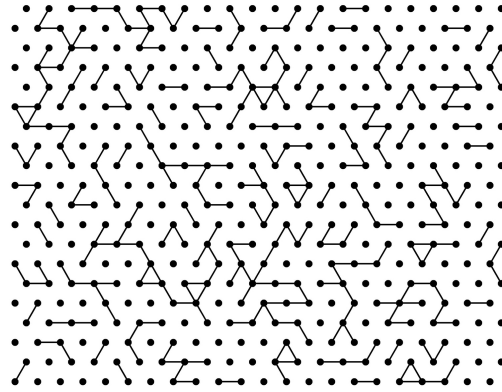
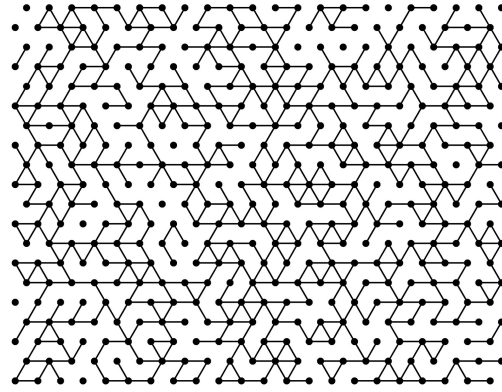
edges. Figure 3 shows three triangular lattices with $\mu = 0.05$, $\mu = 0.2$ and $\mu = 0.5$.

If one could see the whole lattice of \mathbb{Z}^2 , we would be able to observe that for small μ all clusters remain finite whereas for large values of μ an infinite cluster of open edges appears. One goal of percolation theory is the search for a critical value μ_c for the edge-density, so that: for $\mu < \mu_c$ all clusters are finite but for $\mu > \mu_c$ an infinite open cluster will occur, holds almost surely.

To demonstrate the importance of the critical value μ_c , we will have a look at the *Epidemics and fires in orchards*-model proposed in [FH63], which deals with the spread of blight in a large orchard.

Consider a square lattice and imagine that on each vertex a tree is grown. Let μ be a known function of the distance between neighboring trees, representing the probability of a healthy tree being infected by a neighboring blighted tree. The aim is to prevent a single blighted tree from endangering a large proportion of the whole orchard. This is possible by adjusting the space between the planted trees so that μ is smaller than the critical value μ_c .

In the above model we concentrated on two-dimensional problems. To be more general, one could consider some periodic lattice in $d \geq 1$ dimensions

(a) $\mu = 0.05$ (b) $\mu = 0.2$ (c) $\mu = 0.5$ **Figure 3**

Three visualizations of bond percolations on a 20×23 triangular lattice with different μ values (created with MATLAB). Since the same sequence of pseudorandom numbers was used to create each percolation, (a) is a subgraph of (b) and (b) is a subgraph of (c). Whereas for $\mu = 0.05$ only a few small open clusters occur, we can observe that for $\mu = 0.5$ almost all vertices are connected by a series of open edges.

and the probability μ for any edge to be open (and closed otherwise) with $0 \leq \mu \leq 1$. This process is called *bond percolation* since μ determines the probability for a random *edge* to be open. For our estimations and the experiments described in Chapter 1 the two-dimensional bond percolation is the relevant part of percolation theory. We will focus on bond percolation in Section 2 of this chapter.

Another percolation model is the so called *site percolation*. Here, one focuses on the *vertices* rather than the edges in the lattice (all edges are assumed to be open). A random vertex is open with probability μ and closed otherwise. In the stone example above, we could imagine the closed vertices as junctions that prevent water from passing.

Every bond model can be reformulated as a site model on a different lattice. Since this does not hold the other way around, site percolation is more general than bond percolation.

Of course there are more models to think of, such as those where both edges and vertices may be closed ('mixed models') or those where different probabilities apply for different edges to be open ('inhomogeneous models'). However, we do not use those models for our estimations which is why we only mention them here.

Since it is the most significant model for us, we will now focus on bond percolation.

2. Bond Percolation

In this section we will give a short introduction to bond percolation and state some mathematical definitions using basic graph theory. Let $d \geq 1$ be the dimension of the process and $\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$ the set of all integers. Let \mathbb{Z}^d denote the set of all vectors $i = (i^1, i^2, \dots, i^d)$ with integer coordinates. i^s is the s -th coordinate of $i \in \mathbb{Z}^d$.

For the distance $\delta(i, j)$ from vertex i to vertex j , we define

$$\delta(i, j) = \sum_{s=1}^d |i^s - j^s|.$$

For $i \in \mathbb{Z}^d$ and $j \in \mathbb{Z}^d$ we can add edges between all pairs (i, j) with $\delta(i, j) = 1$. This way, we can turn \mathbb{Z}^d into a graph (*d-dimensional cubic lattice*) which we denote by \mathbb{L}^d . We write $\mathbb{L}^d = (\mathbb{Z}^d, \mathbb{E}^d)$, where \mathbb{Z}^d is the set of vertices and \mathbb{E}^d is the set of edges of \mathbb{Z}^d . It is reasonable to think of \mathbb{L}^d as a graph in \mathbb{R}^d , where the end vertices are connected by straight line segments, the edges.

Definition 2.1 (Adjacent/neighbors vertices)

If two vertices i and j are connected by an edge, i.e. if $\delta(i, j) = 1$, we call i and j *adjacent/neighbors* and write $i \sim j$. The corresponding edge is denoted by ξ_{ij} . If the vertex i is an end vertex of the edge ξ_{ij} , ξ_{ij} is called *incident* to i .

We now consider the probability μ that we already mentioned in Section 1 of this chapter. Let $\mu \in \mathbb{R}$ with $0 \leq \mu \leq 1$. Each edge is to be open with probability μ and closed with probability $1 - \mu$, independently of all other edges.

Definition 2.2 (Path)

An alternating sequence $i_0, \xi_{i_0 i_1}, i_1, \xi_{i_1 i_2}, \dots, \xi_{i_{n-1} i_n}, i_n$ of distinct vertices i_s and edges $\xi_{i_s i_{s+1}}$ is called a *path* of \mathbb{L}^d . The path is said to connect i_0 to i_n . The *length* of such path is n .

Definition 2.3 (Circuit)

An alternating sequence $i_0, \xi_{i_0 i_1}, i_1, \xi_{i_1 i_2}, \dots, \xi_{i_{n-1} i_n}, i_n, \xi_{i_n i_0}, i_0$ such that $i_0, \xi_{i_0 i_1}, i_1, \xi_{i_1 i_2}, \dots, \xi_{i_{n-1} i_n}, i_n$ is a path is called a *circuit*. The length of such a circuit is $n + 1$.

A path or circuit with all edges open/closed is called *open/closed path* or *open/closed circuit*. In Figures 2 and 3, only open paths/circuits are visible. One important aspect in bond percolation are the components of open paths or circuits, called *open clusters*.

Definition 2.4 (Open cluster)

The connected components in a random subgraph of \mathbb{L}^d that contains only \mathbb{Z}^d and the open edges, are called *open clusters*. $C(i)$ denotes the open cluster containing the vertex i and is called the *open cluster at i* . The vertices in $C(i)$ are all the vertices of the open cluster that are connected to i by a series of open edges. The set of edges in $C(i)$ consists of all open edges in \mathbb{L}^d that connect neighboring vertices in the open cluster.

Even though $C(i)$ contains both vertices and edges, we will use the term to represent the *set of vertices* only (see Section 5.2).

Although it is not the most efficient method for our MATLAB algorithm in Section 5.2, the following setting can be helpful to create bond model percolation processes:

Let $(U(\xi) : \xi \in \mathbb{E}^d)$ be a family of independent random variables such that each $U(\xi)$ is uniformly distributed on $[0, 1]$.

For $0 \leq \mu \leq 1$, define η_μ by

$$\eta_\mu(\xi) = \begin{cases} 1 & \text{if } U(\xi) < \mu, \\ 0 & \text{if } U(\xi) \geq \mu. \end{cases}$$

The edge ξ is said to be μ -open if $\eta_\mu(\xi) = 1$. It holds

$$P(\eta_\mu(\xi) = 0) = 1 - \mu, \quad P(\eta_\mu(\xi) = 1) = \mu.$$

η_μ can be interpreted as the random outcome of the bond percolation process on \mathbb{L}^d with μ being the probability for an edge to be open.

Obviously $\eta_{\mu_1} \leq \eta_{\mu_2}$ for $\mu_1 \leq \mu_2$ (and $U(\xi)$ fixed), which means that the open edges of the percolation process with edge-probability μ_1 are a subset of the open edges of the percolation process with μ_2 (which you can observe in Figure 3). In other words, if we let μ run increasingly over the interval $[0, 1]$, η_μ will represent different typical configurations of percolation processes with all possible edge-probabilities.

In practice, percolation is one of the simplest models for a disordered medium. It is easy to formulate and yet delivers good qualitative predictions for random media.

We already mentioned that one important aspect in percolation theory is the existence of infinite open clusters that appear almost surely for $\mu > \mu_c$. However, there are many other interesting questions that percolation theory examines:

- What is the mean size of an open cluster?
- How many infinite open clusters will exist almost surely for $\mu > \mu_c$?
- Is there an infinite open cluster for $\mu = \mu_c$?

CHAPTER 3

The Method of Simulated Moments

To get a good understanding of what the *method of simulated moments* is and how it works, we will first give a short introduction to the *method of moments* and the *generalized method of moments*.

1. Method of Moments

The *method of moments* is one of the oldest methods to estimate the unknown components of a parameter vector $\theta = (\theta_1, \theta_2, \dots, \theta_{n_m})^T$.

Let \mathcal{Y} be a random variable and let $\mathcal{Y}_1, \dots, \mathcal{Y}_{n_I}$ be i.i.d. sampling variables with the distribution of \mathcal{Y} . The method of moments is based on the comparison of the theoretical moments and the empirical moments.

For each $r = 1, \dots, n_m$, the r -th (theoretical) moment of the random variable \mathcal{Y} is denoted by

$$M_r = E_\theta[\mathcal{Y}^r].$$

The r -th empirical moment for each $r \in \{1, \dots, n_m\}$ is defined as

$$m_r = \frac{1}{n_I} \sum_{i=1}^{n_I} \mathcal{Y}_i^r.$$

By equating the theoretical with the empirical moments we get a system of equations

$$M_1 = m_1,$$

$$M_2 = m_2,$$

...

$$M_{n_m} = m_{n_m}.$$

The solutions $(\hat{\theta}_1, \dots, \hat{\theta}_{n_m})^T = \hat{\theta}$ of this system are the *moment estimators* for θ .

Example 3.1 (Normal distribution)

Let $\mathcal{Y} \sim N(\mu, \sigma^2)$, with unknown mean $\mu \in \mathbb{R}$ and variance $\sigma^2 > 0$, i.e. we have $\theta = (\mu, \sigma^2)^T$ and $n_m = 2$.

For the normal distribution we have

$$M_1 = E[\mathcal{Y}] = \mu$$

and

$$\begin{aligned} \sigma^2 &= V[\mathcal{Y}] = E[\mathcal{Y}^2] - E[\mathcal{Y}]^2 \\ &= M_2 - M_1^2 \\ \Leftrightarrow M_2 &= \sigma^2 + M_1^2. \end{aligned}$$

Now we equate the theoretical with the empirical moments to get

$$\begin{aligned} M_1 = m_1 &\Rightarrow \mu = \frac{1}{n_I} \sum_{i=1}^{n_I} \mathcal{Y}_i \\ M_2 = m_2 &\Rightarrow \sigma^2 + \mu^2 = \frac{1}{n_I} \sum_{i=1}^{n_I} \mathcal{Y}_i^2. \end{aligned}$$

Solving this system of equations yields the moment estimators

$$\hat{\mu} = \frac{1}{n_I} \sum_{i=1}^{n_I} \mathcal{Y}_i = \bar{\mathcal{Y}}$$

and

$$\hat{\sigma}^2 = \frac{1}{n_I} \sum_{i=1}^{n_I} (\mathcal{Y}_i - \bar{\mathcal{Y}})^2.$$

The method of moments is a very basic and simple estimation method for which the number of unknown parameters equals the number of calculated moments. However, the estimators are often biased. Another disadvantage of the method of moments is that sometimes the moment estimators do not exist, e.g. for a Cauchy-distributed variable \mathcal{Y} it holds $E[\mathcal{Y}] = \infty$ which means the first theoretical moment does not exist [GB06, pp.107-112].

2. Generalized Method of Moments

As the name suggests, the *generalized method of moments* (GMM) is a generalized version of the method of moments. It allows us to find an estimator by defining so called *moment conditions*, i.e. functions that depend on the model parameters and the given data. The main purpose is to estimate the parameter vector by minimizing the sum of squares of the differences between the theoretical moments and the empirical moments.

Note that in Section 1 of this chapter, we saw that when using the method of moments, we need to have the same number of moments as unknown parameters. The GMM can deal with cases where we have more moment conditions than parameters (i.e. the system of equations is overidentified) by introducing a weight matrix Ω .

The methods and proofs described in the remainder of this chapter are inspired by [GM97, Chapters 1 and 2] and [GM90, Properties 1 and 8].

Let $K(\mathcal{Y}_i)$ be an n_m -dimensional function of i.i.d observable variables \mathcal{Y}_i with $i = 1, \dots, n_I$ and let us assume that its expectation under the n_c -dimensional parameter vector θ is

$$k(\theta) = E_\theta[K(\mathcal{Y}_i)],$$

i.e. the entries of k are the generalized moments of the distribution of \mathcal{Y}_i . E_θ denotes the expectation under the true distribution of \mathcal{Y} with parameter θ and θ_0 is the true parameter value.

We introduce some multidimensional function g (the distances between the observed moments and the moments of the model with given parameter values) representing estimating constraints:

$$(3.1) \quad g(\mathcal{Y}_i, \theta) = K(\mathcal{Y}_i) - k(\theta)$$

with

$$(3.2) \quad E_\theta[g(\mathcal{Y}_i, \theta)] = 0 \iff \theta = \theta_0.$$

Definition 3.1

Let Ω be a (n_m, n_m) symmetric positive semi-definite matrix. The GMM estimator $\hat{\theta}(\Omega)$ is then defined as

$$(3.3) \quad \hat{\theta}(\Omega) = \arg \min_{\theta} \left(\sum_{i=1}^{n_I} g(\mathcal{Y}_i, \theta) \right)^T \Omega \left(\sum_{i=1}^{n_I} g(\mathcal{Y}_i, \theta) \right).$$

Proposition 3.1

Under some regularity conditions (see [Han82]), $\hat{\theta}(\Omega)$

- i) is a consistent estimator of θ_0 .
- ii) is asymptotically normal.

The generalized method of moments requires that the moment conditions used have an analytical expression. When an analytical form is not available, it is possible to approximate the moments based on simulations.

3. Method of Simulated Moments

The *method of simulated moments* (MSM) is a simulation based estimation method that can be applied when the moments needed for the GMM do not have an explicit form.

Recalling equation(3.1) and keeping the notations of Section 2 of this chapter, the MSM can be used to approximate k and g by unbiased estimators \tilde{k} and \tilde{g} .

Definition 3.2

Let $\tilde{k}(U_i^s, \theta)$ with $i \in I$ (and $n_I := |I|$) be an unbiased simulator of $k(\theta)$, where U_i^s has a known distribution and $s \in \{1, \dots, n_s\}$ are the different, independent simulations/copies. Similar to equation (3.3), the MSM estimator $\hat{\theta}^{n_I n_s}(\Omega)$ is defined as

$$\begin{aligned}
 \hat{\theta}^{n_I n_s}(\Omega) &= \arg \min_{\theta} \left\{ \sum_{i=1}^{n_I} \left[K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right] \right\}^T \Omega \\
 (3.4) \quad &\times \left\{ \sum_{i=1}^{n_I} \left[K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right] \right\} \\
 &=: \arg \min_{\theta} \Psi_s(\theta).
 \end{aligned}$$

Note that in order to find the MSM estimator, it is important to use the same $U^s = (U_i^s)_{i \in I, s \in \{1, \dots, n_s\}}$ for different values of θ .

The estimator depends on the moments K , the weight matrix Ω , the choice of the simulator \tilde{k} and the number n_s of independent simulations.

It holds $\frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U^s, \theta) \rightarrow E_U[\tilde{k}(U^s, \theta)] = k(\theta)$ for $n_s \rightarrow \infty$, where E_U is the conditional expectation with respect to the distribution of U^s given \mathcal{Y} .

Before we apply the MSM to the laboratory experiments described in Section 1.2, we will state and prove two asymptotic properties of the MSM (see [GM97, p.29]).

Proposition 3.2

Let $\theta \in \Theta$ be an n_c -dimensional vector where Θ is compact. If the number of observable variables n_I tends to infinity and the number of simulations n_s is fixed, then:

- i) the MSM estimator $\hat{\theta}^{n_I n_s}(\Omega)$ is strongly consistent,
i.e. $\hat{\theta}^{n_I n_s}(\Omega) \rightarrow \theta_0$ almost surely for $n_I \rightarrow \infty$.
- ii) if g and $\tilde{g} = K - \tilde{k}$ are differentiable with respect to θ :
 $\sqrt{n_I} \left[\hat{\theta}^{n_I n_s}(\Omega) - \theta_0 \right] \xrightarrow[n_I \rightarrow \infty]{d} N[0, Q_s(\Omega)]$, where

$$Q_s(\Omega) = \Sigma_1^{-1} \Sigma_2 \Sigma_1^{-1} + \frac{1}{n_s} \Sigma_1^{-1} D^T \Omega E_{\theta_0} [V_U[\tilde{g}(\mathcal{Y}, U^s, \theta_0)]] \Omega D \Sigma_1^{-1}$$

with:

$$\begin{aligned} D &= E_{\theta_0} \left[\frac{\partial g}{\partial \theta} \right], \\ \Sigma_1 &= D^T \Omega D, \\ \Sigma_2 &= D^T \Omega V_{\theta_0} [K - k] \Omega D = D^T \Omega V_{\theta_0} [g] \Omega D. \end{aligned}$$

PROOF OF i). Let $\Omega \in \mathbb{R}^{n_m \times n_m}$ be symmetric and positive semi-definite. Define

$$\begin{aligned} \beta(\mathcal{Y}_i, \varrho) &= K(\mathcal{Y}_i) - \varrho \\ \alpha(\eta) &= \eta^T \Omega \eta \end{aligned}$$

and recall $\Psi_s(\theta)$ from (3.4) and three properties that we already mentioned:

- $k(\theta) = E_\theta[K(\mathcal{Y}_i)]$
- $E_U[\tilde{k}(U_i^s, \theta)] = k(\theta)$
- $E_\theta[K(\mathcal{Y}_i) - k(\theta)] = 0 \iff \theta = \theta_0$.

By the strong law of large numbers, we get the almost sure convergence

$$\begin{aligned} \left(\frac{1}{n_I} \right)^2 \Psi_s(\theta) &= \alpha \left(\frac{1}{n_I} \sum_{i=1}^{n_I} \beta \left(\mathcal{Y}_i, \frac{1}{n_s} \sum_{i=1}^{n_s} \tilde{k}(U_i^s, \theta) \right) \right) \\ &\xrightarrow[n_I \rightarrow \infty]{} \alpha \left(E_{\theta_0} \left[E_U \left[\beta \left(\mathcal{Y}_i, \tilde{k}(U_i^s, \theta) \right) \right] \right] \right) \end{aligned}$$

which we assume to be a uniform convergence. β is by definition linear in the second variable, so

$$\alpha \left(E_{\theta_0} \left[E_U [\beta (\mathcal{Y}_i, \tilde{k}(U_i^s, \theta))] \right] \right) = \alpha \left(E_{\theta_0} \left[\beta (\mathcal{Y}_i, E_U [\tilde{k}(U_i^s, \theta)]) \right] \right),$$

which proves the strong consistency since

$$\begin{aligned} \left(\frac{1}{n_I} \right)^2 \Psi_s(\theta) &\xrightarrow{n \rightarrow \infty} \alpha \left(E_{\theta_0} \left[\beta (\mathcal{Y}_i, E_U [\tilde{k}(U_i^s, \theta)]) \right] \right) \\ &= \alpha \left(E_{\theta_0} \left[K(\mathcal{Y}_i) - E_U [\tilde{k}(U_i^s, \theta)] \right] \right) \\ &= \alpha \left(E_{\theta_0} [K(\mathcal{Y}_i)] - E_U [\tilde{k}(U_i^s, \theta)] \right) \\ &= (k(\theta_0) - k(\theta))^T \Omega (k(\theta_0) - k(\theta)), \end{aligned}$$

and $\theta = \theta_0$ is the unique minimum (see equation (3.2)). \square

PROOF OF *ii*). The MMS estimator $\hat{\theta}^{n_I n_s}(\Omega) = (\hat{\theta}_1^{n_I n_s}, \dots, \hat{\theta}_{n_c}^{n_I n_s})^T$ is the minimizer of $\Psi_s(\theta)$ in equation (3.4).

Let $\tilde{g}(\mathcal{Y}_i, U_i^s, \theta) = K(\mathcal{Y}_i) - \tilde{k}(U_i^s, \theta)$. For $r = 1, \dots, n_c$, the first order conditions of the minimization are:

$$\begin{aligned} 0 &= \frac{1}{n_I \sqrt{n_I}} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{\partial \tilde{g}^T}{\partial \theta_r} (\mathcal{Y}_i, U_i^s, \hat{\theta}^{n_I n_s}) \Omega \\ &\quad \times \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}_i, U_i^s, \hat{\theta}^{n_I n_s}), \quad r = 1, \dots, n_c, \end{aligned}$$

where we already multiplied with $\frac{1}{2n_I \sqrt{n_I}}$ so that we will have a good setting for applying the central limit theorem later.

Taking Taylor approximations for each r around θ_0 yields

$$\begin{aligned}
(3.5) \quad 0 &= \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{\partial \tilde{g}^T}{\partial \theta_r}(\mathcal{Y}_i, U_i^s, \theta_0) \Omega \\
&\quad \times \frac{1}{\sqrt{n_I}} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}_i, U_i^s, \theta_0) \\
&\quad + \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{\partial \tilde{g}^T}{\partial \theta_r}(\mathcal{Y}_i, U_i^s, \theta_0) \Omega \\
&\quad \times \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{\partial \tilde{g}}{\partial \theta}(\mathcal{Y}_i, U_i^s, \theta_0) \sqrt{n_I} (\hat{\theta}^{n_I n_s} - \theta_0) \\
&\quad + \sqrt{n_I} (\hat{\theta}^{n_I n_s} - \theta_0)^T \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{\partial^2 \tilde{g}^T}{\partial \theta_r \partial \theta}(\mathcal{Y}_i, U_i^s, \theta_0) \Omega \\
&\quad \times \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}_i, U_i^s, \theta_0) \\
&\quad + \text{higher order terms}
\end{aligned}$$

for $r \in \{1, \dots, n_c\}$.

With $E_{\theta_0}[K(\mathcal{Y}_i)] = E_U[\tilde{k}(U_i^s, \theta_0)]$ we can derive that the last term converges to zero because

$$\frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}_i, U_i^s, \theta_0) = \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \left(K(\mathcal{Y}_i) - \tilde{k}(U_i^s, \theta) \right) \xrightarrow{n_I \rightarrow \infty} 0.$$

For $r = 1, \dots, n_c$ we can combine the remaining terms of (3.5) to an n_c -dimensional approximation

$$\begin{aligned}
(3.6) \quad 0 &\approx \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{\partial \tilde{g}^T}{\partial \theta}(\mathcal{Y}_i, U_i^s, \theta_0) \Omega \\
&\quad \times \frac{1}{\sqrt{n_I}} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}_i, U_i^s, \theta_0) \\
&\quad + \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{\partial \tilde{g}^T}{\partial \theta}(\mathcal{Y}_i, U_i^s, \theta_0) \Omega \\
&\quad \times \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{\partial \tilde{g}}{\partial \theta}(\mathcal{Y}_i, U_i^s, \theta_0) \sqrt{n_I} (\hat{\theta}^{n_I n_s} - \theta_0).
\end{aligned}$$

Now we define D with

$$\begin{aligned} D &= \lim_{n_I \rightarrow \infty} \left\{ \frac{1}{n_I} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{\partial \tilde{g}}{\partial \theta}(\mathcal{Y}_i, U_i^s, \theta_0) \right\} \\ &= E_{\theta_0} \left[E_U \left[-\frac{\partial \tilde{k}}{\partial \theta}(U_i^s, \theta_0) \right] \right] \\ &= E_{\theta_0} \left[\frac{\partial g}{\partial \theta}(\mathcal{Y}_i, U_i^s, \theta_0) \right], \quad \text{since } \frac{\partial K(\mathcal{Y}_i)}{\partial \theta} = 0. \end{aligned}$$

Equation (3.6) turns into

$$(3.7) \quad 0 \approx D^T \Omega \frac{1}{\sqrt{n_I}} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}_i, U_i^s, \theta_0) + D^T \Omega D \sqrt{n_I} (\hat{\theta}^{n_I n_s} - \theta_0).$$

Using the central limit theorem for $n_I \rightarrow \infty$, we have

$$(3.8) \quad \frac{1}{\sqrt{n_I}} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}_i, U_i^s, \theta_0) \xrightarrow{d} N(0, \Sigma_g),$$

where

$$\begin{aligned} \Sigma_g &= V \left[\frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}, U^s, \theta_0) \right] \\ &= V_{\theta_0} \left[E_U \left[\frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}, U^s, \theta_0) \right] \right] + E_{\theta_0} \left[V_U \left[\frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}, U^s, \theta_0) \right] \right] \\ &= V_{\theta_0} [g(\mathcal{Y}, \theta_0)] + \frac{1}{n_s} E_{\theta_0} [V_U [\tilde{g}(\mathcal{Y}, U^s, \theta_0)]] . \end{aligned}$$

Note that we apply the law of total variance for the second equality.

Using (3.8) and transforming (3.7) into

$$\sqrt{n_I} (\hat{\theta}^{n_I n_s} - \theta_0) \approx -(D^T \Omega D)^{-1} D^T \Omega \frac{1}{\sqrt{n_I}} \sum_{i=1}^{n_I} \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{g}(\mathcal{Y}_i, U_i^s, \theta_0)$$

we get that $\sqrt{n_I} (\hat{\theta}^{n_I n_s} - \theta_0)$ converges in distribution to $N(0, Q_s(\Omega))$, where:

$$\begin{aligned} Q_s(\Omega) &= (D^T \Omega D)^{-1} D^T \Omega \Sigma_g \Omega D (D^T \Omega D)^{-1} \\ &= (D^T \Omega D)^{-1} D^T \Omega \left(V_{\theta_0} [g(\mathcal{Y}, \theta_0)] + \frac{1}{n_s} E_{\theta_0} [V_U [\tilde{g}(\mathcal{Y}, U^s, \theta_0)]] \right) \\ &\quad \times \Omega D (D^T \Omega D)^{-1} \\ &= \Sigma_1^{-1} \Sigma_2 \Sigma_1^{-1} + \frac{1}{n_s} \Sigma_1^{-1} D^T \Omega E_{\theta_0} [V_U [\tilde{g}(\mathcal{Y}, U^s, \theta_0)]] \Omega D \Sigma_1^{-1} \end{aligned}$$

with:

$$\begin{aligned} D &= E_{\theta_0} \left[\frac{\partial g}{\partial \theta} \right], \\ \Sigma_1 &= D^T \Omega D, \\ \Sigma_2 &= D^T \Omega V_{\theta_0}[g] \Omega D. \end{aligned}$$

□

The first term $\Sigma_1^{-1} \Sigma_2 \Sigma_1^{-1}$ is the asymptotic covariance matrix of the GMM estimator and the second term is the loss of efficiency we have due to our simulations. For $n_s \rightarrow \infty$, the MSM and the GMM estimator are equivalent. Further information about Ω and its optimal choice can be found in [GM97, pp.31-33].

The asymptotic normality of the MSM estimator is an important property which can e.g. be used to define confidence intervals for θ_0 . However, as we will see in the next chapter, the asymptotic normality will not hold in our case. This is caused by the choice of our moments: they are not differentiable with respect to θ since our variables for the DNA seeding and the edges only take the values 0 and 1.

CHAPTER 4

Applying the MSM

In this chapter we define a mathematical model to apply the MSM to the experimental setup and contamination problem described in Section 1.2.

Let us recall the problem: We have a hexagonal tiling with n_I cavities/wells. For each well we can observe if it contains DNA (up to three different DNA samples) or if the well is empty. However, once two neighboring wells are filled with the same DNA sample(s), we do not know whether DNA was actually seeded into those wells or whether one well was initially empty and got contaminated by its neighbor during the PCR (see Figure 4).

There are different model options to consider:

Contamination between wells can be either **(i)** unidirectional (i.e. the edges are directed and we have independent $\xi_{i \rightarrow j}$ and $\xi_{j \rightarrow i}$ instead of ξ_{ij} as seen in Section 2.2) or **(ii)** symmetric (i.e. undirected edges ξ_{ij}). The edges can be represented by **(1)** independent Bernoulli variables (see Section 2.2,

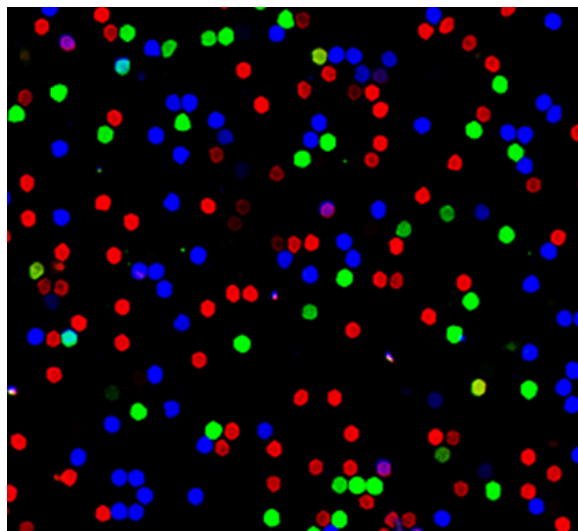
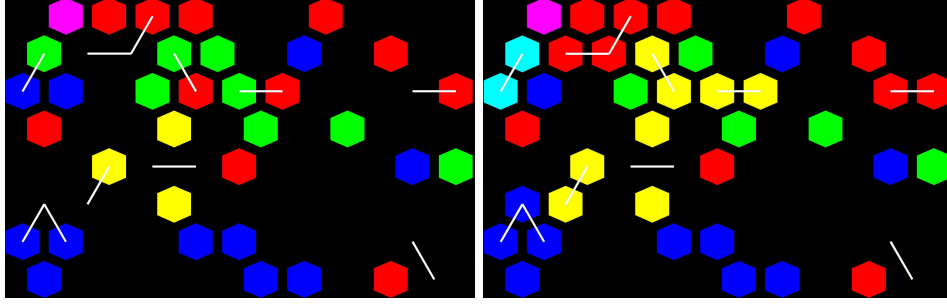


Figure 4

Outcome of a laboratory experiment. For some wells we do not know if they got contaminated by neighboring wells. E.g. the blue cluster at the bottom left could be caused by contamination.



(a) The (usually unknown) state before contamination. The white lines represent open edges (i.e. where contamination will appear). (b) The state after contamination. We can observe that some colors get mixed, e.g. green and blue become cyan (top left) and green and red become yellow (top center).

Figure 5

Synthetic MATLAB experiment for model (ii,1,B) before and after contamination. The open edges and the state before contamination are usually unknown.

definition of $\eta_\mu(\xi)$) or by **(2)** locally correlated random variables. The contamination between wells could be **(A)** limited to direct neighbors or we can allow it to **(B)** spread via a series of open edges.

For our study, we choose the model (ii,1,B) with symmetric contamination that can pass a series of open edges and we let those edges be independent Bernoulli variables. Since the main source for contamination is imperfect sealing by the glass cover this seems to be a reasonable choice (another interesting combination would be (ii,2,B)).

Figure 5, which was created by the MATLAB algorithm for simulations that will be described in Section 5.2, helps to understand the contamination problem for the (ii,1,B) model. It shows a synthetic experiment before and after contamination. Open edges are represented by the white lines. Recall that the color for DNA1 is red, DNA2 is green, DNA3 is blue and mixed colors can occur. In Figure 5(a) only the wells that were initially seeded with DNA are colored. In this case, we could easily calculate the true DNA concentrations by counting the different colors and dividing by the total number of wells. In the real laboratory experiments, Figure 5(a) is unknown and we only see Figure 5(b) (obviously without the white lines). Our goal is to find the best possible estimators for the true concentrations of DNA and for the contamination rate (i.e. the probability that a random edge is open).

The reason why we simulate the moments and apply the MSM is the following: Whether a random well contains a color or remains empty depends on the DNA seeding process as well as on the contamination. Since the contamination in our model is not restricted to direct neighbors, the expected color for each well depends not only on the initial color in the well itself and the neighboring wells but it also depends on neighbors of neighbors and so on (theoretically it depends on all the wells in the grid, even if this usually includes negligible terms). This makes it too complicated to calculate moments (e.g. the expected value for red color of well $i \in I$) and we will try to circumvent this problem using simulations of the moments.

Mathematical model for the experiment and simulations

Let the wells in the hexagonal tiling be denoted by $i \in I$ (with $n_I = |I|$). We have three different DNA samples whose seeding parameters are λ^ℓ , $\ell \in \{1, 2, 3\}$ and the parameter of percolation/contamination is μ . Using this setting, the unknown parameter in the MSM is $\theta_0 = (\lambda_0^1, \lambda_0^2, \lambda_0^3, \mu_0)^T$.

Let us define some variables for the dataset from the laboratory experiment: The (unknown) state after DNA seeding and before contamination (i.e. if well i contains red, green or blue color before contamination, compare Figure 5(a)) is defined as \mathcal{X}_i^ℓ ($i \in I, \ell \in \{1, 2, 3\}$) and we say $\mathcal{X}_i^\ell = 1$ if DNA sample ℓ was seeded into well i and 0 otherwise. Likewise, the state after contamination (see Figure 5(b)) is \mathcal{Y}_i^ℓ , where $\mathcal{Y}_i = (\mathcal{Y}_i^\ell)_{\ell \in \{1, 2, 3\}}$ ($i \in I$). Due to simplicity reasons we assume that for fixed ℓ , the \mathcal{Y}_i^ℓ are identically distributed, i.e. we ignore boundary effects. Obviously we have $\mathcal{X}_i^\ell \leq \mathcal{Y}_i^\ell$.

Now we define the variables for the simulations:

For simulation $s \in \{1, 2, \dots, n_s\}$, well $i \in I$ and DNA sample $\ell \in \{1, 2, 3\}$ we define accordingly: Let $X_i^{\ell, s}$ be the state before and $Y_i^{\ell, s}$ the state after contamination. Moreover we define $Y_i^s = (Y_i^{\ell, s})_{\ell \in \{1, 2, 3\}}$ ($i \in I, s \in \{1, 2, \dots, n_s\}$) and $Y_i = (Y_i^{\ell, s})_{\ell \in \{1, 2, 3\}, s \in \{1, \dots, n_s\}}$ ($i \in I$). Let the edge between wells i and j be denoted by ξ_{ij}^s and $I_2 = \{(i, j) \in I \times I \mid i \sim j, i < j\}$ is the set of pairs of adjacent/neighbors wells. $n_n = |I_2|$ is the total number of such pairs.

Applying the method of simulated moments

Let K be some n_m dimensional function of individual observations \mathcal{Y}_i^ℓ and let k be its expectation under parameter θ : $k(\theta) = E_\theta[K(\mathcal{Y}_i)]$, so that the entries of k are generalised moments of the distribution of \mathcal{Y}_i . Let g represent the distances between observed moments and moments of the model

with given parameter values (which we will approximate via simulations):

$$g(\mathcal{Y}_i, \theta) = K(\mathcal{Y}_i) - k(\theta).$$

As it follows from equation (3.2), $E_{\theta_0}[g(\mathcal{Y}_i, \theta_0)] = 0$.

We try to approximate k and g by unbiased (and as we will see in Section 5.2, also biased) estimators \tilde{k} and $\tilde{g}(\mathcal{Y}_i, U_i^s, \theta) = K(\mathcal{Y}_i) - \tilde{k}(U_i^s, \theta)$. U^s represents the source of randomness for our simulations. In Section 5.2 we will introduce two different options for U_i^s (uniform random variables on $[0, 1]$ and uniform random permutations).

To apply the MSM, we need at least 4 generalized moment conditions since θ is 4-dimensional. We choose the following 9 generalized moments for our estimations with $i \sim j$ and $\ell_1 \neq \ell_2$:

- $E[Y_i^\ell] = P(Y_i^\ell = 1)$, the probability that a random well is colored with red ($\ell = 1$), green ($\ell = 2$) or blue ($\ell = 3$).
- $E[Y_i^{\ell_1} Y_i^{\ell_2}] = P(Y_i^{\ell_1} Y_i^{\ell_2} = 1)$, the probability that a random well is labeled with two colors ((red,green),(red,blue),(green,blue)).
- $E[Y_i^\ell Y_j^\ell] = P(Y_i^\ell Y_j^\ell = 1)$, the probability that two random neighboring wells both are red, green or blue.

Recall that $Y_i^{\ell,s} = 1$ if well i of simulation s is labeled with the corresponding color of DNA sample ℓ and $Y_i^{\ell,s} = 0$ otherwise. We simulate the above moments with

$$\begin{aligned} & \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{1}{n_I} \sum_{i=1}^{n_I} Y_i^{\ell,s}, \\ & \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{1}{n_I} \sum_{i=1}^{n_I} Y_i^{\ell_1,s} Y_i^{\ell_2,s}, \\ & \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{1}{n_n} \sum_{(i,j) \in I_2} Y_i^{\ell,s} Y_j^{\ell,s}, \end{aligned}$$

for $\ell \in \{1, 2, 3\}$. The first simulator sums up all the wells that contain DNA ℓ . This is done for each simulation, i.e. n_s times. The result is divided by the total number of wells in each simulation and by the number of simulations. The other two simulators work the same way, except that for the third simulator, we have to substitute the number of wells by the number of possible edges since these moments focus on the number of neighbors.

For the ease of notation, we define:

$$\begin{aligned}\bar{\mathcal{Y}}^\ell &:= \frac{1}{n_I} \sum_{i=1}^{n_I} \mathcal{Y}_i^\ell, \\ \bar{\mathcal{Y}}^{(\ell_1, \ell_2)} &:= \frac{1}{n_I} \sum_{i=1}^{n_I} \mathcal{Y}_i^{\ell_1} \mathcal{Y}_i^{\ell_2}, \\ \bar{\mathcal{Z}}^\ell &:= \frac{1}{n_n} \sum_{(i,j) \in I_2} \mathcal{Y}_i^\ell \mathcal{Y}_j^\ell\end{aligned}$$

and

$$\begin{aligned}\bar{Y}^{\ell,s} &:= \frac{1}{n_I} \sum_{i=1}^{n_I} Y_i^{\ell,s}, & \bar{Y}^\ell &:= \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{\ell,s}, \\ \bar{Y}^{(\ell_1, \ell_2),s} &:= \frac{1}{n_I} \sum_{i=1}^{n_I} Y_i^{\ell_1,s} Y_i^{\ell_2,s}, & \bar{Y}^{(\ell_1, \ell_2)} &:= \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{(\ell_1, \ell_2),s}, \\ \bar{Z}^{\ell,s} &:= \frac{1}{n_n} \sum_{(i,j) \in I_2} Y_i^{\ell,s} Y_j^{\ell,s}, & \bar{Z}^\ell &:= \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Z}^{\ell,s}.\end{aligned}$$

We recall equation (3.4) for the MSM estimator:

$$\begin{aligned}\hat{\theta}^{n_I n_s}(\Omega) = & \arg \min_{\theta} \left\{ \sum_{i=1}^{n_I} \left[K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right] \right\}^T \Omega \\ & \times \left\{ \sum_{i=1}^{n_I} \left[K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right] \right\}.\end{aligned}$$

Note that for the last 3 moments $\sum_{i=1}^{n_I}$ needs to be replaced by $\sum_{i=1}^{n_n}$ because the moments focus on the number of neighbors (not on the number of total wells). Therefore it is appropriate to divide by the total number of wells (for the first 6 moments) or edges (for the last 3 moments) for weighting reasons. I.e. for the first 6 moments we get

$$\begin{aligned}\hat{\theta}^{n_I n_s}(\Omega) = & \arg \min_{\theta} \left\{ \frac{1}{n_I} \sum_{i=1}^{n_I} \left[K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right] \right\}^T \Omega \\ & \times \left\{ \frac{1}{n_I} \sum_{i=1}^{n_I} \left[K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right] \right\} \\ = & \arg \min_{\theta} \Gamma_s(\theta),\end{aligned}$$

while for the last 3 moments, where the sum is over the total number of

edges n_n , we get:

$$\begin{aligned} \hat{\theta}^{n_I n_s}(\Omega) = & \arg \min_{\theta} \left\{ \frac{1}{n_n} \sum_{i=1}^{n_n} \left[K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right] \right\}^T \Omega \\ & \times \left\{ \frac{1}{n_n} \sum_{i=1}^{n_n} \left[K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right] \right\}. \end{aligned}$$

To make it easier, we first show how $\Gamma_s(\theta)$ looks for our first 3 moments and their simulators $\frac{1}{n_s} \sum_{s=1}^{n_s} \frac{1}{n_I} \sum_{i=1}^{n_I} Y_i^{\ell, s}$:

$$\begin{aligned} & \left(\frac{1}{n_I} \sum_{i=1}^{n_I} \left(K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right) \right)^T \Omega \\ & \times \left(\frac{1}{n_I} \sum_{i=1}^{n_I} \left(K(\mathcal{Y}_i) - \frac{1}{n_s} \sum_{s=1}^{n_s} \tilde{k}(U_i^s, \theta) \right) \right) \\ & = \left(\bar{\mathcal{Y}}^\ell - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{\ell, s} \right)_{l \in \{1, 2, 3\}}^T \Omega \left(\bar{\mathcal{Y}}^\ell - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{\ell, s} \right)_{l \in \{1, 2, 3\}} \\ & = \begin{pmatrix} \bar{\mathcal{Y}}^1 - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{1, s} \\ \bar{\mathcal{Y}}^2 - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{2, s} \\ \bar{\mathcal{Y}}^3 - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{3, s} \end{pmatrix}^T \Omega \begin{pmatrix} \bar{\mathcal{Y}}^1 - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{1, s} \\ \bar{\mathcal{Y}}^2 - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{2, s} \\ \bar{\mathcal{Y}}^3 - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{3, s} \end{pmatrix} \\ & = \sum_{\ell=1}^3 \left(\bar{\mathcal{Y}}^\ell - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{\ell, s} \right)^2 \left(\frac{1}{\bar{\mathcal{Y}}^\ell} \right)^2 = \sum_{\ell=1}^3 \left(\frac{\bar{\mathcal{Y}}^\ell - \bar{Y}^\ell}{\bar{\mathcal{Y}}^\ell} \right)^2, \end{aligned}$$

where $\Omega = \text{diag} \left((1/\bar{\mathcal{Y}}^1)^2, (1/\bar{\mathcal{Y}}^2)^2, (1/\bar{\mathcal{Y}}^3)^2 \right)$. We also assume $\bar{\mathcal{Y}}^\ell \neq 0$, otherwise our MATLAB algorithm will set the corresponding $\left(\frac{\bar{\mathcal{Y}}^\ell - \bar{Y}^\ell}{\bar{\mathcal{Y}}^\ell} \right)^2 = 0$.

Now we apply the MSM estimator to our 9 moments.

Let

$$\begin{aligned} \Omega = & \text{diag}((1/\bar{\mathcal{Y}}^1)^2, (1/\bar{\mathcal{Y}}^2)^2, (1/\bar{\mathcal{Y}}^3)^2, (1/\bar{\mathcal{Y}}^{(1,2)})^2, (1/\bar{\mathcal{Y}}^{(1,3)})^2, \\ & (1/\bar{\mathcal{Y}}^{(2,3)})^2, (1/\bar{\mathcal{Z}}^1)^2, (1/\bar{\mathcal{Z}}^2)^2, (1/\bar{\mathcal{Z}}^3)^2) \end{aligned}$$

and $W := \{(1, 2), (1, 3), (2, 3)\}$.

We define the MSM estimator for our model as the minimizer of:

$$\begin{aligned}
& \left(\begin{array}{c} \left(\bar{\mathcal{Y}}^\ell - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{\ell,s} \right)_{\ell \in \{1,2,3\}} \\ \left(\bar{\mathcal{Y}}^{(\ell_1, \ell_2)} - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{(\ell_1, \ell_2), s} \right)_{(\ell_1, \ell_2) \in W} \\ \left(\bar{\mathcal{Z}}^\ell - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Z}^{\ell,s} \right)_{\ell \in \{1,2,3\}} \end{array} \right)^T \Omega \\
& \times \left(\begin{array}{c} \left(\bar{\mathcal{Y}}^\ell - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{\ell,s} \right)_{\ell \in \{1,2,3\}} \\ \left(\bar{\mathcal{Y}}^{(\ell_1, \ell_2)} - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{(\ell_1, \ell_2), s} \right)_{(\ell_1, \ell_2) \in W} \\ \left(\bar{\mathcal{Z}}^\ell - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Z}^{\ell,s} \right)_{\ell \in \{1,2,3\}} \end{array} \right) \\
& = \sum_{\ell=1}^3 \left(\left(\bar{\mathcal{Y}}^\ell - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{\ell,s} \right)^2 \left(\frac{1}{\bar{\mathcal{Y}}^\ell} \right)^2 \right) \\
& \quad + \sum_{(\ell_1, \ell_2) \in W} \left(\left(\bar{\mathcal{Y}}^{(\ell_1, \ell_2)} - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Y}^{(\ell_1, \ell_2), s} \right)^2 \left(\frac{1}{\bar{\mathcal{Y}}^{(\ell_1, \ell_2)}} \right)^2 \right) \\
& \quad + \sum_{\ell=1}^3 \left(\left(\bar{\mathcal{Z}}^\ell - \frac{1}{n_s} \sum_{s=1}^{n_s} \bar{Z}^{\ell,s} \right)^2 \left(\frac{1}{\bar{\mathcal{Z}}^\ell} \right)^2 \right) \\
& = \sum_{\ell=1}^3 \left(\left(\frac{\bar{\mathcal{Y}}^\ell - \bar{Y}^\ell}{\bar{\mathcal{Y}}^\ell} \right)^2 + \left(\frac{\bar{\mathcal{Z}}^\ell - \bar{Z}^\ell}{\bar{\mathcal{Z}}^\ell} \right)^2 \right) + \sum_{(\ell_1, \ell_2) \in W} \left(\frac{\bar{\mathcal{Y}}^{(\ell_1, \ell_2)} - \bar{Y}^{(\ell_1, \ell_2)}}{\bar{\mathcal{Y}}^{(\ell_1, \ell_2)}} \right)^2.
\end{aligned}$$

In other words, the MSM estimator $\hat{\theta}^{n_I n_s}(\Omega)$ is

$$\begin{aligned}
(4.1) \quad \hat{\theta}^{n_I n_s}(\Omega) = & \arg \min_{\theta} \sum_{\ell=1}^3 \left(\left(\frac{\bar{\mathcal{Y}}^\ell - \bar{Y}^\ell}{\bar{\mathcal{Y}}^\ell} \right)^2 + \left(\frac{\bar{\mathcal{Z}}^\ell - \bar{Z}^\ell}{\bar{\mathcal{Z}}^\ell} \right)^2 \right) \\
& + \sum_{(\ell_1, \ell_2) \in W} \left(\frac{\bar{\mathcal{Y}}^{(\ell_1, \ell_2)} - \bar{Y}^{(\ell_1, \ell_2)}}{\bar{\mathcal{Y}}^{(\ell_1, \ell_2)}} \right)^2.
\end{aligned}$$

Again, we assume $\bar{\mathcal{Y}}^\ell \neq 0$ as well as $\bar{\mathcal{Y}}^{(\ell_1, \ell_2)} \neq 0$ and $\bar{\mathcal{Z}}^\ell \neq 0$, otherwise the corresponding term will be set to zero in our MATLAB algorithm.

This is the estimator we will try to find in Chapter 5.

However, there are some differences between our estimator and the estimator we described in Section 3.3. In contrast to the assumptions in Proposition 3.2, our random variables Y_i^s are not independent due to contamination. For the strong consistency in Proposition 3.2 i) to hold, we need the strong

law of numbers to apply for these weakly dependent variables. We do not state a proof for this here. Dr. Bence Mélykúti is currently working on the proof and will possibly publish it in the near future. Another problem concerning the consistency could be the uniqueness of θ_0 which is not always given. If all wells are white (i.e. red, green and blue), it is obvious that $(\lambda^1, \lambda^2, \lambda^3, \mu)^T = (1, 1, 1, \cdot)^T$ as well as any combination of $\mu = 1$ and arbitrary positive $(\lambda^\ell)_{\ell \in \{1,2,3\}}$ yield the right result and we have an infinite number of possible estimators. Nevertheless, if we look at the tests of our algorithm in Chapter 6, the estimators seem to be strongly consistent (we expect θ_0 to be unique for $n_I \rightarrow \infty$ when $\mu \neq 1$ or $(\lambda^\ell)_{\ell \in \{1,2,3\}} \notin \{0, 1\}$). Also, the asymptotic normality property in Proposition 3.2 ii) is not applicable in our case since the moments we use are not differentiable with respect to θ (we use so called *frequency simulators*, for more information see [GM97, p.96]). Thus we cannot calculate the first order conditions needed in the proof.

Now that we stated the theoretical background of our simulation model, the next chapter will explain how the implemented MATLAB algorithm works.

CHAPTER 5

MATLAB algorithms

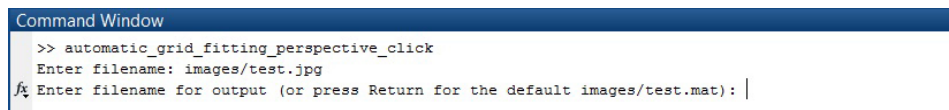
Our MATLAB Code is divided into two parts. The first part extracts information from the picture of the laboratory experiment. We try to calculate the triangular grid and to find out which color(s) belong(s) to each grid point. The information is saved in matrices where the entries represent the grid points. The second part of our algorithm is the implementation of the method of simulated moments. Given the matrices we created in the first part (which represent the experiment picture), we will simulate n_s synthetic experiments and calculate the estimators that most likely led to our experiment picture.

In both sections of this chapter, we will first give a short instruction on how to use the program as a MATLAB user. Afterwards we will briefly explain how the MATLAB algorithm works. If one wants to get more detailed information about the algorithm, the whole MATLAB code including comments for every function can be found in the appendix.

1. Algorithm for the recognition of the grid and colors

MATLAB instructions

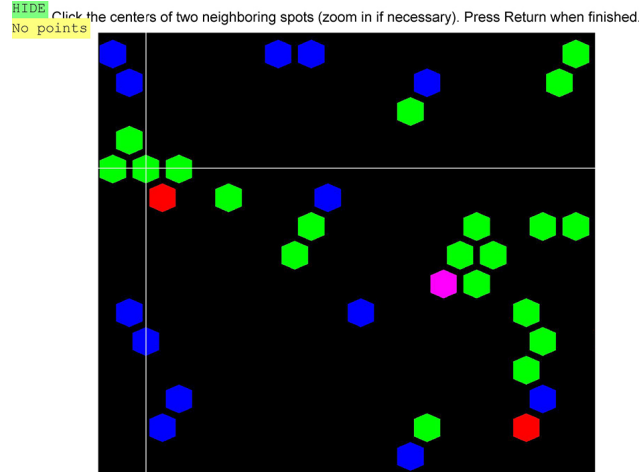
To start the program, we have to open the MATLAB command window, type `'automatic_grid_fitting_perspective_click;'` and press return. We are asked to enter the filename of our experiment picture. Let us assume the picture `'test.jpg'` is saved in the subfolder `'images'` of our MATLAB files. In that case, we type `'images/test.jpg'` (see Figure 6). Once we confirmed the filename we can choose a name for the output file (i.e. the variables that will represent the experiment and that will be needed for the MSM). We can either choose a new name or press return to use the default name.



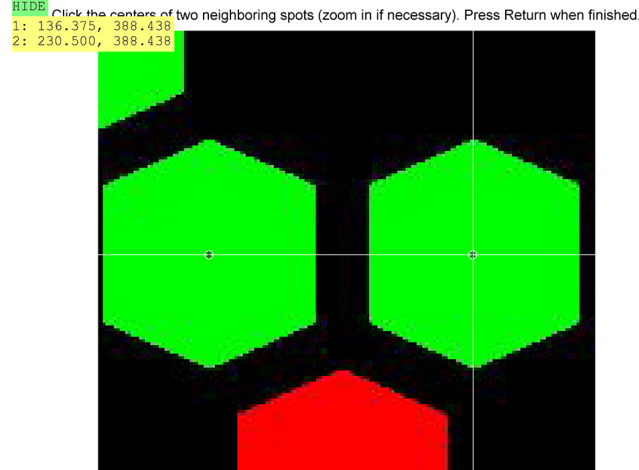
```
Command Window
>> automatic_grid_fitting_perspective_click
Enter filename: images/test.jpg
fx Enter filename for output (or press Return for the default images/test.mat): |
```

Figure 6

MATLAB command to run the grid and color recognition.



(a) The user is asked to click on the centers of two neighboring wells.



(b) To be more accurate, it is possible to zoom in.

Figure 7

A new window showing the input image opens and we are asked to click on the centers of two neighboring spots. Even if it will not affect the results of the MSM, it is recommended to choose two horizontal neighbors if possible. Since these two clicks are used to calculate some initial values for the grid, it is important to be accurate. If needed, we can zoom in before clicking the centers. This process is shown in Figure 7.

Note that for demonstration reasons most of the figures in this section show a synthetic experiment that was created with MATLAB. This is because the hexagonal tiling in laboratory experiments is usually not perfect and our algorithm might have problems recognizing it correctly (as we will see later).

You can add any missing blue, magenta, cyan and white spots by marking their centers. Press Return when done.

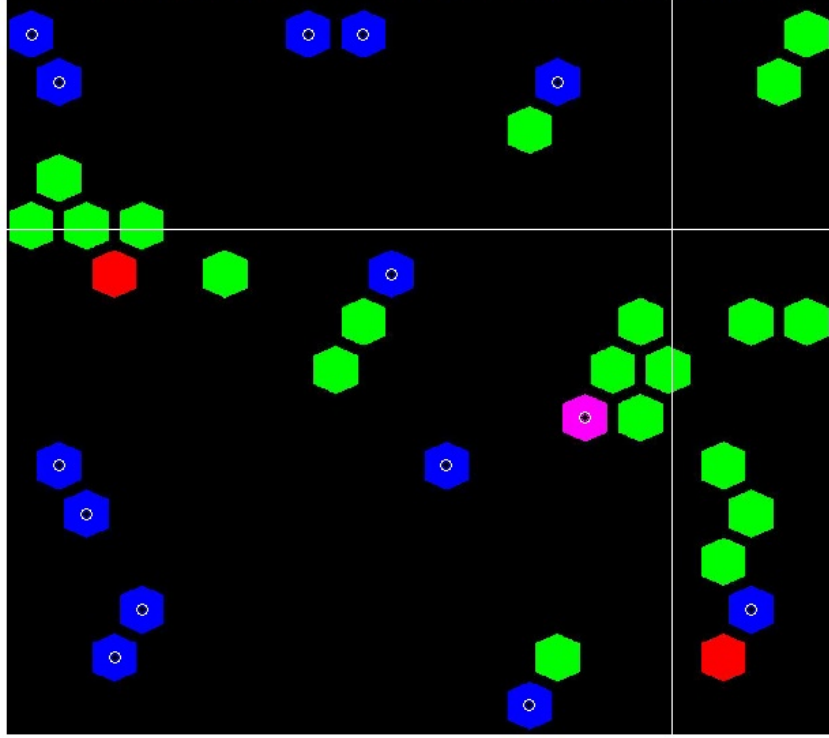
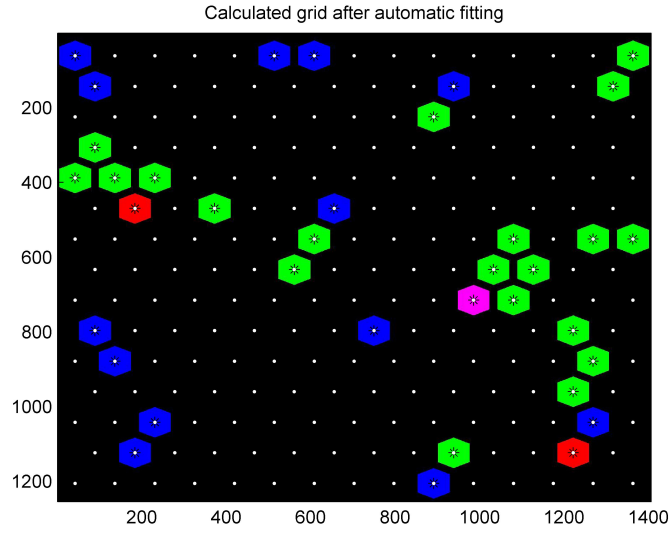


Figure 8

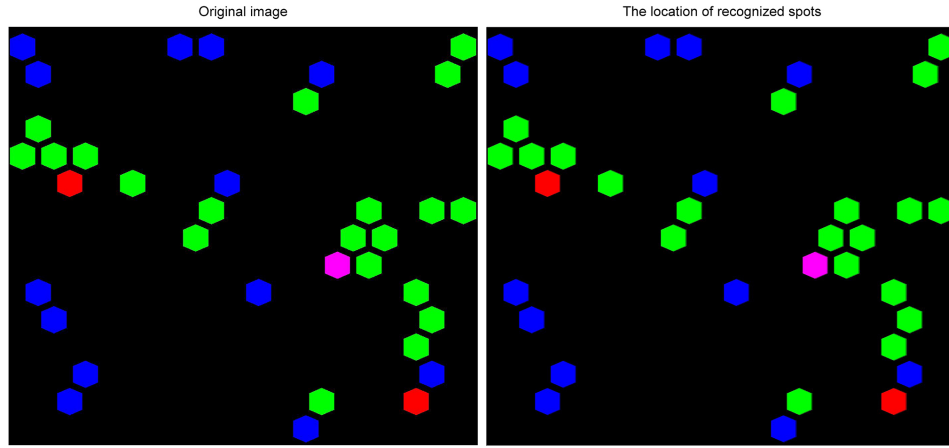
The user is asked to add any missing spots with blue color (i.e. blue, magenta, cyan or white). Since the figure shows a synthetic experiment with perfectly arranged wells, all spots were already recognized by our algorithm. This is usually not the case for laboratory images as we will see later in this section.

Another window pops up showing the recognized spots with red color (i.e. red, yellow, magenta and white). We are asked to add missing spots manually by clicking on their centers and to press enter when done. This step repeats for green and blue color (see figure 8). The centers provide extra information for the fitting of the triangular grid.

Our algorithm now fits the grid and visualizes the result in two ways. One window shows the calculated grid points (Figure 9) and the second one (Figure 10) shows the input image as well as the recognized image (i.e. the image whose information will be saved in the matrices as an output). We are now asked to investigate the grid and the recognized spots to decide whether the results are good enough. If this is the case, a .mat file is saved. This file will be the input parameter for the MSM algorithm in Section 5.2. If the results are not good enough, the user can choose to either start again or quit the program.

**Figure 9**

The calculated grid points (white) are visualized on the original image. As we can see, the grid fitting worked perfectly. For an example where the fitting is not optimal, see Figure 13



(a) The original image.

(b) The image as it was recognized by the algorithm.

Figure 10

We can see that the two images match. This usually means that the spot recognition worked perfectly. An example where some spots are missing or are at the wrong place can be seen in Figure 14

Explanation of the algorithm

The box below shows the code structure of the MATLAB algorithm, i.e. the functions in the order they are used in the code.

Code structure:

```

    automatic_grid_fitting_perspective_click.m
    ginputc.m
    detect_centroids.m (3x)
    ginputc.m
    errorFunctionProjectiveNew.m
    kdtree.m
    compute_transformed_grid_points_perspective.m
    compute_transformed_grid_points_perspective.m
    grid_cropping.m
    color_to_matrix.m

```

Notes:

- *ginputc.m* behaves similarly to the built-in MATLAB function `ginput`, except you can customize the cursor color, line width, and line style.

Source and description: [Dok12]

- *kdtree.m* implements a kdtree for nearest neighbor and range searching. Using MATLAB 2012b and 2013a, the function `kdtree.m` does not work under Windows7 without further adjustments. Instructions on how to use it in Windows can be found in the comment section of [Mic08]. However, using MATLAB R2015b under Linux, `kdtree.m` works without any adjustments.

Source and description: [Mic08]

Without going into great detail, we will briefly explain the most important functions to get an idea about how the algorithm works. Many variables that are important for the algorithm to work will not be mentioned in this section. Details about all variables and functions can be found in the MATLAB code in the appendix.

The algorithm is split up into 3 main parts:

- (1) The Recognition of the centroids of the colored wells.

- (2) Based on two clicks by the user, a triangular grid is created. This grid is deformed by a perspective map and the parameters are optimized until the sum of squared distances between the centroids detected in (1) and the nearest grid point is minimal.
- (3) The color of the experiment image at each optimized grid point is scanned to decide whether the corresponding well contains DNA or is empty.

automatic_grid_fitting_perspective_click.m is the main function for the recognition of the grid and the colors. It calls the input image chosen by the user and creates a perspective mapping of the grid with 8 parameters (this way we allow the usually parallel lines to converge to a vantage point). For this perspective mapping, *automatic_grid_fitting_perspective_click.m* calculates some initial values for the optimization of the triangular grid based on the first two clicks by the user (i.e. the distance between two neighboring centers and the angle for the rotation of the grid around a reference point). To get a good fitting of the true grid, it is important to know the location of as many well-centers as possible. *detect_centroids.m* detects the coordinates of the centers of the wells that contain red, green or blue color. The coordinates are saved in a vector and visualized. The user can add centroids that were not detected to improve the grid fitting (see Figure 12). Since the shape of the triangular grid in the laboratory experiments is usually not perfect, we use 8 parameters to optimize the grid:

- The X and Y coordinates of the first point the user clicked on form the reference point (x_{g0}, y_{g0}) . We expect one grid point to be close to the reference point.
- The distances $r \in \mathbb{R}^2$, i.e. the distance from the reference point to a neighboring well in the same row and the vertical distance from the reference point to a neighbor in the row above (or below), both initialized as the euclidean distance between the first and the second click by the user.
- A vector $\alpha \in (-\pi/2, \pi/2]^2$ representing the rotation of the grid, where the initial values are determined by the angle between the first two clicks.
- A normalization vector $c \in \mathbb{R}^2$ with initial value $(0, 0)$, where c_1 and c_2 are independent variables that are needed for the perspective mapping.

With these initial values, *compute_transformed_grid_points_perspective.m* computes a first grid. Using this grid, *errorFunctionProjectiveNew.m* finds the closest grid points for all detected centroids by calling the function *kdtree.m*. The distance between these centroids and the grid points is then minimized by changing the 8 parameter values to find the optimal grid.

Once we created our fitted grid, the function *grid_cropping.m* deletes rows and columns whose coordinates are completely outside of the image boundaries. What remains is a grid of size (nrows,ncols) where some grid points might still lie outside the image boundaries (see Figure 11(b)). For cases like this, we introduce an 'area of interest matrix' of size (nrows,ncols). The purpose of this matrix is to decide whether a grid point should be part of our calculations or not. In the area of interest matrix, all grid points inside the image boundaries are 1 whilst grid points outside the boundaries are 0, i.e. they will be ignored for further calculations. This way, the amount of ones in the area of interest matrix equals the number of grid points in the image from the laboratory experiment.

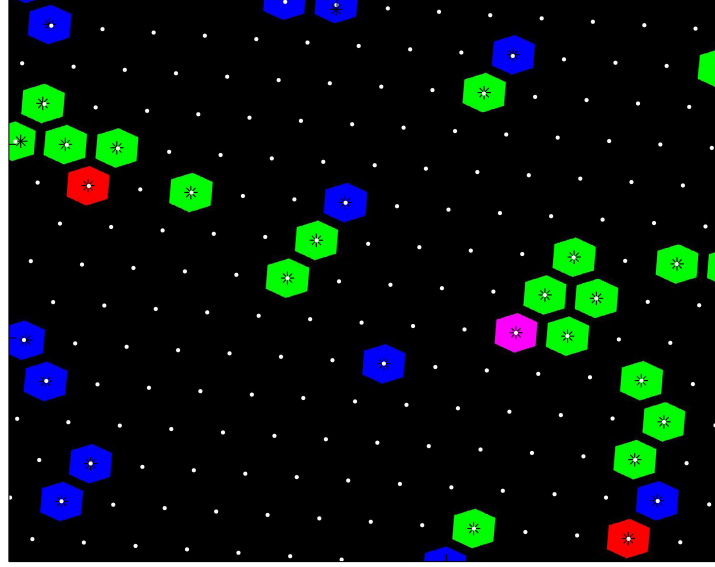
Note that if we want some specific wells not to be part of our calculations, we can manually set the corresponding entries in the area of interest matrix to 0 and they will be ignored in further calculations.

Another important output of *grid_cropping.m* is the variable 'shape', that tells us the shape of the triangular grid, i.e. whether all even or odd numbered rows are shifted to the right.

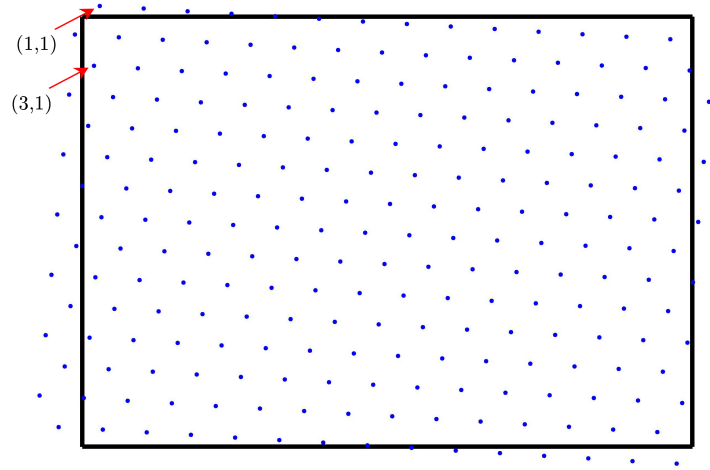
One of the most important variables in both parts of our program is the so called 'experiment_matrix'. It is a matrix of size (nrows,ncols,4), i.e. four matrices of size (nrows,ncols). The first of those matrices is the area of interest matrix. The second matrix represents the red wells: if well (i, j) contains red color, $\text{experiment_matrix}(i, j, 2) = 1$ and 0 otherwise. Accordingly the third matrix represents the green and the fourth matrix the blue wells.

The function *color_to_matrix.m* creates the matrices for the three colors in the following way: We take the coordinates of all optimized grid points that lie inside the area of interest and check the color of the experiment image at those coordinates. E.g. take the coordinates of grid point (i, j) and check the corresponding color channel in our experiment image. If the grayscale value of the red channel exceeds a specified threshold, we set $\text{experiment_matrix}(i, j, 2) = 1$. If there is no sign of red color, it remains 0. The same happens for green and blue color, i.e. for $\text{experiment_matrix}(i, j, 3)$ and $\text{experiment_matrix}(i, j, 4)$.

Finally, *automatic_grid_fitting_perspective_click.m* visualizes the recognized



(a) The triangular grid is not always perfectly straight.



(b) The blue spots show the fitted grid after cropping. The black frame represents the boundaries of our experiment image. The grid points outside the image boundaries are set to zero in the so called 'area of interest matrix' and will be ignored for further calculations. E.g. entry $(1,1)$ is 0 whilst entry $(3,1)$ is 1.

Figure 11

Since we optimize the grid with our 8 parameters, the algorithm can deal with tilted grids like this.

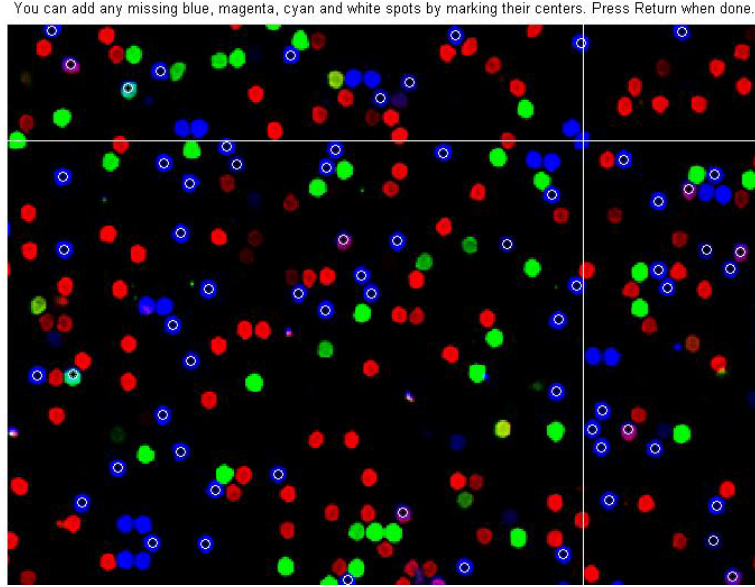


Figure 12

Many blue spots could not be recognized automatically, e.g. the cluster consisting of five blue wells at the bottom left.

grid and a comparison between the original experiment image and the recognized image (see Figures 9, 10, 13 and 14).

The variable 'experiment_matrix' containing the area of interest and information about the wells and their colors can be seen as a representation of the original experiment image. 'experiment_matrix' and 'shape' hold the data we need for the MSM in the second part of the program. They are saved in a .mat file which can be used as an input to the MSM.

The recognition of the grid and colors is not optimal yet and we will now state some shortcomings.

In Figure 8 we saw that the recognition of blue spots worked perfectly and every spot was recognized by our algorithm. However, Figure 12 shows another example (of a laboratory experiment) where some spots in the image were not recognized and the user can manually add missing centers. This problem is caused by bad resolution and the fact that the colors of some neighboring wells merge and there is no space in between.

The images we get from the laboratory experiments are usually not based on a perfect triangular grid which leads to problems concerning the recognition of the grid and colors.

Figure 14(a) shows a laboratory experiment image. We can observe that

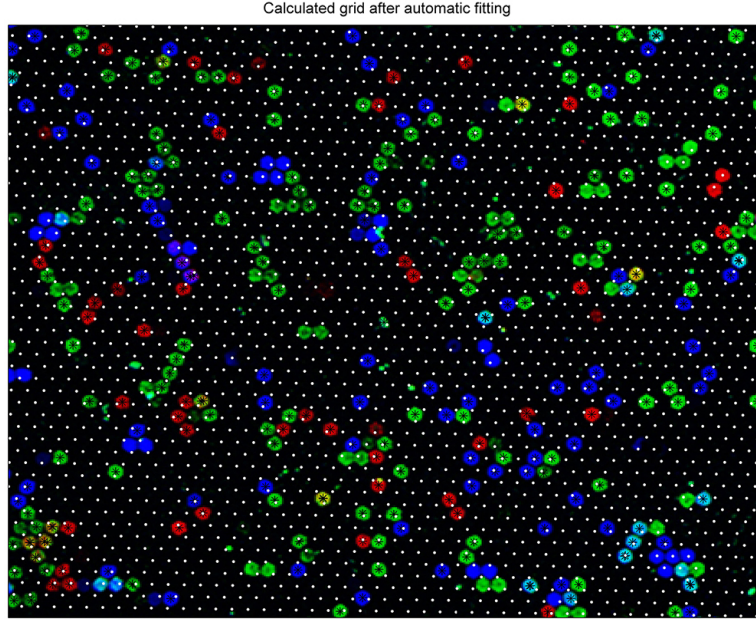
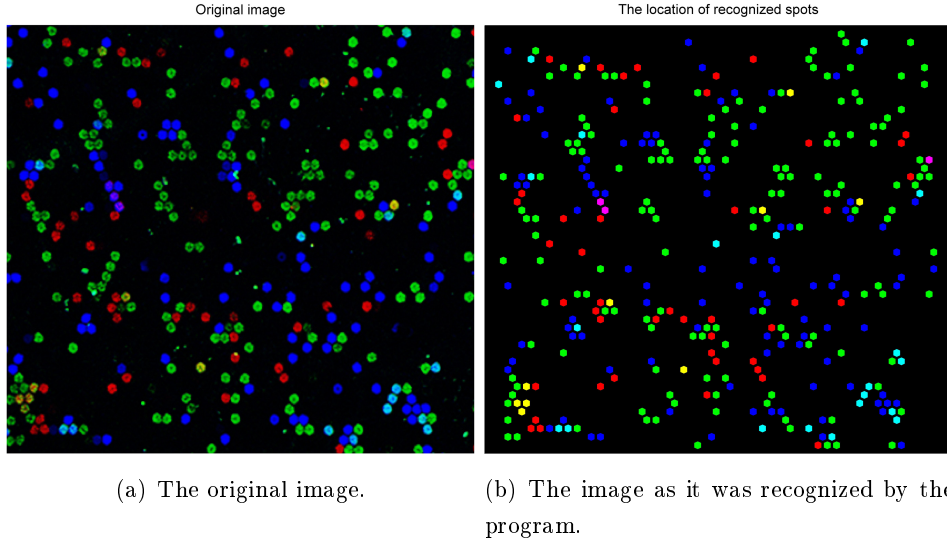


Figure 13

Grid recognition of an experiment image. Due to the lack of symmetry of the spots and the partly curved grid, the fitting is not optimal.

many wells do not have a symmetric form which makes it hard to compute the true centers. In addition, the grid seems to be curved at some places. Those two properties lead to an imprecise grid recognition as seen in Figure 13. Figure 14 shows the resulting comparison between the original experiment image and the recognized image.

We have seen that the grid and color recognition works for perfectly shaped triangular grids. Our algorithm only starts to have some difficulties if the quality of the experiment image is not good enough.

**Figure 14**

We can see that some colors are located in the wrong place whilst some other colored wells were not even detected.

2. Implementation of the MSM

MATLAB instructions

To start the program we need to call the function *msm.m* which needs the following 3 input variables:

- the number of simulations (n_s in Chapter 4),
- the number of optimizations n_{opt} with different initial values for $(\lambda^1, \lambda^2, \lambda^3, \mu)^T$,
- an upper limit μ_{max} for the initial value of the contamination rate μ (note that this is just an initial value, it is not a constraint for the estimator of μ_0).

In the evaluation of our tests in Chapter 6 we show some examples of how to possibly choose the 3 variables.

In general, the estimators get better as the number of simulations n_s increases.

μ_{max} should be chosen intuitively by the user depending on the experiment image. E.g. $\mu_{max} = 0.1$ means that we do not expect more than 10% of all possible edges to be open.

The initial values for the different optimizations $r \in \{1, \dots, n_{opt}\}$ are determined as follows:

We calculate $\lambda_{max}^\ell = \frac{1}{n_I} \sum_{i=1}^{n_I} \mathcal{Y}_i^\ell$ for $\ell \in \{1, 2, 3\}$. Note that the true DNA concentration is $\leq \lambda_{max}^\ell$ since it represents the concentration for $\mu = 0$, i.e. when there is no contamination. We define the initial values for the first optimization as

$$\begin{aligned}\lambda_{ini_1}^\ell &= \lambda_{max}^\ell, \\ \mu_{ini_1} &= 0\end{aligned}$$

for $\ell \in \{1, 2, 3\}$. If $n_{opt} > 1$, the initial values for optimization r are

$$\begin{aligned}\lambda_{ini_r}^\ell &= \lambda_{ini_{r-1}}^\ell - \frac{\lambda_{max}^\ell}{n_{opt}}, \\ \mu_{ini_r} &= \mu_{ini_{r-1}} + \frac{\mu_{max}}{n_{opt} - 1}\end{aligned}$$

for $\ell \in \{1, 2, 3\}$ and $r \in \{2, \dots, n_{opt}\}$, i.e. the initial values for the DNA concentrations decrease linearly (note that they are always positive) as the initial values for the contamination increase linearly until $\mu_{ini_{n_{opt}}} = \mu_{max}$. E.g. if we have $(\lambda_{max}^1, \lambda_{max}^2, \lambda_{max}^3, \mu_{max})^T = (0.06, 0.12, 0.09, 0.06)^T$ and we choose $n_{opt} = 3$, i.e. we get 3 simulations, the initial values would be

$$\begin{aligned}(\lambda_{ini_1}^1, \lambda_{ini_1}^2, \lambda_{ini_1}^3, \mu_{ini_1})^T &= (0.06, 0.12, 0.09, 0)^T, \\ (\lambda_{ini_2}^1, \lambda_{ini_2}^2, \lambda_{ini_2}^3, \mu_{ini_2})^T &= (0.04, 0.08, 0.06, 0.03)^T, \\ (\lambda_{ini_3}^1, \lambda_{ini_3}^2, \lambda_{ini_3}^3, \mu_{ini_3})^T &= (0.02, 0.04, 0.03, 0.06)^T.\end{aligned}$$

For the number of optimizations we suggest $n_{opt} \leq 100\mu_{max}$. Tests showed that the estimators do not show significant improvement for more optimizations but the algorithm takes longer to calculate as n_{opt} increases. In general it holds: The bigger the grid, the less optimizations are needed.

For demonstration reasons we let the number of simulations be 20, the number of optimizations 6 and $\mu_{max} = 0.1$. In this case we would open the MATLAB command window and type `'msm(20,6,0.1)'`.

We are asked to enter the filename for the .mat file that was created in the

first part of the program. E.g. if our file is 'test.mat' and it is located in the subfolder 'images' of our MATLAB files, we have to type 'images/test.mat' (see yellow box in Figure 15). Once we confirmed the filename, we can choose a name for the output file which will contain our estimators and some other variables. We can either choose a new name or press return to use the default name.

MATLAB now starts the simulations and calculates the best estimator of all optimization processes. Figure 15 displays the command window after the best estimator has been found. The green box shows our estimator $\hat{\theta}^{n_{ins}}$. The column 'optim number' simply shows the number of the current optimization (recall that we chose to have 6 optimizations with different initial values), i.e. each row represents one optimization process. Columns 2, 3, 4, 5 show the different initial values for $(\lambda^1, \lambda^2, \lambda^3, \mu)^T$ and columns 7, 8, 9, 10 list the corresponding values after each optimization. Columns 6 and 11 concern the objective function of the MSM that is to be minimized. Column 6 shows its value with the given initial estimators while column 11 represents the value of the objective function for the optimized estimators.

As we can see, our minimal value for the objective function is 0.1085 (row 3, column 11).

Accordingly our estimator is $\hat{\theta}^{n_{ins}} = (0.0095, 0.0746, 0.0358, 0.0531)^T$.

Note that the variable 'abst' in Figure 15 shows the normalized squared distances between the empirical moments and the simulated moments for each of our 9 moments. In our program, we set the moments 4, 5 and 6 to zero (the moments that concentrate on the probability that a random well is labeled with two colors), because we suspect that they do not lead to any improvement of the estimators. Those moments can be easily set to be non-zero by removing the two '0*' in lines 104 and 106 of *optim.m*. The sum of the 9 entries in *abst* equals the minimal objective function value of 0.1085 (up to a rounding error of 0.0001).

```

Command Window
>> msm(20,6,0.1);
Filename for data set: images/test.mat
Enter filename for output (or press Return for the default images/test_estimators.mat):

optimizations =

Columns 1 through 6

    'optim number'    'lambda_1_ini'    'lambda_2_ini'    'lambda_3_ini'    'mu_ini'    'error_ini'
    [         1]    [    0.0138]    [    0.0917]    [    0.0642]    [         0]    [    0.6447]
    [         2]    [    0.0115]    [    0.0765]    [    0.0535]    [    0.0200]    [    0.3445]
    [         3]    [    0.0092]    [    0.0612]    [    0.0428]    [    0.0400]    [    0.3671]
    [         4]    [    0.0069]    [    0.0459]    [    0.0321]    [    0.0600]    [    0.3985]
    [         5]    [    0.0046]    [    0.0306]    [    0.0214]    [    0.0800]    [    0.6945]
    [         6]    [    0.0023]    [    0.0153]    [    0.0107]    [    0.1000]    [    1.5246]

Columns 7 through 11

    'lambda_1'    'lambda_2'    'lambda_3'    'mu'    'error'
    [    0.0138]    [    0.0917]    [    0.0642]    [         0]    [    0.6447]
    [    0.0096]    [    0.0826]    [    0.0435]    [    0.0440]    [    0.1198]
    [    0.0095]    [    0.0746]    [    0.0358]    [    0.0531]    [    0.1085]
    [    0.0089]    [    0.0574]    [    0.0278]    [    0.0690]    [    0.1576]
    [    0.0041]    [    0.0438]    [    0.0215]    [    0.0841]    [    0.4442]
    [    0.0038]    [    0.0265]    [    0.0072]    [    0.1210]    [    0.5944]

abst =

    0.0136    0.0095    0.0522
         0         0         0
    0.0000    0.0154    0.0177

result =

    'estimators'
    [0.0095]
    [0.0746]
    [0.0358]
    [0.0531]

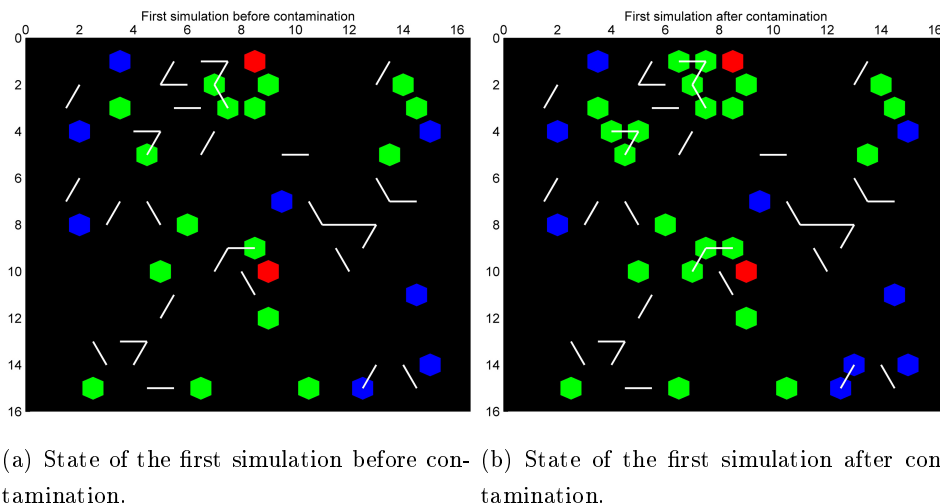
The file images/test_estimators.mat was created successfully.
>>

```

Figure 15

MATLAB command window for the method of simulated moments. The values in the green box are our estimators $(\hat{\lambda}^1, \hat{\lambda}^2, \hat{\lambda}^3, \hat{\mu})^T$.

Apart from the command window, another window (see Figure 16) opens that visualizes the first out of n_s simulations before and after contamination and the contamination clusters. It is displayed as a typical realization of the process, generated with our MSM estimator $\hat{\theta}^{n_s}$.

**Figure 16**

Visualization of the first simulation with values $(0.0095, 0.0746, 0.0358, 0.0531)^T$.

The white lines show where contamination takes place.

In this example, we can observe that the different values of the objective function are not very close to zero (see Figure 15). This is due to the fact that our test-grid of size $(15, 15)$ is rather small. In Chapter 6 we will see that those values will get closer to zero as the grid size increases.

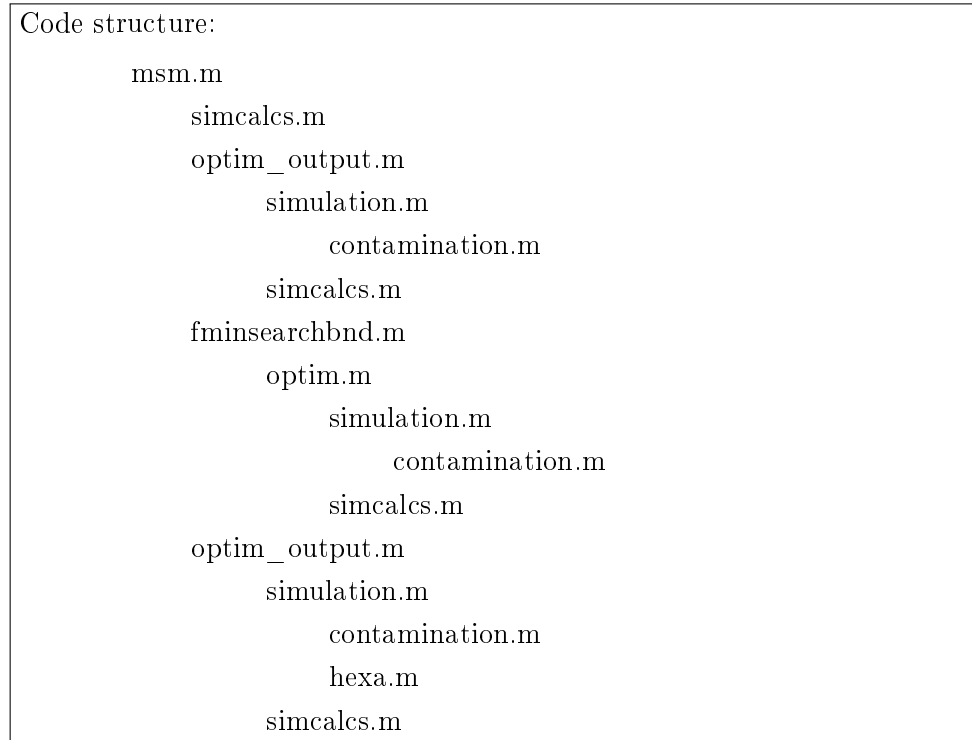
In the final step of the algorithm, the already mentioned .mat file that contains the MSM estimator and some other variables (see appendix for more information) is saved.

The output variables are:

- *errors*: the values of the objective function for a trivial estimator $(\mu = 0, \lambda^\ell = \sum_{i=1}^{n_I} \sum_{s=1}^{n_s} \frac{Y_i^{\ell,s}}{n_I n_s} \text{ for } \ell \in \{1, 2, 3\})$ and for the MSM estimator $\hat{\theta}^{n_I n_s}$.
- *estimators*: the MSM estimator $\hat{\theta}^{n_I n_s} = (\hat{\lambda}^1, \hat{\lambda}^2, \hat{\lambda}^3, \hat{\mu})^T$
- *input_variables*: the number of simulations, number of optimizations and μ_{max} chosen by the user.
- *max_edges*: (nrows,ncols,3) matrix containing information about if an edge can possibly be open.
- *time*: the elapsed time.
- *wells*: (nrows,ncols,4) matrix with information about the area of interest and the colors in each well of the visualized simulation after contamination.

Explanation of the algorithm

The box below shows the code structure of the MATLAB algorithm, i.e. the functions in the order they are used in the code.



Note:

fminsearchbnd.m behaves similarly to the built-in MATLAB function *fminsearch*, except you can add bound constraints. Source and description: [D'E12]

Just like in Section 1 of this chapter, we will briefly explain how the MATLAB algorithm works. For more detailed information, the whole MATLAB code including comments on every function and variable can be found in the appendix.

msm.m is the main function for the parameter estimation by the method of simulated moments.

It calls the variables 'experiment_matrix' and 'shape' that represent the data set of the experiment (these variables are stored in the .mat file that

was created by *automatic_grid_fitting_perspective_click.m*).

Before we continue to explain the algorithm itself, we will explain how we handle the edges in our grid.

It is important to have a method that lets us access all edges in the grid uniquely. The triangular shape suggests the following method: For each vertex/well $i \in I$ we have three edges that can lead to possible contamination: One edge connecting i and its right neighbor, one edge connecting i and its down-right neighbor and one edge connecting i and its down-left neighbor (see Figure 17). The boundary wells form a special case. E.g. the last well in each row does not have a right neighbor i.e. a contamination to the right cannot exist. Our algorithm deletes those extra edges and they are not part of any calculations.

We define three sets for our edges:

$$I_r = \{(i, j) \in I \times I \mid j \text{ is right neighbor of } i\}$$

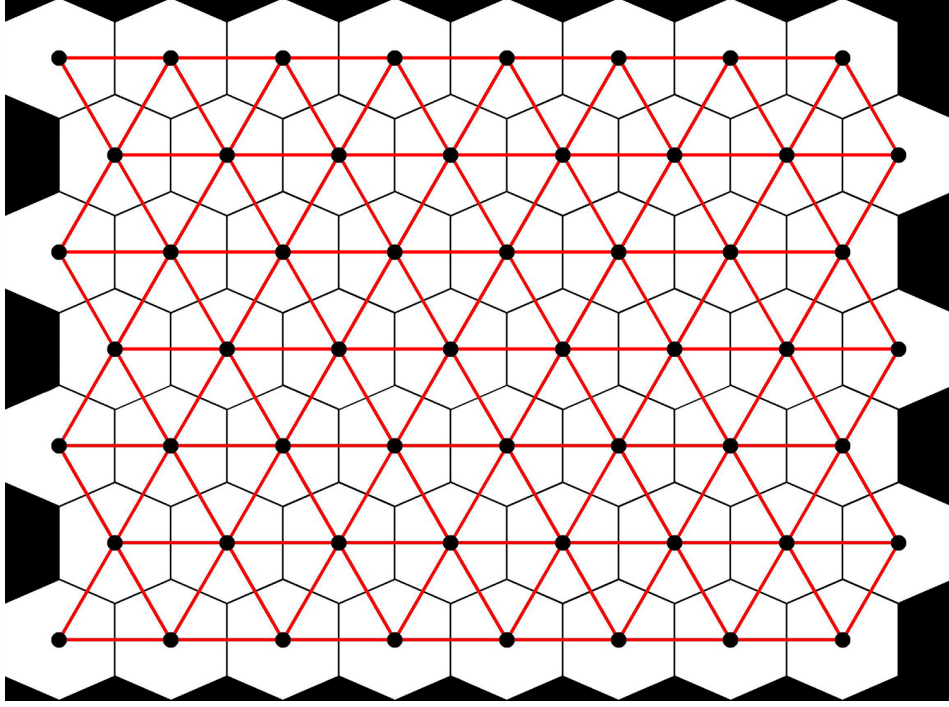
$$I_{dr} = \{(i, j) \in I \times I \mid j \text{ is down-right neighbor of } i\}$$

$$I_{dl} = \{(i, j) \in I \times I \mid j \text{ is down-left neighbor of } i\}$$

To be able to simulate the seeding process of DNA and the open edges, we will create some random variables U_i^s . As mentioned in Chapter 4, we can do this in two different ways:

Method 1):

We let $U_i^{v,s}$ be independent uniform random variables on $[0, 1]$ where $i \in \{1, \dots, n_I\}$, $v \in \{1, \dots, 6\}$ and $s \in \{1, \dots, n_s\}$. $U_i^{1,s}, U_i^{2,s}$ and $U_i^{3,s}$ represent the source of randomness for the seeding of the three DNA samples. $U_i^{4,s}, U_i^{5,s}$ and $U_i^{6,s}$ form the source of randomness for the edges: $U_i^{4,s}$ concerns the edge from well i in simulation s to the right, $U_i^{5,s}$ concerns the

**Figure 17**

Triangular grid with the corresponding edges (red). Every vertex (except the ones on the boundaries) has three edges leading to the right, down-right and down-left. We connect those edges and take away the unnecessary edges for boundary vertices to get the set of all edges.

down-right edge and $U_i^{6,s}$ the down-left edge. Similarly to $\eta_\mu(\xi)$ in Section 2.2, we define for $s \in \{1, \dots, n_s\}$:

$$\begin{aligned}
 X_i^{\ell,s} &= \begin{cases} 1 & \text{if } U_i^{\ell,s} < \lambda^\ell \\ 0 & \text{if } U_i^{\ell,s} \geq \lambda^\ell \end{cases} \text{ for } \ell \in \{1, 2, 3\}, \\
 \eta_\mu(\xi_{ij}^s) &= \begin{cases} 1 & \text{if } U_i^{4,s} < \mu \\ 0 & \text{if } U_i^{4,s} \geq \mu \end{cases} \text{ for } (i, j) \in I_r, \\
 \eta_\mu(\xi_{ij}^s) &= \begin{cases} 1 & \text{if } U_i^{5,s} < \mu \\ 0 & \text{if } U_i^{5,s} \geq \mu \end{cases} \text{ for } (i, j) \in I_{dr}, \\
 \eta_\mu(\xi_{ij}^s) &= \begin{cases} 1 & \text{if } U_i^{6,s} < \mu \\ 0 & \text{if } U_i^{6,s} \geq \mu \end{cases} \text{ for } (i, j) \in I_{dl}
 \end{aligned}$$

for given parameters $(\lambda^1, \lambda^2, \lambda^3, \mu)^T$. We recall that $(X_i^{\ell,s})_{\ell \in \{1,2,3\}}$ represents the state of well i before contamination (e.g. if red color/DNA1 was seeded into well i of simulation s , we have $X_i^{1,s} = 1$ and 0 otherwise) and that edge

ξ_{ij}^s is open for $\eta_\mu(\xi_{ij}^s) = 1$ and closed for $\eta_\mu(\xi_{ij}^s) = 0$.

One feature of this method is that it generates binomially distributed random numbers for the amount of colored wells and open edges, i.e. we have rather high standard deviations. E.g. let us assume the probability for an edge to be open is $\mu = 0.1$ and the number of total edges is $n_n = 50000$. The binomial variable has standard deviation $\sqrt{n_n\mu(1-\mu)} = 67.08$ and the mean is 5000. This means that due to random fluctuations it would not be surprising to get anywhere around 5000 ± 67 open edges in a simulation. However, e.g. 5067 open edges would go along with $\mu = 5067/50000 = 0.1013$ instead of 0.1.

We try to circumvent this problem by introducing another method that is biased on the one hand but reduces variance and delivers the 'right amount' of open edges and colored wells (i.e. 5000 in the example above) on the other hand.

Method 2):

In this method, we use random permutations to generate the number of seeds and open edges. Note that this method to create simulators is biased, as we will see later. For each simulation and color, we permute the indices of all wells and all edges and save those permutations in a variable (it is important for the MSM that the same permutations are used for different values of θ , see Definition 3.2 in Section 3.3).

We define a vector $w = (1, \dots, n_I)$ that contains the indices of all wells. Using uniform random permutations $\sigma(\cdot) \in S_{n_I}$ (where S_{n_I} is the symmetric group on n_I elements) for each color and simulation, we get new vectors

$$(\tilde{w}^{\ell,s})_{\ell \in \{1,2,3\}, s \in \{1, \dots, n_s\}} = (\sigma^{\ell,s}(1), \dots, \sigma^{\ell,s}(n_I))_{\ell \in \{1,2,3\}, s \in \{1, \dots, n_s\}}.$$

Let $\zeta(\cdot)$ be a function that rounds to the nearest integer. We define

$$(\tilde{U}^{\ell,s})_{\ell \in \{1,2,3\}, s \in \{1, \dots, n_s\}} = (\sigma^{\ell,s}(1), \dots, \sigma^{\ell,s}(\zeta(\lambda^\ell n_I)))_{\ell \in \{1,2,3\}, s \in \{1, \dots, n_s\}}$$

with $|\tilde{U}^{\ell,s}| = \zeta(\lambda^\ell n_I)$.

Since $\zeta(\lambda^\ell n_I)$ is the closest integer to the expected number of cavities seeded with DNA sample ℓ , we choose

$$X_g^{l,s} = \begin{cases} 1 & \text{if } g \in \tilde{U}^{\ell,s} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } \ell \in \{1, 2, 3\} \text{ and } s \in \{1, \dots, n_s\}.$$

This method achieves that the difference between seeded wells and the expected number of seeded wells is at most 0.5 for each simulation.

The same technique is applied for the edges:

For all simulations, we create a vector with all elements of $\{I_r \cup I_{dr} \cup I_{dl}\}$, i.e. with all edges that can possibly be open: $z = (\xi_1, \dots, \xi_{n_n})$. Again, we use uniform random permutations $\gamma(\cdot) \in S_{n_n}$ to get new vectors

$$(\tilde{z}^s)_{s \in \{1, \dots, n_s\}} = (\xi_{\gamma(1)}^s, \dots, \xi_{\gamma(n_n)}^s)_{s \in \{1, \dots, n_s\}}.$$

We take the first $\zeta(\mu n_n)$ elements to define

$$\tilde{U}^{4,s} = \{\xi_{\gamma(1)}^s, \dots, \xi_{\gamma(\zeta(\mu n_s))}^s\} \quad \text{for } s \in \{1, \dots, n_s\}$$

with $|\tilde{U}^s| = \zeta(\mu n_n)$, which is the closest integer to the expected value of open edges for given μ and n_n . We choose

$$\eta(\xi_h^s) = \begin{cases} 1 & \text{if } h \in \tilde{U}^{4,s} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } s \in \{1, \dots, n_s\}.$$

and choose the number of open edges in our simulation by this definition.

As mentioned earlier, this simulation of the open edges and the seeding process is biased since it generates (weakly) dependent variables. E.g. if we know for all except one edge whether it is open or closed, we can infer the state of the last edge.

Using this method can be interpreted as a trick. We get a trade-off between bias and decreased variance. Indeed, tests in Chapter 6 show that method 2) seems to deliver more accurate estimators, which is why we choose to work with this method.

Once we created the matrices for the seeding and the open edges, the function *simcalcs.m* calculates the observed moments $K(Y_i)$ for the data set (using the variables 'experiment_matrix' and 'shape' that represent the experiment image).

Via the above *method 2)*, the function *simulation.m* provides matrices for

the seeding and open edges for given values θ . Using those matrices, *contamination.m* generates the contamination clusters and saves the vertices of each cluster in a vector (compare to $C(i)$ in Chapter 2.2). The process of contamination with the calculated paths is implemented in *simulation.m*. The resulting matrices form our synthetic experiments, i.e. our simulations. They contain the information about the state $\mathcal{Y}_i^{\ell,s}$ of each well after contamination.

In the next step, *simcalcs.m* calculates the simulated moments \tilde{k} . The function *optim.m* computes the value of the objective function. By using *minsearchbnd.m* the objective function gets minimized and we obtain an optimized estimator for our initial values.

In case the user decides to choose more than one optimization process, the whole procedure starts again. The difference between the various optimization processes lies in the different initial values for each optimization. It is likely that different optimizations deliver different estimators.

The algorithm now compares the objective function values of each estimator. The estimator with the lowest objective function value will finally be our MSM estimator $\hat{\theta}^{n_I n_s}$.

CHAPTER 6

Results

In this chapter, we test our program using some synthetic experiments for which we know the true parameter vector θ_0 . We also test the program on a laboratory experiment and try to evaluate the MSM estimators.

For our tests, we created synthetic experiments of different sizes n_I . Note that we used method 1) in Section 5.2 to create these synthetic experiments. We apply our program using different combinations of the input variables (i.e. the number of simulations, the number of optimizations and μ_{max}). For the evaluation we take into account the true values, our MSM estimators, the absolute estimation errors in percentage and some other information that we can only obtain from synthetic experiments but not from laboratory experiments.

The variables in Table 1-4 are defined as follows:

- n_I : the total number of wells
- n_n : the total number of possible edges
- n_s : the number of simulations (chosen by the user)
- n_{opt} : the number of optimizations (chosen by the user)
- μ_{max} : the upper limit for the initial value of the contamination rate μ before optimization (chosen by the user)
- $\theta_0 = (\lambda_0^1, \lambda_0^1, \lambda_0^1, \mu_0)^T$: the true values used to create the synthetic experiment
- $\hat{\theta}_{M1}^{n_I n_s} = (\hat{\lambda}_{M1}^1, \hat{\lambda}_{M1}^2, \hat{\lambda}_{M1}^3, \hat{\mu}_{M1})^T$: the MSM estimator using method 1) in Section 5.2
- d_{M1} : the absolute deviations of $\hat{\theta}_{M1}^{n_I n_s}$ from θ_0 in percentage:

$$d_{M1} = 100 \left| 1 - \frac{\hat{\theta}_{M1}^{n_I n_s}}{\theta_0} \right|$$
- $\hat{\theta}_{M2}^{n_I n_s} = (\hat{\lambda}_{M2}^1, \hat{\lambda}_{M2}^2, \hat{\lambda}_{M2}^3, \hat{\mu}_{M2})^T$: the MSM estimators using method 2) in Section 5.2
- d_{M2} : the absolute deviations of $\hat{\theta}_{M1}^{n_I n_s}$ from θ_0 in percentage:

$$d_{M2} = 100 \left| 1 - \frac{\hat{\theta}_{M2}^{n_I n_s}}{\theta_0} \right|$$
- $err_{\hat{\theta}_{M2}^{n_I n_s}}$: the value of the objective function for $\hat{\theta}_{M2}^{n_I n_s}$.

$n_I = 25 \times 25 = 625, n_n = 1776$								
n_s	n_{opt}	μ_{max}	θ_0	$\hat{\theta}_{M1}^{n_I n_s}$	d_{M1}	$\hat{\theta}_{M2}^{n_I n_s}$	d_{M2}	$err_{\hat{\theta}_{M2}^{n_I n_s}}$
10	10	0.1	0.1	0.1287	28.7%	0.1223	22.3%	0.0126
			0.05	0.0597	19.4%	0.0605	21%	
			0.07	0.0614	12.29%	0.0587	16.14%	
			0.06	0.0428	28.67%	0.0436	27.33%	
50	10	0.1	0.1	0.1248	24.8%	0.1229	22.9%	0.0107
			0.05	0.0627	25.4%	0.0626	25.2%	
			0.07	0.0595	15%	0.0602	14%	
			0.06	0.0420	30%	0.0403	32.83%	
100	10	0.1	0.1	0.1339	33.9%	0.1271	27.1%	0.0062
			0.05	0.0629	25.8%	0.0592	18.4%	
			0.07	0.0607	13.29%	0.0606	13.43%	
			0.06	0.0396	34%	0.0413	31.17%	

Table 1

Three tests on a synthetic experiment with $n_I = 25 \times 25 = 625$ wells and $\theta_0 = (0.1, 0.05, 0.07, 0.06)$.

$n_I = 100 \times 100 = 10000, n_n = 29601$								
n_s	n_{opt}	μ_{max}	θ_0	$\hat{\theta}_{M1}^{n_I n_s}$	d_{M1}	$\hat{\theta}_{M2}^{n_I n_s}$	d_{M2}	$err_{\hat{\theta}_{M2}^{n_I n_s}}$
20	10	0.05	0.07	0.0734	4.86%	0.0728	4%	0.00020
			0.05	0.0508	1.6%	0.0497	0.6%	
			0.04	0.0390	2.5%	0.0393	1.75%	
			0.03	0.0259	13.67%	0.0258	14%	
40	10	0.05	0.07	0.0736	5.14%	0.0738	5.43%	0.00014
			0.05	0.0504	0.8%	0.0497	0.6%	
			0.04	0.0392	2%	0.0391	2.25%	
			0.03	0.0254	15.33%	0.0256	14.67%	

Table 2

Two tests on a synthetic experiment with $n_I = 100 \times 100 = 10000$ wells and $\theta_0 = (0.07, 0.05, 0.04, 0.03)$.

$n_I = 300 \times 300 = 90000, n_n = 268801$								
n_s	n_{opt}	μ_{max}	θ_0	$\hat{\theta}_{M1}^{n_I n_s}$	d_{M1}	$\hat{\theta}_{M2}^{n_I n_s}$	d_{M2}	$err_{\hat{\theta}_{M2}^{n_I n_s}}$
1	6	0.03	0.05	0.0480	4%	0.0460	8%	0.0056
			0.06	0.0575	4.17%	0.0573	4.5%	
			0.03	0.0298	0.67%	0.0311	3.67%	
			0.02	0.0239	19.5%	0.0225	12.5%	
6	3	0.03	0.05	0.0480	4%	0.0479	4.2%	0.0009
			0.06	0.0580	3.33%	0.0586	2.33%	
			0.03	0.0299	0.33%	0.0301	0.33%	
			0.02	0.0244	22%	0.0223	11.5%	

Table 3

Two tests on a synthetic experiment with $n_I = 300 \times 300 = 90000$ wells and $\theta_0 = (0.05, 0.06, 0.03, 0.02)$.

$n_I = 500 \times 500 = 250000, n_n = 748001$								
n_s	n_{opt}	μ_{max}	θ_0	$\hat{\theta}_{M1}^{n_I n_s}$	d_{M1}	$\hat{\theta}_{M2}^{n_I n_s}$	d_{M2}	$err_{\hat{\theta}_{M2}^{n_I n_s}}$
2	3	0.04	0.03	0.0295	1.67%	0.0293	2.33%	0.0011
			0.04	0.0402	0.5%	0.0401	0.25%	
			0.05	0.0522	4.4%	0.0520	4%	
			0.02	0.0192	4%	0.0195	2.5%	

Table 4

Test on a synthetic experiment with $n_I = 500 \times 500 = 250000$ wells and $\theta_0 = (0.03, 0.04, 0.05, 0.02)$.

Let us first consider the columns $\hat{\theta}_{M1}^{n_I n_s}$, d_{M1} , $\hat{\theta}_{M2}^{n_I n_s}$, and d_{M2} in Tables 1-4. We can observe that the MSM estimators of the two methods are mostly close together. However, it seems like (especially for large n_I) method 2) delivers the better estimators after all. The largest difference is found in the second part of Table 3, where $(n_s, n_{opt}, \mu_{max}) = (6, 3, 0.03)$. The deviation to the true value μ_0 is 22% for method 1) whilst we have only 11.5% for method 2).

For sample size $n_I = 250000$ in table 4, we get similar results. The deviation to μ_0 in method 1) is 4%. Method 2) only has a deviation of 2.5%. The information in table 4 is especially important for us since $n_I = 250000$ is our largest sample size.

Due to those observations, we will consider method 2) in the remaining of this chapter.

If our estimator $\hat{\mu}$ is much higher than the true value, we expect the estimators $(\hat{\lambda}^\ell)_{\ell \in \{1,2,3\}}$ to be lower than their true counterparts and vice versa (e.g. if we have a higher contamination rate, we need less seeding to get an amount of colored wells that is similar to the experiment).

In all tests of table 1, we can observe that the estimated $\hat{\mu}_{M2}$ are much smaller than the true value (at least 27.33%). At the same time, we see that the different $\hat{\lambda}_{M2}^3$ are also much smaller (up to 16.14%) than the true λ_0^3 . Intuitively, we would not expect estimators with such clearly wrong properties. However, an explanation for this incident could be the following:

In table 1 we have $n_I = 625$ and $\theta_0 = (0.1, 0.05, 0.07, 0.06)^T$. Given those values, the expected number of seeded wells is

$$\begin{aligned} & \bullet \sum_{i=1}^{625} E_{\theta_0}[\mathcal{X}_i^1] = 625\lambda_0^1 = 62.5 \text{ for DNA1,} \\ & \bullet \sum_{i=1}^{625} E_{\theta_0}[\mathcal{X}_i^2] = 625\lambda_0^2 = 31.25 \text{ for DNA2,} \\ & \bullet \sum_{i=1}^{625} E_{\theta_0}[\mathcal{X}_i^3] = 625\lambda_0^3 = 43.75 \text{ for DNA3.} \end{aligned}$$

However, our synthetic experiment with $n_I = 625$ delivered slightly different values:

$$\begin{aligned} & \bullet \sum_{i=1}^{625} \mathcal{X}_i^1 = 67 \text{ for DNA1,} \\ & \bullet \sum_{i=1}^{625} \mathcal{X}_i^2 = 33 \text{ for DNA2,} \\ & \bullet \sum_{i=1}^{625} \mathcal{X}_i^3 = 40 \text{ for DNA3,} \end{aligned}$$

As we see, we have more seeded wells than expected for DNA1 and DNA2 and less for DNA3. This can lead to wrong estimations (especially for small n_I) in the sense that the estimator for λ_0^3 is likely to be too low.

Looking at Tables 1-3, we can observe that, for fixed n_I , the value $err_{\hat{\theta}_{M2}^{n_I n_s}}$ of the objective function decreases as the number of simulations n_s increases (even for a decreasing number of optimizations μ_{opt} in Table 3), which suggests that our program works the way it should and there is no sign for bugs in our code. Nevertheless, we can see that our MSM estimators for e.g. μ_0 in Tables 1-3 still have an absolute deviation of at least 11.5%. This leads to the conjecture that the corresponding sample sizes n_I are not large enough and we do not get better estimators due to random fluctuations.

n_I	n_n	n_s	n_{opt}	μ_{max}	$err_{\hat{\theta}_{M2}^{n_I n_s}}$	err_{tri}	err_{θ_0}
625	1776	10	10	0.1	0.0126	0.6196	0.6783
625	1776	50	10	0.1	0.0107	0.5833	0.9022
625	1776	100	10	0.1	0.0062	0.5901	0.9006
10000	29601	20	10	0.05	0.0002	0.5841	0.0508
10000	29601	40	10	0.05	0.0001	0.5970	0.0461
90000	268801	1	6	0.03	0.0056	0.6317	0.0331
90000	268801	6	3	0.03	0.0009	0.6406	0.0099
250000	748001	2	3	0.04	0.0011	0.6443	0.0085

Table 5

Comparing the values of the objective function for the MSM estimator, a trivial estimator and the true value θ_0 . The synthetic experiments used are the same as in Tables 1-4.

One important observation in Tables 1-4 that supports this presumption is that the estimators in our tests get closer to the true values as n_I increases. In Table 4, the largest deviation of $\hat{\theta}_{M2}^{n_I n_s}$ from θ_0 is 2.5%, which means we found an estimator that is very close to θ_0 .

We introduce some new variables for Table 5:

- err_{tri} : the value of the objective function for the trivial estimator $\mu = 0, \lambda^\ell = \sum_{i=1}^{n_I} \sum_{s=1}^{n_s} \frac{Y_i^{\ell, s}}{n_I n_s}$ for $\ell \in \{1, 2, 3\}$, i.e. we assume there is no contamination.
- err_{θ_0} : the value of the objective function given the true value θ_0 . Note that this value has to be close to 0 to allow good estimations.

The MSM estimators $\hat{\theta}_{M2}^{n_I n_s}$ and the true values θ_0 are the same as in Tables 1-4, e.g.

$$\begin{aligned}
\theta_0 &= (0.1, 0.05, 0.07, 0.06)^T \text{ for } n_I = 625, \\
\theta_0 &= (0.07, 0.05, 0.04, 0.03)^T \text{ for } n_I = 10000, \\
\theta_0 &= (0.05, 0.06, 0.03, 0.02)^T \text{ for } n_I = 90000, \\
\theta_0 &= (0.03, 0.04, 0.05, 0.02)^T \text{ for } n_I = 250000.
\end{aligned}$$

Table 5 delivers more options to examine our estimators.

The values of the objective function for our trivial estimators (err_{tri}) are always higher than the values for the MSM estimators. This is of course

n_I	n_n	n_s	err_{θ_0}
625	1776	10	0.6783
10000	29601	10	0.0505
90000	268801	10	0.0115
250000	748001	10	0.0071
436921	1308120	10	0.0013

Table 6

Different sizes of synthetic experiments and their objective function values given the true value θ_0 . err_{θ_0} clearly decreases as n_I increases.

what we expect.

Nevertheless, we see that for $n_I = 625$, err_{tri} is smaller than err_{θ_0} , i.e. than the objective value function given θ_0 . This can be explained looking at the size of n_I . For small n_I , it is likely that our sample is not yet a good representation of the underlying distribution and other estimators can lead to a smaller objective function. In our case of $n_I = 625$, the random fluctuations are so large that even the trivial estimator has a smaller objective function value.

For our MSM estimator $\hat{\theta}_{M2}^{n_I n_s}$, a situation of *overfitting* occurs: If the sample size is not large enough, our estimation method will 'overfit' to insignificant fluctuation in the data and create an estimator with a smaller objective function value than the true θ_0 .

In our tests with $n_I = 10000$ and $n_I = 90000$, we can observe that err_{θ_0} is already a lot smaller than err_{tri} and err_{θ_0} seems to decrease with an increasing number of simulations n_s . We expect the influence of random fluctuation to decrease as n_I increases, i.e. we expect our MSM estimators to get better with increasing n_I (which is what we can observe in Tables 1-4).

In all cases of Table 5, the MSM algorithm finds estimators whose objective function values are smaller than the ones for θ_0 . As stated before, the reason for this is overfitting. However, we can observe that the larger the sample size, the better our MSM estimator. This is not surprising since the evolution of err_{θ_0} in our tests suggests: The larger the sample size, the smaller the objective function value for θ_0 .

Table 6 shows the objective function values err_{θ_0} for the true values θ_0 of the four synthetic experiments in Tables 1-5 and one extra synthetic experiment of size $n_I = 661 \cdot 661 = 436921$ (with $\theta_0 = (0.02, 0.04, 0.05, 0.03)^T$). The objective function values are all based on 10 simulations.

Again, we can clearly observe that err_{θ_0} decreases when n_I increases. This leads to the presumption that $err_{\theta_0} \rightarrow 0$ for $n_I \rightarrow \infty$ and fixed n_s . Looking at the evolution of the MSM estimators in Tables 1-4 (especially the estimator in Table 4), it also seems legitimate to assume that θ_0 is unique for $n_I \rightarrow \infty$.

With these observations, we expect our estimator to be strongly consistent (see Proposition 3.2) for $n_I \rightarrow \infty$ and fixed n_s , i.e. $\hat{\theta}_{M2}^{n_I n_s} \rightarrow \theta_0$ almost surely.

Unfortunately the program took ~ 30 hours to estimate $\hat{\theta}_{M2}^{n_I n_s}$ with the used input values in Table 4, which is why we do not state a test with larger input values for n_s and n_{opt} in this thesis (computations were carried out on a laptop computer equipped with an Intel(R) Core(TM) i5-3210M CPU @ 2.50 GHz processor and 8 GB RAM).

Note that using the above laptop, MATLAB managed to create a synthetic experiment of size $n_I = 661 \times 661$, while for $n_I = 662 \times 662$ a 'memory error' occurred and a synthetic experiment could not be created. Regardless of the time, this restricts our testing possibilities.

Testing our program on a real laboratory experiment

We now want to test our program on a laboratory experiment. We let Figure 1(b) of Section 1.2 be our experiment image. After running *automatic_grid_fitting_perspective_click.m*, the recognized grid points and colored wells are as seen in Figures 13 and 14 in Section 5.1 and the program saves a .mat file. The recognized grid consists of $n_I = 2312$ wells and we have $n_n = 6744$ possible edges. It is important to recall that the recognized grid and the colors are not perfect due to the quality of the experiment image. The difference between Figure 14 (a) and (b) influences our MSM estimator. However, this effect will be ignored in the remaining of this chapter.

We run *msm.m* with 3 different combinations of input values to get the estimators and objective function values shown in Table 7.

It is logical to assume that the best of the 3 estimators in Table 7 is the one with the smallest objective function value, i.e. we expect that $\hat{\theta}_{M2}^{n_I n_s} = (0.0186, 0.0628, 0.0443, 0.0458)^T$ is closest to the unknown θ_0 (note that it

$n_I = 2312, n_n = 6744$					
n_s	n_{opt}	μ_{max}	$\hat{\theta}_{M2}^{n_I n_s}$	$err_{\hat{\theta}_{M2}^{n_I n_s}}$	elapsed time
5	5	0.1	0.0171	0.0446	225 sec
			0.0661		
			0.0440		
			0.0463		
20	20	0.1	0.0186	0.0164	4719 sec
			0.0628		
			0.0443		
			0.0458		
80	20	0.1	0.0182	0.0193	15938 sec
			0.0638		
			0.0442		
			0.0442		

Table 7

MSM estimators for a laboratory experiment with different combinations of input values. The last column is the time that the program needed for the estimations. This is just an understanding of how much time different combinations of input variables need.

is not very surprising that $err_{\hat{\theta}_{M2}^{n_I n_s}}$ is smaller for $n_s = 20$ than for $n_s = 80$ since our sample size is relatively small).

Assuming that our estimator equals the true θ_0 and considering the expected values, we get that

- 43 wells were initially seeded with DNA1 (i.e. red color) since

$$\sum_{i=1}^{2312} E_{\hat{\theta}_{M2}^{n_I n_s}}[\mathcal{X}_i^1] = 2312 \cdot 0.0186 = 43.0,$$

- 145 wells were initially seeded with DNA2 (i.e. green color) since

$$\sum_{i=1}^{2312} E_{\hat{\theta}_{M2}^{n_I n_s}}[\mathcal{X}_i^2] = 2312 \cdot 0.0628 = 145.2,$$

- 102 wells were initially seeded with DNA3 (i.e. blue color) since

$$\sum_{i=1}^{2312} E_{\hat{\theta}_{M2}^{n_I n_s}}[\mathcal{X}_i^3] = 2312 \cdot 0.0443 = 102.4,$$

- 309 edges were open for contamination since

$$\sum_{(i,j) \in I_2} E_{\hat{\theta}_{M2}^{n_I n_s}}[\eta(\xi_{ij})] = 6744 \cdot 0.0458 = 308.9.$$

The standard deviations are

- $\sigma_{\hat{\lambda}_1} = \sqrt{2312 \cdot 0.0186(1 - 0.0186)} = 6.5,$
- $\sigma_{\hat{\lambda}_2} = \sqrt{2312 \cdot 0.0628(1 - 0.0628)} = 11.7,$
- $\sigma_{\hat{\lambda}_3} = \sqrt{2312 \cdot 0.0443(1 - 0.0443)} = 9.9,$
- $\sigma_{\hat{\mu}} = \sqrt{6744 \cdot 0.0458(1 - 0.0458)} = 17.2.$

Combining expectation and standard deviation, we get an interval $[308.9 - 17.2, 308.9 + 17.2] \approx [292, 326]$. Let us assume that the number of open edges

$$\sum_{(i,j) \in I_2} \eta_{\hat{\mu}}(\xi_{ij}) =: \mathcal{E} \text{ lies in this interval.}$$

Note that the maximum amount of wells that contain an open edge is $\leq \min\{2\mathcal{E}, n_I\}$. The maximum is reached when all open edges are isolated from each other. In fact, this is only possible for a certain number of open edges. There is a limit for which open clusters of size > 2 have to appear (but $\leq \min\{2\mathcal{E}, n_I\}$ still holds).

Taking this information and the above interval, we expect the number of contaminated wells to be $\leq \min\{2 \cdot 326, 2312\} = 652$, i.e. under the assumption that our MSM estimator equals θ_0 , we expect not more than 28.2% of the wells to be contaminated (this percentage concerns *all contamination clusters*, even those where no component contains DNA or where all components were already seeded with the same DNA, i.e. clusters, where the open edges do not have a contamination effect).

As a final approach, we create another synthetic experiment with the values that were the result of the previous estimation, i.e. we choose $n_I = 2312, n_n = 6744, \theta_0 = (0.0186, 0.0628, 0.0443, 0.0458)^T$.

We apply our program to estimate θ_0 using 200 simulations, 20 optimizations and $\mu_{max} = 0.1$. The result is shown in Table 8.

We can see that the absolute deviations of $\hat{\theta}_{M2}^{n_I n_s}$ to θ_0 take values up to 20.74% (we expect the estimator for the laboratory experiment to have similar deviations).

The MSM estimator is much better than our trivial estimator (assuming there is no contamination), which has a deviation of 100% from μ_0 . For laboratory experiments of e.g. size $n_I = 300 \times 300 = 90000$, we expect the

$n_I = 2312, n_n = 6744$					
n_s	n_{opt}	μ_{max}	θ_0	$\hat{\theta}_{M2}^{n_I n_s}$	d_{M2}
200	20	0.1	0.0186	0.0171	8.06%
			0.0628	0.0532	15.29%
			0.0443	0.0463	4.51%
			0.0458	0.0553	20.74%

Table 8

MSM estimators for a synthetic experiment that was created based on the MSM estimator and the size of a laboratory experiment.

maximum deviation to be around 12% (see Table 3) and to keep decreasing as the experiment size increases.

CHAPTER 7

Summary and outlook

We created a MATLAB program for the method of simulated moments that delivers estimators for the true DNA concentrations and the contamination rate in a digital PCR experiment for triangular grids.

For small numbers of cavities n_I , we saw that our MSM estimators have absolute deviations of up to 33% from the (unknown) true values θ_0 . Our assumption is that due to random fluctuations and overfitting, we do not find a better estimator for θ_0 for small n_I (note that our estimator is still much better than anything achievable with the naked eye or than the trivial estimator with no contamination which has a deviation of 100% from $\mu_0 \neq 0$).

However, as n_I increases, our estimators get closer to θ_0 and for $n_I = 250000$, we saw that the relative deviations were $\leq 2.5\%$.

We presume that our estimators are strongly consistent for $n_I \rightarrow \infty$ and a fixed number of simulations n_s , i.e. $\hat{\theta}^{n_I n_s} \rightarrow \theta_0$ almost surely.

The MATLAB program behaves as expected in every way which means it does not seem to have any bugs.

Our implementation of the MSM finds good estimators for $n_I \rightarrow \infty$.

Yet, one could try to improve the estimators by modifying our algorithm. The moments we choose for the MSM could be replaced by other moments that might lead to more accurate estimators for large n_I . The weight matrix Ω could also be substituted for another symmetric positive semi-definite matrix.

We have seen in Chapter 4 that there are different models to consider for the contamination process. The reason for contamination during the PCR (imperfect sealing by the glass cover) suggests that an implementation of an algorithm where the edges are represented by locally correlated random variables could also improve the estimators.

Apart from the MSM algorithm, we have seen that the algorithm for the recognition of the grid and colors does not always work properly when the experiment image shows an imperfect triangular grid or is of bad quality. Note that if the recognition of the grid works perfectly, the assignment of the colors is perfect, too. Therefore, another suggestion for improvement

is to find a better method to recognize the grid points that works even for non-perfect triangular grids.

APPENDIX A

MATLAB Code I: Recognition of the grid and spots

The following functions are shown in the order they are called in the program structure.

```
1 function automatic_grid_fitting_perspective_click(col_threshold)
2 % AUTOMATIC_GRID_FITTING_PERSPECTIVE_CLICK analyses the image chosen ...
   by the
3 % user by calling the functions:
4 % detect_centroids.m ,
5 % errorFunctionProjectiveNew.m ,
6 % compute_transformed_grid_points_perspective.m ,
7 % grid_cropping.m.
8 % These deliver the shape, size and coordinates of the grid we work with.
9 % From color_to_matrix.m it gets the variable experiment_matrix, which
10 % stores information about the area of interest (ie which grid points will
11 % be part of our calculations) and tells us whether a well is empty or
12 % filled with either R, G or B.
13 % The results are visualized and the user has to choose if the calculated
14 % grid is good enough. If so, the function saves these variables in
15 % filename.mat: experiment_matrix, shape. These can be called from
16 % msm.m to start the simulations and estimations.
17 %
18 % EXPLANATION OF VARIABLES
19 %
20 % col_threshold      Threshold for recognition of colors (only used in
21 %                   color_to_matrix)
22 % filename           Name of the data set image
23 % outputsuggestion   Default name of the output file
24 % outputname         Name of the output chosen by user
25 % I_original         Three matrices containing RGB data with the
26 %                   corresponding coordinates in the image. The first
27 %                   entry of Image(I_original) gives the value of the
28 %                   Y-axis, the second value gives the X-value.
29 %                   Therefore, we switch X and Y in Image(I_original)
30 %                   => RGB value for us = (Y,X,:)
31 % Xmax               Maximum pixel value of X-axis of image
32 % Ymax               Maximum pixel value of Y-axis of image
33 % x                  x coordinates of the two points clicked by user
34 % y                  y coordinates of the two points clicked by user
35 % r                  Approximate distance between the reference point
36 %                   and one of its neighbors. Initial value: determined
37 %                   by Euclidean distance between first and second
38 %                   clicks
39 % xg0                Reference point, X coordinate of first point that
40 %                   was clicked on
41 % yg0                Reference point, Y coordinate of first point that
```

```

42 %                                was clicked on
43 % alpha                          Rotation of image around reference point
44 % c                              Parameter of normalization
45 % centroids                      Coordinates of all RGB centroids found in the image
46 % nrows                          Number of rows
47 % ncols                          Number of columns
48 % choice                          User's choice whether the program should continue,
49 %                                restart or stop
50 % choice_2                        User's choice whether an output file should be
51 %                                created or not
52 %
53 % Felix Beck, Maja Temerinac-Ott, Bence Melykuti (University of ...
    Freiburg, Germany)
54 % 16/7/2015
55
56
57 if nargin == 0 % If no input was given, set col_threshold to default value
58     col_threshold=40;
59 end
60
61 filename=input('Enter filename: ','s');
62 I_original = imread(filename);
63
64 dots=strfind(filename, '.'); % Find the dot in the filename
65 if length(dots)>0
66     outputsuggestion=filename(1:max(dots)-1);
67 else
68     outputsuggestion=filename;
69 end
70 outputname=input(sprintf('Enter filename for output (or press Return ...
    for the default %s.mat): ','s'), 's');
71
72
73 while(1) % If user decides that the simulated grid is not good enough, ...
    the program repeats this loop until the user wishes to continue
74
75     [Ymax,Xmax] = size(I_original(:,:,1));
76
77     % Let user click on two neighbor centers
78     figure;imshow(uint8(I_original));title('Click the centers of two ...
        neighboring spots (zoom in if necessary). Press Return when ...
        finished.')
79
80     % Use the function ginputc to click the centers of two neighbors.
81     % ginputc behaves similarly to ginput, except you can customize ...
        the cursor
82     % color, line width, and line style.
83     [x, y] = ginputc('Color','w'); % Click centers of two neighbors to ...
        get (approximately) the distance between two spot centers
84     close(gcf)
85
86     % We use 8 parameters to optimize the grid
87     r = norm([x(1),y(1)]-[x(2),y(2)]);
88     r=[r r];
89
90     % Reference point (origin)
91     % Interval: must be in the image

```



```

92 % Initial value: determined by first click
93 xg0 = x(1);
94 yg0 = y(1);
95
96 lbxg = max(0.5, xg0 - 0.5 * r(1)); % Lower bound xg
97 ubxg = min(Xmax + 0.5, xg0 + 0.5 * r(1)); % Upper bound xg
98
99 lbyg = max(0.5, yg0 - 0.5 * r(2)); % Lower bound yg
100 ubyg = min(Ymax + 0.5, yg0 + 0.5 * r(2)); % Upper bound yg
101
102 % Rotation of image around reference point
103 % Interval: (-pi/2, pi/2]
104 % Initial value: gradient of the first two clicks
105 if x(2) - x(1) ~= 0
106     alpha = atan((y(2) - y(1)) / (x(2) - x(1)));
107 else
108     alpha = pi / 2;
109 end
110
111 alpha = [alpha alpha]; % We allow different rotation for the two ...
    canonical unit vectors (1,0), (0,1) but initialize them as equal
112
113 % Normalization
114 % Interval: (-Inf, Inf)
115 % Initial value: [0 0]
116 c = [1e-6 1e-6];
117
118 % Setting of the eight parameters:
119 par = [r, alpha, c, xg0, yg0]; % The optimization must run on all these ...
    8 parameters
120 lb = [0.5 * r(1), 0.5 * r(2), -pi/2, -pi/2, -inf, -inf, lbxg, lbyg]; % Lower bound
121 ub = [2 * r(1), 2 * r(2), pi/2, pi/2, inf, inf, ubxg, ubyg]; % Upper bound
122
123 % Detect spots for all colors
124 centroidsRed = detect_centroids(I_original, 1, 'red, yellow, ...
    magenta and white', r(1));
125 centroidsGreen = detect_centroids(I_original, 2, 'green, yellow, ...
    cyan and white', r(1));
126 centroidsBlue = detect_centroids(I_original, 3, 'blue, magenta, ...
    cyan and white', r(1));
127
128 centroids = [centroidsRed; centroidsGreen; centroidsBlue]; % All ...
    coordinates of all RGB centroids
129
130 % Set options for lsqnonlin optimization
131 % options = optimset('TolFun', 1e-14, 'TolX', 1e-18, 'MaxFunEvals', ...
    3000, 'MaxIter', 3000, 'Algorithm', 'levenberg-marquardt');
132 options = optimset('TolFun', 1e-14, 'TolX', 1e-18);
133
134 % Parameter optimization
135 % tic % For testing
136 [params_automat, resnorm] = lsqnonlin(@(params) ...
    errorFunctionProjectiveNew(centroids, params, Ymax, Xmax), ...
    par, lb, ub, options);
137 % toc
138 % disp(sprintf('Residual norm: %f', resnorm)) % For testing
139

```

```

140 % Calculate transformed_grid_points with new, optimized parameters
141 [~,transformed_grid_matrix] = ...
    compute_transformed_grid_points_perspective(params_automatic, ...
    Ymax, Xmax);
142
143 % Crop the grid_matrix so that only the entries that lie inside ...
    the limits
144 % of the image remain
145 [transformed_grid_matrix,shape] = grid_cropping ...
    (Xmax,Ymax,transformed_grid_matrix,centroids);
146
147 % Create four matrices: First matrix for area of interest (ie if ...
    the grid
148 % point lies within the image boundaries, corresponding area of ...
    interest
149 % entry=1 and the grid point will be part of our calculations). ...
    Remaining
150 % three matrices represent the colors of the data set: Three ...
    matrices with
151 % ones where there is a R/G/B spot at the coordinates of the grid ...
    and zeros
152 % otherwise.
153 experiment_matrix=color_to_matrix(transformed_grid_matrix, ...
    I_original, col_threshold);
154
155
156 % Plot original image, the created grid and the spotted centroids
157 figure
158 image(uint8(I_original))
159 hold on
160 title('Calculated grid after automatic fitting')
161 plot(centroids(:,1), centroids(:,2), 'k*')
162 plot(transformed_grid_matrix(:, :, 2), ...
    transformed_grid_matrix(:, :, 3), 'w.')
163 hold off
164
165
166
167 nrows=size(experiment_matrix,1);
168 ncols=size(experiment_matrix,2);
169
170 hexa(nrows,ncols,experiment_matrix,I_original,shape,1); % Plot ...
    original image and recognized grid with colors to check if fit ...
    is good enough for user's needs
171 choice = menu({'Please compare the original image to the extracted ...
    data and decide if they match.', 'How would you like to ...
    proceed?'}, 'Save', 'Try again', 'Exit');
172 close all; % Close all figures
173
174 if choice==1 || choice==3 % Continue with program (i.e. exit ...
    while(1) loop)
175     break
176 end
177 % If choice==2, start grid fitting again
178 end
179
180 if choice==3 % If user decided to stop the proogram, continue here

```

```
181     disp('The program was terminated by the user.')
182 else % If choice==1
183     if length(outputname)==0 % If user decides to use the default name ...
        outputsuggestion
184     choice_2 = menu(['Are you sure you want to save the file ', ...
        outputsuggestion, '.mat?'], 'Yes','No'); % Check if the ...
        user would like to save the file
185     if choice_2==1
186         save (sprintf('%s.mat',outputsuggestion), ...
        'experiment_matrix', 'shape')
187         disp(['The file ', outputsuggestion, '.mat was created ...
        successfully.'])
188     else % User decides not to save the file
189         disp('No file was created.')
190     end
191 else % outputname chosen by the user
192     choice_2 = menu(['Are you sure you want to save the file ', ...
        outputname, '?'], 'Yes','No'); % Check if the user would ...
        like to save the .mat file
193     if choice_2==1
194         save (sprintf('%s',outputname), 'experiment_matrix', 'shape')
195         disp(['The file ', outputname, ' was created successfully.'])
196     else % User decides not to save the file
197         disp('No file was created.')
198     end
199 end
200 end
201
202 end
```

```

1  function centroids = detect_centroids(I_original, j, ColorName, r)
2  % DETECT_CENTROIDS detects the coordinates of the centers of the spots that
3  % are R,G or B. The coordinates are saved in centroids and visualized. The
4  % user can add centroids that were not detected to improve the fitting of
5  % the grid. The adding of new centroids is done by calling the function
6  % ginputc. This function behaves similarly to ginput, except you can
7  % customize the cursor color, line width, and line style.
8  %
9  %
10 % EXPLANATION OF VARIABLES
11 %
12 % level          Global threshold (level) that can be used to
13 %                convert an intensity image to a binary image with ...
14 %                im2bw.
15 %                level is a normalized intensity value that lies in the
16 %                range [0, 1].
17 % j              Index of the color the function runs for
18 % BW             Binary image of I_Original(:, :, j)
19 % r              Approximate distance between the centers of two
20 %                neighbors
21 % P              Half of the surface area of a hexagon (with all sides
22 %                equal) with radius 1/2*r
23 % OriginalBW     Binary image where all connected components
24 %                (objects) that have fewer than P pixels are removed
25 % se             Structuring element, SE, of the type disk (a
26 %                flat, disk-shaped structuring element, where 1
27 %                specifies the radius).
28 % erodeBW        Eroded version of the image originalBW
29 % closeBW        Morphological closing on the grayscale or binary image
30 %                erodeBW
31 % dilateBW       Dilation of closeBW
32 % s              Centroids of each ellipse
33 % s2             Shortest axis of each ellipse
34 % s3             Longest axis of each ellipse
35 % s4            Scalar that specifies the angle between the
36 %                x-axis and the major axis of the ellipse
37 % MajorAxisLength (.,1)-vector MajorAxisLength that includes every length
38 % median_diameter Median of MajorAxisLength
39 % centroids      Vector containing all the centroids detected
40 % ColorName      Name of current colors
41 % I_original     Three matrices containing RGB data with the
42 %                corresponding coordinates in the image.
43 % centroids      Coordinates of all RGB centroids found in the image
44 %
45 % Felix Beck, Maja Temerinac-Ott, Bence Melykuti (University of ...
46 %                Freiburg, Germany)
47 % 16/7/2015
48
49
50
51 % Output of this function: centroids(:,1:2) => (x,y)-coordinates of the
52 % centroids
53
54 level = graythresh(I_original(:, :, j));
55

```

```

56 BW = im2bw(I_original(:, :, j), level);
57
58 % Now we calculate the area A of a hexagon with outer radius 1/2*r and ...
    remove
59 % all the connected components from a binary image that have fewer pixels
60 % than 1/2*A. In practice this means removing all signals which are not
61 % strong enough.
62 P = ceil(1/2*3/2*(1/2*r)^2*sqrt(3));
63 originalBW = bwareaopen(BW, P);
64
65 se = strel('disk', 1);
66
67 erodeBW = imerode(originalBW, se);
68 closeBW = imclose(erodeBW, se);
69 dilateBW = imdilate(closeBW, se);
70
71 BW = dilateBW;
72
73 % Calculate centers
74 s = regionprops(BW, 'centroid');
75 s2 = regionprops(BW, 'MinorAxisLength');
76 s3 = regionprops(BW, 'MajorAxisLength');
77 s4 = regionprops(BW, 'Orientation');
78
79
80 MajorAxisLength = cat(1, s3.MajorAxisLength); % Generate (.,1)-vector ...
    MajorAxisLength that includes every length
81
82 median_diameter = median(MajorAxisLength);
83 for i = 1:size(s, 1) % For each ellipse
84     if(abs(MajorAxisLength(i) - median_diameter) > median_diameter/3) % If ...
        ellipse is too big (e.g. when two neighbors are connected), ignore
85         s(i).Centroid(:) = [];
86         s2(i).MinorAxisLength = [];
87         s3(i).MajorAxisLength = [];
88         s4(i).Orientation = [];
89     end
90 end
91
92 centroids = cat(1, s.Centroid);
93
94 % Show centroids that were detected
95 str1 = ['You can add any missing ' num2str(ColorName) ' spots by marking ...
    their centers. Press Return when done.'];
96 figure; imshow(uint8(I_original)); title(str1)
97 hold on
98 if size(centroids, 1) > 0 % If any centroids in the current color were ...
    detected
99     plot(centroids(:, 1), centroids(:, 2), 'k*')
100     plot(centroids(:, 1), centroids(:, 2), 'wo')
101 end
102 hold off
103
104 % Add missing centroids
105 [t, u] = ginputc('Color', 'w'); % Gathering an unlimited number of ...
    points until you press the Return key.

```

```
106 centroids = [centroids;[t,u]]; % Add selected centroids to existing ...  
    centroids  
107 close(gcf) % Close image  
108  
109 end
```

```

1  function dist = errorFunctionProjectiveNew(centroids, params, Ymax, Xmax)
2  % ERRORFUNCTIONPROJECTIVENEW.m uses the function kdtree to create a kd-tree
3  % and finds the closest grid points for all centroids found in the ...
    image by
4  % calling the functions kdtree and kdtree_closestpoint. The distance
5  % between these centroids and the grid points is to be minimized.
6  %
7  %
8  % EXPLANATION OF VARIABLES
9  %
10 % params          8 parameters that need to be optimized
11 % Xmax            Maximum pixel value of X-axis of image
12 % Ymax            Maximum pixel value of Y-axis of image
13 % tree            kd-tree based on grid points
14 % closest_points  Coordinates of the closest grid point for each ...
    centroid
15 % centroids        Coordinates of all RGB centroids found in the image
16 % dist            Distances between grid points and centroids. dist is
17 %                 the objective function for the optimization
18 %
19 %
20 % Felix Beck, Maja Temerinac-Ott, Bence Melykuti (University of ...
    Freiburg, Germany)
21 % 16/7/2015
22
23
24 transformed_grid_points = ...
    compute_transformed_grid_points_perspective(params, Ymax, Xmax);
25
26 % Find the closest point to each grid point
27 tree = kdtree(transformed_grid_points);
28 % For each centroid, find the closest grid point and save the ...
    coordinates of
29 % that point:
30 closest_points=transformed_grid_points (kdtree_closestpoint (tree, ...
    centroids), :);
31 % Minimize distance between coordinates of grid and centroids
32 % to get the best grid possible
33 dist =[closest_points(:,1)-centroids(:,1) ; ...
    closest_points(:,2)-centroids(:,2)];
34 % Note that the output is a vector specifically for the lsqnonlin ...
    optimizer.
35 % Optionally, to use another optimizer, individually square the
36 % coordinates and sum:
37 % dist = sum((closest_points(:,1)-centroids(:,1)).^2 + ...
    (closest_points(:,2)-centroids(:,2)).^2);
38
39 end

```

```

1 function [transformed_grid_points,transformed_grid_matrix] = ...
    compute_transformed_grid_points_perspective(par, Ymax, Xmax)
2 % COMPUTE_TRANSFORMED_GRID_POINTS_PERSPECTIVE.m creates the centers of
3 % the spots in the grid. By applicating the optimized 8 parameters, the
4 % grid points are adjusted.
5 %
6 %
7 % EXPLANATION OF VARIABLES
8 %
9 % par                8 parameters that need to be optimized
10 % [r,alpha,c,xg,yg]  Splitting up of par
11 % L                  Number of dots (given r) on the diagonal ...
    of the
12 %                  image
13 % Xmax               Maximum pixel value of X-axis of image
14 % Ymax               Maximum pixel value of Y-axis of image
15 % grid_rows          Number of rows of meshgrid (compare output of
16 %                  [X,Y]=meshgrid(...))
17 % grid_cols          Number of columns
18 % grid_points        Grid points after shifting
19 % transformed_grid_points  Grid points after applying the parameters for
20 %                  transformation
21 % transformed_grid_matrix  Matrix of size (grid_rows,grid_cols,3). (:,:,1)
22 %                  is for area of interest (initially 1),
23 %                  (:,:,2:3) are x and y values of the grid
24 %
25 % Information about output:
26 % transformed_grid_points is used in errorFunctionProjectiveNew
27 % transformed_grid_matrix is used in ...
    automatic_grid_fitting_perspective_click
28 %
29 % Felix Beck, Maja Temerinac-Ott, Bence Melykuti (University of ...
    Freiburg, Germany)
30 % 16/7/2015
31
32
33 % Splitting up the parameters into different variables
34 r=par(1:2);
35 alpha=par(3:4);
36 c=par(5:6);
37 xg=par(7);
38 yg=par(8);
39
40 % Define the grid; its point are originally spaced by one unit
41 extra=2; % Depth of extra dots around the parameter we probably do not need
42 L = sqrt(Ymax^2+Xmax^2)/r(1); % Maximal length (diagonal divided by ...
    distance => number of dots on the diagonal)
43 [X, Y] = meshgrid((-ceil(L)-extra):(ceil(L)+extra), ...
    (-ceil(2/sqrt(3)*L)-extra):(ceil(2/sqrt(3)*L)+extra)); % 2/sqrt(3) ...
    etc is chosen because we visualize with hexagons
44 Y=sqrt(3)/2*Y; % Adjusting to y axis distance in hexagon grid with all ...
    sides equal
45
46 % Size of meshgrid:
47 grid_rows=size(X,1);
48 grid_cols=size(X,2);
49

```



```

50 % Moving every even/odd row to get a hexagon shape. Making sure the origin
51 % stays part of the grid and original shape==0
52 if (mod(size(X,1),4)==1) % Origin is (odd,xxx), even rows must move
53     X(2:2:end,:) = X(2:2:end,:) - 0.5;
54 else % ie. mod(size(X,1),4)==3, origin is (even,xxx), odd rows must move
55     X(1:2:end,:) = X(1:2:end,:) + 0.5;
56 end
57
58
59 grid_points = [X(:),Y(:)]; % Shifted grid points
60
61 % Rotation:
62 a=zeros(3,1);b=zeros(3,1);
63 a(1:2)=[r(1)*cos(alpha(1)) - r(2)*sin(alpha(2))];
64 b(1:2)=[r(1)*sin(alpha(1)) r(2)*cos(alpha(2))];
65 a(3)=xg;b(3)=yg;
66
67 transformed_grid_points=linfrac_translated(grid_points,a,b,c);
68
69 transformed_grid_matrix(:, :, 1)=ones(grid_rows,grid_cols);
70 transformed_grid_matrix(:, :, 2)=reshape( transformed_grid_points(:,1) , ...
    [grid_rows,grid_cols]);
71 transformed_grid_matrix(:, :, 3)=reshape( transformed_grid_points(:,2) , ...
    [grid_rows,grid_cols]);
72
73 end
74
75 function X_tr = linfrac_translated(X,a,b,c) % Calculate transformed ...
    grid points with rotation and parameters
76 X_tr=[(a(1)*X(:,1)+a(2)*X(:,2)) ./ (c(1)*X(:,1)+c(2)*X(:,2)+1)+a(3) , ...
    (b(1)*X(:,1)+b(2)*X(:,2)) ./ (c(1)*X(:,1)+c(2)*X(:,2)+1)+b(3)];
77 end

```

```

1  function [transformed_grid_matrix, shape] = grid_cropping ...
    (Xmax, Ymax, transformed_grid_matrix, centroids)
2  % GRID_CROPPING deletes rows and columns of transformed_grid_matrix which
3  % are completely outside of the image. Grid points outside the image might
4  % still remain, but their indicators (first coordinate in
5  % transformed_grid_matrix) will be set to zero in color_to_matrix.
6  % transformed_grid_matrix is overwritten by the cropped version which ...
    is an
7  % output of this function. The shape of the created grid is saved in shape.
8  %
9  %
10 % EXPLANATION OF VARIABLES
11 %
12 % Xmax                Maximum pixel value of X-axis of image
13 % Ymax                Maximum pixel value of Y-axis of image
14 % transformed_grid_matrix  Matrix of size (grid_rows, grid_cols, 3). (:, :, 1)
15 %                        is for area of interest (initially 1),
16 %                        (:, :, 2:3) are x and y values of the grid
17 % firstrow            Number of the first row after cropping the
18 %                        grid.
19 % del_col             Columns that have to be deleted from
20 %                        transformed_grid_matrix
21 % del_row             Rows that have to be deleted from
22 %                        Transformed_grid_matrix
23 % shape               Shape of the grid (determined by checking if
24 %                        firstrow is odd or even.) for more information
25 %                        see msm.m.
26 %
27 % Felix Beck, Maja Temerinac-Ott, Bence Melykuti (University of ...
    Freiburg, Germany)
28 % 16/7/2015
29
30
31 for i=1:size(transformed_grid_matrix,1)          % For all rows
32     temp=transformed_grid_matrix(i, :, 3);
33
34     % y-values of row i that are outside I_original
35     check=find(temp>=0.5 & temp<Ymax+0.5); % Find all y-values of row ...
        i that are inside I_original
36     if length(check)==0
37         transformed_grid_matrix(i, :, 1)=0; % If all entries are outside ...
            I_original, grid_matrix row i gets value 0
38     else % If at least one y-value in the row lies inside I_original: ...
        If none of the corresponding x-values in the row lies inside ...
            I_original, grid_matrix row i gets value 0
39         if max(transformed_grid_matrix(i, check, 2)>=0.5 & ...
            transformed_grid_matrix(i, check, 2)<Xmax+0.5)==0
40             transformed_grid_matrix(i, :, 1)=0;
41         end
42     end
43 end
44 end
45
46 % Use analogue technique for all columns
47 for j=1:size(transformed_grid_matrix,2)
48     temp_2=transformed_grid_matrix(:, j, 2);
49

```

```

50     check=find(temp_2>=0.5 & temp_2<Xmax+0.5); % Find all x-values of ...
        column j that are inside I_original
51     if length(check)==0
52         transformed_grid_matrix(:,j,1)=0; % If all entries are outside ...
            I_original, grid_matrix column j gets value 0
53     else % If at least one x-value in the column lies inside ...
        I_original: If none of the corresponding y-values in the ...
        column lies inside I_original, grid_matrix column j gets value 0
54         if max(transformed_grid_matrix(check,j,3)>=0.5 & ...
            transformed_grid_matrix(check,j,3)<Ymax+0.5)==0
55             transformed_grid_matrix(:,j,1)=0;
56         end
57     end
58 end
59
60
61 % Determine, which row number of transformed_grid_matrix is the first ...
    row that is still in the image.
62 firstrow=0;
63 i=1;
64 while firstrow==0
65     if length(find(transformed_grid_matrix(i,:,1)==1))>0
66         firstrow=i;
67     end
68     i=i+1;
69 end
70
71 % Determine shape by checking if firstrow is odd or even
72 if mod(firstrow,2)==0 % even => shape ==1
73     shape=1;
74 else
75     shape=0;
76 end
77
78
79 % We get our cropped matrix by determining all the entries that are 0
80 del_col=[];
81 for j=1:size(transformed_grid_matrix,2) % Delete columns
82     if max(transformed_grid_matrix(:,j,1))==0 % if all column entries ...
        are 0
83         del_col=[del_col,j];
84     end
85 end
86 transformed_grid_matrix(:,del_col,:)=[];
87
88 del_row=[];
89 for i=1:size(transformed_grid_matrix,1) % Delete rows
90     if max(transformed_grid_matrix(i,:,1))==0 % if all row entries are 0
91         del_row=[del_row,i];
92     end
93 end
94 transformed_grid_matrix(del_row,:,:)=[];
95
96
97 %Visualizing the cropped grid
98 % figure
99 % plot(centroids(:,1),centroids(:,2),'k*')

```

```
100 % hold on
101 % title('Image boundaries with centroids and fitted grid')
102 % line([0,0],[0,Ymax],'linewidth',2,'color','k')
103 % line([0,Xmax],[Ymax,Ymax],'linewidth',2,'color','k')
104 % line([Xmax,Xmax],[Ymax,0],'linewidth',2,'color','k')
105 % line([Xmax,0],[0,0],'linewidth',2,'color','k')
106 % plot(transformed_grid_matrix(:, :, 2), transformed_grid_matrix(:, :, 3), 'b.')
107 % set(gca, 'YDir', 'reverse')
108 % set(gca, 'XAxisLocation', 'top')
109 % hold off
110
111
112 end
```

```

1 function experiment_matrix = color_to_matrix ...
    (grid_matrix,I_original,col_threshold)
2 % COLOR_TO_MATRIX reads out the color of I_original at the locations stored
3 % in grid_matrix. A matrix experiment_matrix which consists of four ...
    matrices
4 % with the size of the grid is created. The first matrix is an ...
    indicator, it
5 % represents the area of interest (ie which grid points will be part ...
    of our
6 % calculations), the remaining three matrices store the RGB ...
    information of
7 % the data set. E.g. if red color is detected at grid point (i,j),
8 % experiment_matrix(i,j,2)=1, 0 otherwise.
9 %
10 %
11 % EXPLANATION OF VARIABLES
12 %
13 % [Ymax,Xmax]           Pixel limits of original image (see
14 %                       automatic_grid_fitting_perspective_click.m)
15 % I_original            Three matrices containing RGB data with the
16 %                       corresponding coordinates in the image. The
17 %                       first entry of Image(I_original) gives the
18 %                       value of the Y-axis, the second value ...
    gives the
19 %                       X-value. Therefore, we switch X and Y in
20 %                       Image(I_original)
21 %                       => RGB value for us = (Y,X,:)
22 % experiment_matrix     Consists of four different 2-dimensional
23 %                       matrices with the column and row size of the
24 %                       data set grid. The second, third and fourth
25 %                       matrix represent R, G and B. Whenever a spot
26 %                       has a color, the corresponding matrix ...
    entry is
27 %                       set to 1. Whenever there is no color in the
28 %                       spot, the matrix entry remains 0. The first
29 %                       matrix indicates the 'area of interest'
30 %                       (initially all the entries are 1). E.g. it can
31 %                       happen that some parts of a column or row are
32 %                       not within the limits of our original ...
    image. If
33 %                       that happens, the corresponding area of
34 %                       interest matrix entry is set to 0. Whenever
35 %                       there is a 0 entry in the area of interest
36 %                       matrix, the corresponding R/G/B matrix entries
37 %                       will be ignored for any further calculations.
38 %                       E.g. let us assume experiment_matrix(1,5,1)=0.
39 %                       If that happens, exmeriment_matrix(1,5,2:4)
40 %                       will be ignored in any further ...
    calculations and
41 %                       the grid will consist of one grid point
42 %                       less.
43 % grid_matrix           Matrix of size (grid_rows,grid_cols,3). (:,:,1)
44 %                       is for area of interest (initially 1),
45 %                       (:,:,2:3) are x and y values of the grid.
46 % col_threshold         Threshold for recognition of colors
47 %

```

```

48 % Felix Beck, Maja Temerinac-Ott, Bence Melykuti (University of ...
    Freiburg, Germany)
49 % 16/7/2015
50
51
52
53
54 % Returns filled_matrix (size of grid_matrix) with zeros where there was
55 % no R/G/B color in the original image and ones where a color was detected
56
57 [Ymax,Xmax,~]=size(I_original);
58
59 experiment_matrix=zeros(size(grid_matrix,1),size(grid_matrix,2),4); % ...
    Prepare variable
60
61 % Check if image has a R/G/B value higher than col_threshold at the ...
    grid coordinates.
62 % If so, set the corresponding experiment_matrix=1. Also check if the grid
63 % coordinates lie within the image boundaries and adjust area of interest.
64 for i=1:size(grid_matrix,1)
65     for j=1:size(grid_matrix,2)
66         if grid_matrix(i,j,2)>=0.5 && grid_matrix(i,j,2)<Xmax+0.5 && ...
67             grid_matrix(i,j,3)>=0.5 && grid_matrix(i,j,3)<Ymax+0.5
68             experiment_matrix(i,j,1)=1; % If grid point lies within ...
                the images boundaries: area of intrest=1.
69             for col=2:4 % For RGB
70                 if I_original(round(grid_matrix(i,j,3)),round ...
                    (grid_matrix(i,j,2)),col-1)>col_threshold % ...
                    (i,j,2) and (i,j,1) need to be exchanged. Just ...
                    like in e.g. size(I_original)
71                     experiment_matrix(i,j,col)=1;
72                 end
73             end
74         end
75     end
76 end
77
78 end

```

```

1 function hexa (nrows,ncols,wells,iniwells, shape, plotmode)
2 % HEXA.m visualizes simulations. It first creates a hexagonal grid and
3 % then adds the corresponding color to each spot.
4 %
5 %
6 % EXPLANATION OF VARIABLES
7 %
8 % shr          Shrinkage parameter is in (0,1], shows how much the spots
9 %              are shrunk relative to the circumscribed spots
10 % nrows       Number of rows
11 % ncols       Number of columns
12 % shape       Shape of the grid
13 % plotmode    Mode of plotting:
14 %              0: Plot the first simulation before and after
15 %              contamination
16 %              1: Plot original image and recognized grid with colors
17 %              to check if fit is good enough for user's needs
18 % iniwells    If plotmode==0, iniwells is the same as the variable wells
19 %              before contamination.
20 %              If plotmode==1, iniwells contains three matrices containing
21 %              RGB data with the corresponding coordinates in the image
22 %              (see I_original in
23 %              automatic_grid_fitting_perspective_click.m)
24 % wells       Consists of four different 2-dimensional
25 %              matrices with the column and row size of the
26 %              data set grid. The second, third and fourth
27 %              matrix represent R, G and B. Whenever a spot
28 %              is filled with a color, the corresponding
29 %              matrix entry is set to 1. Whenever there is no
30 %              color in the spot, the matrix entry remains
31 %              0. The first matrix indicates the 'area of
32 %              interest' (initially all the entries are 1).
33 %              E.g. it can happen that some parts of a column
34 %              or row are not within the limits of our
35 %              original image. If that happens, the
36 %              corresponding area of interest matrix entry is
37 %              set to 0. Whenever there is a 0 entry in the
38 %              area of interest matrix, the corresponding
39 %              R/G/B matrix entries will be ignored for any
40 %              further calculations. E.g. let us assume
41 %              wells(1,5,1)=0. If that happens,
42 %              wells(1,5,2:4) will be ignored in any further calculations
43 %              and the grid will consist of one grid point less.
44 % colors       Vector with row and col numbers of colored wells
45 % color        Vector containing the corresponding color to each ...
46 %              entry in
47 %              colors
48 %
49 % Felix Beck, Maja Temerinac-Ott, Bence Melykuti (University of ...
50 %              Freiburg, Germany)
51 % 16/7/2015
52
53
54
55 shr=0.8 ;

```

```

56
57 [X, Y] = meshgrid(-1:ncols+1,0:nrows+1); % Create well-centers
58 m = size(X,1); % == nrows+2
59 n = size(X,2); % == ncols+3
60
61 % Adjust grid according to shape
62 if shape == 0
63     if mod(m,2)==0
64         X = X + repmat([0; 0.5],[m/2,n]); % Shift x-axis
65     else
66         X = X + [repmat([0; 0.5],[floor(m/2),n]); zeros(1,n)];
67     end
68 else % shape==1
69     if mod(m,2)==0
70         X = X + repmat([0.5; 0],[m/2,n]); % Shift x-axis
71     else
72         X = X + [repmat([0.5; 0],[floor(m/2),n]); 0.5*ones(1,n)];
73     end
74 end
75
76 % Prepare plots
77 figure;
78 subplot(1,2,1); % First grid shows wells before contamination
79 if plotmode==0 % If plotmode==0, the first plot is the first ...
80     simulation before contamination
81     % [XV, YV] = voronoi(X(:),Y(:)); % voronoi can be activated to ...
82     visualize the (hexagonal) grid
83     % plot(XV,YV,'w')
84 else
85     imshow(uint8(iniwells)); title('Original image') % If plotmode==1, ...
86     the first plot is the original image
87     subplot(1,2,2); % Second grid shows recognized grid with colors
88     hold on
89     title('The location of recognized spots')
90     set(gca,'XTickLabel',[]) % Remove labels from second plot
91     set(gca,'YTickLabel',[])
92     hold off
93 end
94
95 for l=2:-1:1
96     if l==1 % Before contamination
97         wells1=iniwells;
98     else % After contamination
99         wells1=wells;
100     end
101     colors=zeros(2,0);
102     color='';
103
104     % Find matrix entries with red, green or blue color and save into ...
105     colors
106     [rrow, ...
107         rcol]=find(wells1(:,:,2).*(1-wells1(:,:,3)).*(1-wells1(:,:,4))); ...
108         % Find red entries
109     colors=[colors [rrow rcol]'];

```



```

107 color= repmat('r',[1 length(row)]);
108 [grow, ...
    gcol]=find((1-wells1(:,:,2)).*(wells1(:,:,3)).*(1-wells1(:,:,4))); ...
    % Find green entries
109 colors=[colors [grow gcol]'];
110 color=[color repmat('g',[1 length(grow)])];
111 [brow, ...
    bcol]=find((1-wells1(:,:,2)).*(1-wells1(:,:,3)).*(wells1(:,:,4))); ...
    % Find blue entries
112 colors=[colors [brow bcol]'];
113 color=[color repmat('b',[1 length(brow)])];
114
115 % Find matrix entries with two or three colors and save into colors
116 [yrow, ...
    ycol]=find(wells1(:,:,2).*(wells1(:,:,3)).*(1-wells1(:,:,4))); % ...
    Find yellow (red&green) entries
117 colors=[colors [yrow ycol]'];
118 color=[color repmat('y',[1 length(yrow)])];
119 [crow, ...
    ccol]=find((1-wells1(:,:,2)).*(wells1(:,:,3)).*(wells1(:,:,4))); % ...
    Find cyan (green&blue) entries
120 colors=[colors [crow ccol]'];
121 color=[color repmat('c',[1 length(crow)])];
122 [mrow, ...
    mcol]=find(wells1(:,:,2).*(1-wells1(:,:,3)).*(wells1(:,:,4))); % ...
    Find magenta (red&blue) entries
123 colors=[colors [mrow mcol]'];
124 color=[color repmat('m',[1 length(mrow)])];
125 [wrow, wcol]=find(wells1(:,:,2).*(wells1(:,:,3)).*(wells1(:,:,4))); % ...
    Find white (red&green&blue) entries
126 colors=[colors [wrow wcol]'];
127 color=[color repmat('w',[1 length(wrow)])];
128
129 % Prepare color variable for plotting colors into corresponding grid
130 subplot(1,2,1); % Select grid
131
132 % Presettings for subplots in the loop
133 set(gca,'color','k','XAxisLocation','top','YDir','reverse');
134 axis([0 ncols+1.5 0 nrows+1]);
135 daspect([sqrt(3),2,1]); % Determine the relative scaling of the ...
    data units along the axes
136
137 for k=1:size(colors,2) % For all the color entries
138
139     % Fill spots with corresponding color
140     i=colors(1,k); % Row number of current spot
141     j=colors(2,k); % Column number of current spot
142
143     % Add corresponding color to current spot by using the ...
    coordinates of the current spot and filling it with patch
144     if shape==0
145         if mod(i,2)==0
146             patch([j, j+shr*0.5, j+shr*0.5, j, j-shr*0.5, ...
                j-shr*0.5], [i-shr*5/8, i-shr*3/8, i+shr*3/8, ...
                i+shr*5/8, i+shr*3/8, i-shr*3/8], color(k))
147         else

```

```

148         patch([j, j+shr*0.5, j+shr*0.5, j, j-shr*0.5, ...
                j-shr*0.5] + 0.5*ones(1,6), [i-shr*5/8, i-shr*3/8, ...
                i+shr*3/8, i+shr*5/8, i+shr*3/8, i-shr*3/8], color(k))
149     end
150
151     else % shape==1
152         if mod(i,2)==0
153             patch([j, j+shr*0.5, j+shr*0.5, j, j-shr*0.5, ...
                    j-shr*0.5] + 0.5*ones(1,6), [i-shr*5/8, i-shr*3/8, ...
                    i+shr*3/8, i+shr*5/8, i+shr*3/8, i-shr*3/8], color(k))
154         else
155             patch([j, j+shr*0.5, j+shr*0.5, j, j-shr*0.5, ...
                    j-shr*0.5], [i-shr*5/8, i-shr*3/8, i+shr*3/8, ...
                    i+shr*5/8, i+shr*3/8, i-shr*3/8], color(k))
156         end
157     end
158 end
159
160
161 if plotmode==1 % If plotmode==1, the first plot stays untouched; ...
    Exit the loop
162     break
163 end
164
165 end
166 end

```

APPENDIX B

MATLAB Code II: Estimation by MSM

The following functions are shown in the order they are called in the program structure.

```
1  function [ solutions , result , experiment_matrix , wells ] = ...
    msm( simstep_max , loops , mu_max )
2
3  % MSM is the main function for the parameter estimation by the method of
4  % simulated moments. It calls the RGB matrices and shape of the data set
5  % from a file that was created by
6  % automatic_grid_fitting_perspective_click.m. The moments we need for the
7  % estimation are called from simcalcs.m. Initial estimators are ...
    chosen , the
8  % objective function for the optimization is computed in optim.m and then
9  % optimized with fminsearchbnd. The best estimator is saved in
10 % in a file.
11 %
12 %
13 %
14 % EXPLANATION OF VARIABLES
15 %
16 % simstep_max      Number of simulations
17 % loops            Number of optimizations with different initial values
18 % mu_max           Maximum initial mu value for optimizations
19 % shape            Shape of the grid
20 %
21 % shape=0: Odd numbered rows are shifted to the right by half a unit
22 % 1 2 3 ... ncols-1 ncols (shiftedrowsds)
23 % 1 2 3 ... ncols-1 ncols (originalrowsds)
24 % 1 2 3 ... ncols-1 ncols (shiftedrowsds)
25 % ...
26 %
27 % shape=1: Even numbered rows are shifted to the right by half a unit
28 % 1 2 3 ... ncols-1 ncols
29 % 1 2 3 ... ncols-1 ncols (shiftedrowsds)
30 % 1 2 3 ... ncols-1 ncols
31 % ...
32 %
33 % experiment_matrix Four matrices of size (nrows,ncols) - representing the
34 % grid we get from the data set - with entries of
35 % either 0 or 1. experiment_matrix(:, :, 1) represents the
36 % area of interest, ie if experiment_matrix(i,j,1)==1
37 % that well is going to be part of the final grid ...
    and of
38 % the calculations. If experiment_matrix(i,j,1)==0, we
39 % remove that well from all our calculations.
```

```

40 %           experiment_matrix(i,j,2:4) represent the RGB color(s)
41 %           in well (i,j). e.g. if experiment_matrix(i,j,3)==1,
42 %           there is green color in well (i,j).
43 %           For more detailed information see explanation of
44 %           experiment_matrix in
45 %           automatic_grid_fitting_perspective_click.m.
46 % nrows      Number of rows
47 % ncols      Number of columns
48 % originalrowsds Rows in experiment_matrix of the data set (ds) ...
    %         that are
49 %           'further left'. see
50 %           explanation of shape.
51 % shiftedrowsds Rows in experiment_matrix that are 'further ...
    %         right'. see
52 %           explanation of shape.
53 % Note:
54 % The main purpose is to know which rows are 'further
55 % right', which we call shiftedrowsds. They are just
56 % called originalrowsds and shiftedrowsds, it is not
57 % important to know if shiftedrowsds were actually
58 % shifted to the right or if originalrowsds were shifted
59 % to the left.
60 % max_edges Matrix of size (nrows,ncols,3) that carries information
61 %           about if there can possibly be contamination between
62 %           well (i,j) and its neighbors. The last dimension
63 %           represents the three directions of possible
64 %           contamination. Edge direction: 1=right, 2=right down,
65 %           3=left down. E.g. if there could be contamination
66 %           between well (i,j) and its right down neighbor,
67 %           max_edges(i,j,2)==1. If there cannott be contamination
68 %           (ie at least one of the two wells lies outside the area
69 %           of interest), max_edges (i,j,2)==0.
70 % wells Matrix of size (nwors,ncols,4,simstep_max). Each
71 %           simulation step has one matrix of size
72 %           (nrows,ncols,4). (:,:,1) represents the
73 %           area of interest. (:,:,2:4) represent the wells of
74 %           the data set that can be filled with R, G or B
75 %           (e.g. if well (i,j) of the first simulation is
76 %           filled with green color, wells(i,j,3,1)=1, else 0).
77 % totalwells Number of wells that could possibly have a color.
78 % indexed_wells wells(:,:,1,:) has indicator variables 0 and 1;
79 %           indexed_wells will index the locations of 1s in
80 %           increasing order, 0s will remain 0
81 % rpwells Permutation of totalwells
82 % edges simstep_max matrices of size (nrows,ncols,3) matrix
83 %           whose entries (i,j,k) are indicator variables of edges
84 %           between well (i,j) and its neighbor to the (k=1) right,
85 %           (k=2) right down, (k=3) left down.
86 % totaledgelocations Number of edges that could possibly be open.
87 % indexed_edges max_edges has indicator variables 0 and 1;
88 %           indexed_edges will index the locations of 1s in
89 %           increasing order, 0s will remain 0.
90 % rpedges Permutation of totaledgelocations.
91 % solutions Vector containing the initial estimators and the
92 %           objective function value (first 5 entries of each row),
93 %           the corresponding estimators after the optimization
94 %           (entries 6-9 of each row) and the objective function

```

```

95 % value (10th entry of each row).
96 % lambda_max max/initial lambda: Sum of all R-G-B wells divided
97 % by number total wells (there could not have been more
98 % seeds than how many times the color is observed after
99 % contamination).
100 % lambda=[x; y; z] Seeding rate for the three colors
101 % mu Contamination rate (probability of any given
102 % undirected edge being open)
103 % estimator Best estimator of all optimizations
104 % result Output of the best estimator [lambda,mu]
105 % optimizations Variable with the number of loops and all estimators
106 % and errors before and after optimizations
107 %
108 %
109 % Felix Beck, Bence Melykuti (University of Freiburg, Germany)
110 % 16/7/2015
111
112
113 % Load the RGB matrices of the data set and shape, ie the variables ...
114 % 'experiment_matrix' and 'shape'
115 filename=input('Filename for data set: ','s');
116 load(filename);
117 dots=strfind(filename, '.'); % Find the dot in the filename
118 if length(dots)>0
119     outputsuggestion=sprintf('%s_estimators',filename(1:max(dots)-1));
120 else
121     outputsuggestion=sprintf('%s_estimators',filename);
122 end
123 outputname=input(sprintf('Enter filename for output (or press Return ...
124 % for the default %s.mat): ','s'), 's');
125
126 tic
127
128 % Preset sizes of simulation matrices
129 nrow=size(experiment_matrix,1);
130 ncol=size(experiment_matrix,2);
131
132 % rng('shuffle'); % Shuffle random generator
133 rng('default'); % Reset random generator
134
135 % Create originalrowsds & shiftedrowsds
136 if shape==0;
137     originalrowsds=2:2:size(experiment_matrix,1);
138     shiftedrowsds=1:2:size(experiment_matrix,1);
139 else %shape ==1
140     originalrowsds=1:2:size(experiment_matrix,1);
141     shiftedrowsds=2:2:size(experiment_matrix,1);
142 end
143
144
145 % We create three matrices max_edges(:, :, 1:3) for possible open edges (ie
146 % possible contamination between two neighbors), needed for the ...
147 % calculations of the moments:
148 % Edge direction: 1=right, 2=right down, 3=left down. If an edge would

```

```

148 % connect a well outside and one inside the area of interest , it must ...
    be set to closed (0). We first allow every edge to be open ...
    randomly , then zero those which cannot be:
149 % (originalrowsds , 1): no left down; (shiftedrowsds, end): no right , ...
    no right down; (originalrowsds, end): no right; (end,:) : no right ...
    down, no left down
150 max_edges=ones(nrows,ncols,3);
151 max_edges(originalrowsds,1,3)=0;
152 max_edges(shiftedrowsds,end,1)=0;
153 max_edges(shiftedrowsds,end,2)=0;
154 max_edges(originalrowsds,end,1)=0;
155 max_edges(end,:[2 3])=0;
156
157 % Checking if the neighbors for each well are part of area of interest and
158 % if not , close the corresponding edge.
159 for i=1:nrows
160     for j=1:ncols
161         if experiment_matrix(i,j,1)==1 % 'Area of interest '
162             if max(i==originalrowsds)==1 % If row is part of originalrowsds
163                 if j<ncols && experiment_matrix(i,j+1,1)==0 % If right ...
                    neighbor is not in area of interest
164                     max_edges(i,j,1)=0; % right edge cannot exist
165                 end
166                 if i<nrows && experiment_matrix(i+1,j,1)==0 % If right ...
                    down neighbor is not in area of interest
167                     max_edges(i,j,2)=0; % right down edge cannot exist
168                 end
169                 if i<nrows && j>1 && experiment_matrix(i+1,j-1,1)==0 % ...
                    If down left neighbor is not in area of interest
170                     max_edges(i,j,3)=0; % left down edge cannot exist
171                 end
172             else % Analogue if row is part of shiftedrowsds
173                 if j<ncols && experiment_matrix(i,j+1,1)==0
174                     max_edges(i,j,1)=0;
175                 end
176                 if j<ncols && i<nrows && experiment_matrix(i+1,j+1,1)==0
177                     max_edges(i,j,2)=0;
178                 end
179                 if i<nrows && experiment_matrix(i+1,j,1)==0
180                     max_edges(i,j,3)=0;
181                 end
182             end
183         else
184             max_edges(i,j,:)=0; % If (i,j) is not in area of interest , ...
                all three edges are zero
185         end
186     end
187 end
188
189 % Preparing wells
190 wells=zeros(nrows,ncols,4);
191 wells(:, :, 1)=experiment_matrix(:, :, 1); % Area of interest equals the ...
    area of interest of the experiment_matrix
192 totalwells=sum(sum(wells(:, :, 1)));
193
194 indexed_wells=wells(:, :, 1);

```

```

196 counter=0;
197 for i=1:nrows*ncols
198     counter=counter+indexed_wells(i);
199     indexed_wells(i)=indexed_wells(i)*counter; % Only insert counter ...
        if not 0 (ie. 1)
200 end
201
202 rpwells=zeros(totalwells,3,simstep_max); % This is the source of ...
        randomness for wells.
203 for simstep=1:simstep_max
204     for i=1:3
205         rpwells(:,i,simstep)=randperm(totalwells);
206     end
207 end
208
209 % Preparing edges
210 edges=zeros(nrows,ncols,3,simstep_max);
211
212 totaledgelocations=sum(sum(sum(max_edges)));
213
214 indexed_edges=max_edges;
215 counter=0;
216 for i=1:nrows*ncols*3
217     counter=counter+max_edges(i);
218     indexed_edges(i)=max_edges(i)*counter; % Only insert counter if ...
        not 0 (ie. 1)
219 end
220
221 rpedges=zeros(totaledgelocations,simstep_max); % This is the source of ...
        randomness for edges.
222 for simstep=1:simstep_max
223     rpedges(:,simstep)=randperm(totaledgelocations);
224 end
225
226
227 [ dsRGB,dstwocol,dsnb ] = simcalcs(nrows,ncols,shiftedrowsds,...
        originalrowsds,experiment_matrix,max_edges); % Calculations for ...
        data set
228
229 solutions=zeros(loops,10); % Presettings for solutions
230
231 lambda_max=[sum(sum(experiment_matrix(:, :, 2))) / ...
        sum(sum(experiment_matrix(:, :, 1))),... % max/initial lambda: sum ...
        of all R-G-B wells / number total wells
232     sum(sum(experiment_matrix(:, :, 3))) / ...
        sum(sum(experiment_matrix(:, :, 1))),...
233     sum(sum(experiment_matrix(:, :, 4))) / ...
        sum(sum(experiment_matrix(:, :, 1)))]];
234
235
236 % Settings for initial estimator(s) of lambda and mu
237 for l=1:loops % For number of optimizations
238     if l==1 % If first optimization
239         lambda=lambda_max; % lambda gets the largest value possible
240         mu=0; % mu gets the smallest value possible
241     else % If loops>1, mu is going up, lambda is going ...
        down; lambda is always >0

```

```

242     lambda=lambda-lambda_max/loops; % New initial lambda ...
        estimators (assumption: lambda_max/loops<=lambda<=lambda_max)
243     mu=mu+mu_max/(loops-1); % New initial mu estimators ...
        (assumption: 0<=mu<=mu_max)
244
245     end
246
247     solutions(1,1:4)=[lambda,mu]; % Estimators before optimization
248
249     % Calculating the max error with initial values of lambda and mu ...
        to test if
250     % the optimized error is really smaller
251     [~,abst]=optim_output([lambda,mu], nrows, ncols, shape, ...
        simstep_max, dsRGB, dstwocol, dsnb, max_edges, 0, totalwells, ...
        indexed_wells, rpwells, wells, totaledgelocations, ...
        indexed_edges, rpedges, edges);
252     solutions(1,5)=sum(abst(:));
253
254     % Options for fminsearchbnd optimization (function optim returns the
255     % estimators for lambda and mu)
256     options = ...
        optimset('MaxFunEvals',20000,'MaxIter',20000,'TolFun',1e-7,'TolX',1e-7);
257     % fminsearchbnd behaves similarly to fminsearch, except you can ...
        add bound
258     % constraints.
259     [param,fval]=fminsearchbnd(@(param) optim(param, nrows, ncols, ...
        shape, simstep_max, dsRGB, dstwocol, dsnb, max_edges, ...
        totalwells, indexed_wells, rpwells, wells, totaledgelocations, ...
        indexed_edges, rpedges, edges), [lambda,mu], [0,0,0,0], ...
        [lambda_max,0.2], options);
260
261     solutions(1,6:10)=[param,fval]; % Estimators after optimization ...
        and objective function value
262
263     end % of for
264
265     [~,rowmin]=min(solutions(:,end)); % Searching for best estimator in ...
        solutions
266
267     %Preparing output
268     estimator=[solutions(rowmin,6); solutions(rowmin,7); ...
        solutions(rowmin,8); solutions(rowmin,9)];
269     result=[{'estimators'};estimator(1);estimator(2);estimator(3);estimator(4)]; ...
        % Output of the best estimator [lambda_1,lambda_2,lambda_3,mu]
270
271
272     % Returning an optimized simulation as an output because the ...
        optimization function has limited output options: (note that if ...
        simstep_max>1, the image that will be shown as an output is not
273     % representative for the calculated errors/distances because the image ...
        only shows the first simulation!!)
274     lambda=estimator(1:3)';
275     mu=estimator(4)';
276     [wells,abst]=optim_output([lambda,mu], nrows, ncols, shape, ...
        simstep_max, dsRGB, dstwocol, dsnb, max_edges, 1, totalwells, ...
        indexed_wells, rpwells, wells, totaledgelocations, indexed_edges, ...
        rpedges, edges);

```



```

277
278
279 optimizations=cell(size(solutions,1)+1,size(solutions,2)+1); % Create ...
    variable with all estimators and errors before and after optimizations
280 titles=['optim number'; 'lambda_1_ini'; 'lambda_2_ini'; 'lambda_3_ini';...
281         'mu_ini'; 'error_ini'; 'lambda_1'; 'lambda_2'; ...
    'lambda_3'; 'mu'; ...
282         'error'];
283 optimizations(1,:)=cellstr(titles);
284 for i=1:size(solutions,1)
285     optimizations(i+1,:)=num2cell([i,solutions(i,:)]);
286 end
287
288 optimizations % Output for testing reasons
289 abst          % Output for testing reasons
290 result        % Output of final estimators
291
292
293 % Create a file with the best estimator.
294 estimators=cell(4,2);
295 est_titles=['lambda_red'; 'lambda_green'; 'lambda_blue'; 'mu ...
    '];
296 estimators(:,1)=cellstr(est_titles);
297 estimators(:,2)=num2cell(estimator(1:4));
298
299 % Create output for ellapsed time
300 eltime=toc;
301 time_titles='Ellapsed time in sec';
302 time(1)=cellstr(time_titles);
303 time(2)=num2cell(eltime);
304
305
306 % Create output for trivial error and optimized error
307 error_titles=['Trivial error'; 'Optimized error'];
308 errors(:,1)=cellstr(error_titles);
309 errors(:,2)=num2cell([solutions(1,5);sum(abst(:)]);
310
311 % Create output for inputs by the user
312 input_titles=['Number of simulations'; 'Number of ...
    optimizations'; 'Max initial mu value for ...
    optimizations'];
313 input_values(:,1)=cellstr(input_titles);
314 input_values(:,2)=num2cell([simstep_max,loops,mu_max]);
315
316 if length(outputname)==0 % If user decides to use the default name ...
    outputsuggestion
317     choice = menu(['Do you want to save the results in the file ', ...
        outputsuggestion, '.mat?'], 'Yes','No'); % Check if the user ...
        would like to save the file
318
319     if choice==1
320         save (sprintf('%s.mat',outputsuggestion), ...
            'estimators','wells','time','errors','input_values','max_edges')
321         disp(['The file ', outputsuggestion, '.mat was created ...
            successfully.'])
322     else
323         disp('No file was created.')

```

```
324     end
325 else % If user chose a file name
326     choice = menu(['Do you want to save the results in the file ', ...
                    outputname, '?'], 'Yes','No'); % Check if the user would like ...
                    to save the file
327     if choice==1
328         save (sprintf('%s',outputname), ...
                'estimators','wells','time','errors','input_values','max_edges')
329         disp(['The file ', outputname, ' was created successfully.'])
330     else
331         disp('No file was created.')
332     end
333 end
334
335 end
```

```

1 function [ avRGB,avtwocol,avnb ] = ...
    simcalcs(nrows,ncols,shiftedrows,originalrows,wells,max_edges)
2 % SIMCALCS.m calculates the moments we chose for the optimization of our
3 % simulations: averages of colors, multiple colors per well and neighbors
4 % with the same color
5 %
6 %
7 % EXPLANATION OF VARIABLES
8 %
9 % simstep_max      Number of simulations
10 % wells            Matrix of size (nrows,ncols,4,simstep_max). Each
11 %                  simulation step has one matrix of size
12 %                  (nrows,ncols,4). (:,:,1) represents the
13 %                  area of interest: E.g. it can happen that some ...
    parts of a column
14 %                  or row are not within the limits of our
15 %                  original image. If that happens, the
16 %                  corresponding area of interest matrix entry is
17 %                  set to 0. Whenever there is a 0 entry in the
18 %                  area of interest matrix, the corresponding
19 %                  R/G/B matrix entries will be ignored for any
20 %                  further calculations. (:,:,2:4) represent the ...
    wells of
21 %                  the data set that can be filled with R, G or B
22 %                  (e.g. if well (i,j) of the first simulation is
23 %                  filled with green color, wells(i,j,3,1)=1, else
24 %                  0).
25 % avRGB            [R,G,B]-vector of well-average of color
26 % RGsim            Matrix with ones where there is R and G color, else 0
27 % RBsim            Matrix with ones where there is R and B color, else 0
28 % GBsim            Matrix with ones where there is G and B color, else 0
29 % avRG             Average #R&G per well (divided by total number of ...
    wells & divided by number of simulations)
30 % avRB             Average #R&B per well (divided by total number of ...
    wells & divided by number of simulations)
31 % avGB             Average #G&B per well (divided by total number of ...
    wells & divided by number of simulations)
32 % avtwocol         [RG,RB,GB]-vector of well-average of multiple colors
33 %                  in one well
34 % nrows            Number of rows
35 % ncols            Number of columns
36 % originalrows      Rows in experiment_matrix that are 'further left'. See
37 %                  explanation of shape
38 % shiftedrows       Rows in experiment_matrix that are 'further ...
    right'. See
39 %                  explanation of shape
40 % max_edges         Matrix of size (nrows,ncols,3) that decides if there
41 %                  can possibly be contamination between well (i,j) and
42 %                  its neighbors. The last dimension represents the
43 %                  three directions of possible contamination. Edge
44 %                  direction: 1=right, 2=right down, 3=left down.
45 %                  E.g. if there could be contamination between well (i,j)
46 %                  and its right down neighbor, max_edges(i,j,2)==1. If
47 %                  there cannot be contamination (ie the neighbor lies
48 %                  outside the area of interest), max_edges(i,j,2)==0.
49 % simnb_r           # same colored neighbors average for simulations -
50 %                  right arrow

```

```

51 % simnb_dr          # same colored neighbors average for simulations -
52 %                  right down arrow
53 % simnb_dl          # same colored neighbors average for simulations -
54 %                  left down arrow
55 % avnb              [R,G,B]-vector of well-average of same colored
56 %                  neighbors
57 %
58 %
59 % Felix Beck, Bence Melykuti (University of Freiburg, Germany)
60 % 16/7/2015
61
62 simstep_max=size(wells,4); % By initial definition of simstep_max
63
64
65 % Calculate number of RGB in data set by adding together all occupied
66 % wells of all simulations. Divide by total number of wells (number of ...
    ones in area of
67 % interest matrix) and by total number of simulations to get average per
68 % well.
69 avRGB=[sum(sum(sum(wells(:,:,2,:))./sum(sum(wells(:,:,1,:)))/simstep_max,...
70         sum(sum(sum(wells(:,:,3,:))./sum(sum(wells(:,:,1,:)))/simstep_max,...
71         sum(sum(sum(wells(:,:,4,:))./sum(sum(wells(:,:,1,:)))/simstep_max)];
72
73
74
75 % Calculate number of wells with (at least) two colors by adding ...
    together all
76 % wells of all simulations that are RG,GB or RB. Divide by total ...
    number of wells and by total number of simulations to get average
77 % per well.
78
79 RGsim=wells(:,:,2,:)==wells(:,:,3,:) & wells(:,:,2,:)==1;
80 RBsim=wells(:,:,2,:)==wells(:,:,4,:) & wells(:,:,2,:)==1;
81 GBsim=wells(:,:,3,:)==wells(:,:,4,:) & wells(:,:,3,:)==1;
82
83 avRG=sum(sum(sum(RGsim(:,:,1,:))./sum(sum(wells(:,:,1,:)))/simstep_max;
84 avRB=sum(sum(sum(RBsim(:,:,1,:))./sum(sum(wells(:,:,1,:)))/simstep_max;
85 avGB=sum(sum(sum(GBsim(:,:,1,:))./sum(sum(wells(:,:,1,:)))/simstep_max;
86
87 avtwoocol=[avRG,avRB,avGB];
88
89 avnb=zeros(1,3); % Preset
90
91 % Calculate number of neighbored wells with same color by adding together
92 % all the R/G/B wells of all simulations that have a neighbor with the ...
    same color. Divide by
93 % total number of wells and by total number of simulations to get average
94 % per well.
95 for k=2:4 % Check neighbors for same color for R-G-B
96
97
98     simnb_r = sum(sum(sum(wells(:,1:(ncols-1),k,:)==wells(:,...
        2:ncols,k,:) & wells(:,1:(ncols-1),k,:)==1))./sum(sum...
        (max_edges(:,:,1))));
99

```

```

100     simnb_dr = sum (sum (sum ([ wells(shiftedrows (shiftedrows~=nrows), ...
101         1:(ncols-1), k, :) == wells((shiftedrows ...
102         (shiftedrows~=nrows)+1), 2:ncols, k, :) ...
103         & wells(shiftedrows (shiftedrows~=nrows), 1:(ncols-1), k, :) ...
104         == 1; ...
105         wells(originalrows (originalrows~=nrows), 1:(ncols-1), k, :) ...
106         == wells((originalrows (originalrows~=nrows)+1), ...
107         1:(ncols-1), k, :) ...
108         & wells(originalrows (originalrows~=nrows), 1:(ncols-1), k, :) ...
109         == 1]))./sum( sum (max_edges(:, :, 2))));
110
111     simnb_dl = sum (sum (sum ([ wells(shiftedrows (shiftedrows~=nrows), ...
112         1:(ncols-1), k, :) == wells((shiftedrows ...
113         (shiftedrows~=nrows)+1), 1:(ncols-1), k, :) ...
114         & wells(shiftedrows (shiftedrows~=nrows), 1:(ncols-1), k, :) ...
115         == 1;
116         wells(originalrows (originalrows~=nrows), 2:ncols, k, :) == ...
117         wells((originalrows (originalrows~=nrows)+1), 1:(ncols-1), ...
118         k, :) ...
119         & wells(originalrows (originalrows~=nrows), 2:ncols, k, :) == ...
120         1]))./sum (sum (max_edges(:, :, 3))));
121
122     avnb(k-1)=sum([simnb_r,simnb_dr,simnb_dl])/simstep_max;
123 end
124 end

```

```

1 function [wells_output,abst] = optim_output( param, nrows, ncols, ...
    shape, simstep_max, dsRGB, dstwocol, dsnb, max_edges, visualize, ...
    totalwells, indexed_wells, rpwells, wells, totaledgelocations, ...
    indexed_edges, rpedges, edges )
2 % OPTIM_OUTPUT.m is used for returning the results of a simulation ...
    using the
3 % already optimized values of lambda and mu in addition to the errors
4 % because the function optim.m has limited output options.
5 %
6 % NOTE: The difference between optim.m and optim_output.m is the ...
    number of
7 % outputs (and the variable visualize). This is because the 'real'
8 % optim.m that delivers the objective function for the optimization can
9 % only have limited outputs.
10 %
11 %
12 % EXPLANATION OF VARIABLES
13 %
14 % param                Current values of [lamda,mu]
15 % simstep_max          Number of simulations
16 % originalrows         Rows that are 'further left'. See explanation of
17 %                      shape in msm.m.
18 % shiftedrows          Rows that are 'further right'. See explanation of
19 %                      shape in msm.m.
20 % wells                Matrix of size (nrows,ncols,4,simstep_max). Each
21 %                      simulation step has one matrix of size
22 %                      (nrows,ncols,4). (:,:,1) represents the
23 %                      area of interest. (:,:,2:4) represent the ...
    wells of
24 %                      the data set that can be filled with R, G or B
25 %                      (e.g. if well (i,j) of the first simulation is
26 %                      filled with green color, wells(i,j,3,1)=1, else
27 %                      0).
28 % totalwells           Number of wells that could possibly have a color.
29 % indexed_wells        wells(:,:,1,:) has indicator variables 0 and 1;
30 %                      indexed_wells will index the locations of 1s in
31 %                      increasing order, 0s will remain 0
32 % rpwells              Permutation of totalwells
33 % edges                simstep_max matrices of size (nrows, ncols, 3)
34 %                      matrix whose entries (i,j,k) are indicator
35 %                      variables of edges between well (i,j) and its
36 %                      neighbor to the (k=1) right, (k=2) right down,
37 %                      (k=3) left down.
38 % totaledgelocations   Number of edges that could possibly be open.
39 % indexed_edges        max_edges has indicator variables 0 and 1;
40 %                      indexed_edges will index the locations of 1s in
41 %                      increasing order, 0s will remain 0.
42 % rpedges              Permutation of totaledgelocations.
43 % nrows                Number of rows
44 % ncols                Number of columns
45 % shape                Shape of the grid
46 % visualize            Information about if simulation.m
47 %                      visualizes the simulation or not (if visualize==1
48 %                      => visualize)
49 % max_edges            Matrix of size (nrows,ncols,3) that carries
50 %                      information about if there can possibly be
51 %                      contamination between well (i,j) and its neighbors.

```

```

52 %           The last dimension represents the three directions
53 %           of possible contamination. Edge direction: 1=right ,
54 %           2=right down, 3=left down. E.g. if there could be
55 %           contamination between well (i,j) and its right down
56 %           neighbor, max_edges(i,j,2)==1. If there cannot be
57 %           contamination (ie the neighbor lies outside the
58 %           area of interest), max_edges (i,j,2)==0.
59 % d_RGB      Normalized squared distances of averages of single
60 %           colors [R,G,B] over all wells, between data set
61 %           and simulations
62 % d_twocol   Normalized squared distances of averages of
63 %           multiple colors [RG,RB,GB] in one well over all
64 %           wells, between data set and simulations (currently
65 %           decided not to be important and set to zero)
66 % d_nbRGB    Normalized squared distances of averages of same
67 %           colored [R,G,B] neighbors over all possible
68 %           edges, between data set and simulations
69 % dsRGB      [R,G,B]-vector of well-average of color for data
70 %           set
71 % dstwocol   [RG,RB,GB]-vector of well-average of multiple
72 %           colors in one well for data set
73 % dsnb       [R,G,B]-vector of well-average of same colored
74 %           neighbors for data set
75 % abst       (3,3) matrix with all the above distances
76 %
77 % outputs:
78 % wells_output Wells calculated with the optimal estimators
79 % abst         Distances calculated with the optimal estimators
80 %
81 %
82 % Felix Beck, Bence Melykuti (University of Freiburg, Germany)
83 % 16/7/2015
84
85
86 lambda=param(1:3);
87 mu=param(4);
88
89
90 % Start simulation with optimized estimators and visualize results
91 [wells, shiftedrows, originalrows]=simulation(nrows, ncols, shape, ...
        lambda, mu, visualize, totalwells, simstep_max, indexed_wells, ...
        rpwells, wells, totaledgelocations, indexed_edges, rpedges, edges);
92
93 wells_output=wells(:,:,:,1); % Save first simulation for output of ...
        simulated wells
94
95 % Get averages from simcalcs
96 [ avRGB,avtwocol,avnb ] = simcalcs( ...
        nrows,ncols,shiftedrows,originalrows,wells,max_edges ); % ...
        Calculations for simulations
97
98 % Presetting for absolute distances between data set and simulations
99 d_RGB=zeros(1,3);
100 d_twocol=zeros(1,3);
101 d_nbRGB=zeros(1,3);
102
103 %(squared) distances between data set and simulations

```

```

104 for i=1:3
105     if dsRGB(i)~=0
106         d_RGB(i)=((dsRGB(i)-avRGB(i))/dsRGB(i))^2;
107     else
108         d_RGB(i)=(dsRGB(i)-avRGB(i))^2;
109     end
110     if dstwocol(i)~=0
111         d_twocol(i)=0*((dstwocol(i)-avtwocol(i))/dstwocol(i))^2;
112     else
113         d_twocol(i)=0*(dstwocol(i)-avtwocol(i))^2;
114     end
115     if dsnb(i)~=0
116         d_nbRGB(i)=((dsn(i)-avnb(i))/dsn(i))^2;
117     else
118         d_nbRGB(i)=(dsn(i)-avnb(i))^2;
119     end
120 end
121
122 abst=[d_RGB;d_twocol;d_nbRGB]; % (3,3) matrix with all distances
123
124 end

```



```

1 function [wells, shiftedrows, originalrows] = simulation (nrows, ncols, ...
    shape, lambda, mu, visualize, totalwells, simstep_max, ...
    indexed_wells, rpwells, wells, totaledgelocations, indexed_edges, ...
    rpedges, edges)
2 % SIMULATION.m creates the wells and the contamination edges for the
3 % simulations. The wells before the contamination, as well as the
4 % wells after contamination, which are an output of contamination.m, are
5 % visualized.
6 %
7 %
8 % EXPLANATION OF VARIABLES
9 %
10 % nrows           Number of rows
11 % ncols           Number of columns
12 % shape           Shape of the grid
13 % originalrowsds  Rows in wells that are 'further left'.
14 % shiftedrowsds   Rows in wells that are 'further right'.
15 % simstep_max     Number of simulations
16 % wells           simstep_max matrices of size (nrows,ncols,4) -
17 %                 representing the grid we get from the data set - with
18 %                 entries of either 0 or 1.
19 %                 First coordinate: row
20 %                 Second coordinate: column
21 %                 Third coordinate: 1:Indicator whether well is part of
22 %                 the area of interest. Ie if
23 %                 wellsds.w(i,j,1)==1 that well is
24 %                 going to be part of the final grid
25 %                 and of the calculations. If
26 %                 wellsds.w(i,j,1)==0, we ignore that
27 %                 well for all our calculations.
28 %                 2:Indicator if red (1=well is filled
29 %                 with red seed, 0=well is not filled
30 %                 with red seed)
31 %                 3:Indicator of green
32 %                 4:Indicator of blue
33 % totalwells       Number of wells that could possibly have a color.
34 % threshwell       Number of seeds that will be put into wells
35 % indexed_wells    wells(:, :, 1, :) has indicator variables 0 and 1;
36 %                 indexed_wells will index the locations of 1s in
37 %                 increasing order, 0s will remain 0
38 % rpwells          Permutation of totalwells
39 % iniwells         The wells before contamination (after seeding). Needed
40 %                 for visualization
41 % edges            simstep_max matrices of size (nrows, ncols, 3)
42 %                 matrix whose entries (i,j,k) are indicator
43 %                 variables of edges between well (i,j) and its
44 %                 neighbor to the (k=1) right, (k=2) right down,
45 %                 (k=3) left down.
46 % totaledgelocations Number of edges that could possibly be open.
47 % threshedge       Number of edges that will be put into edges.
48 % indexed_edges    max_edges in msm.m has indicator variables 0 and 1;
49 %                 indexed_edges will index the locations of 1s in
50 %                 increasing order, 0s will remain 0.
51 % rpedges          Permutation of totaledgelocations
52 % simstep          Number of current simulation
53 % lambda=[x; y; z] Seeding rate for the three colors
54 % mu              Contamination rate (probability of any given

```

```

55 %                undirected edge being open)
56 % visualize      Information about if simulation.m visualizes
57 %                the simulation or not (if visualize==1 => visualize)
58 %
59 %
60 %
61 % Felix Beck, Bence Melykuti (University of Freiburg, Germany)
62 % 16/7/2015
63
64 % Setting originalrows & shiftedrows for zeroing out the indicator of the
65 % last well in every odd row.
66 if shape==0;
67     originalrows=2:2:nrows;
68     shiftedrows=1:2:nrows;
69 else % shape=1
70     originalrows=1:2:nrows;
71     shiftedrows=2:2:nrows;
72 end
73
74 % lambda=[0.1,0.2,0.3]; % For testing reasons
75 % mu=0.04; % For testing reasons
76
77 threshwell=round(lambda*totalwells);
78 % We use the random permutations rpwells so that the proportions of ...
79 %   seeded wells are very accurately the respective parameters.
80 % rpwells(1:threshwell(i),i,simstep); -- Gives index values; the ...
81 %   places in indexed_wells where these values are found are those ...
82 %   that must get the seeds.
83 % The entries of wells with the four coordinates
84 % [j1 j2]=find(indexed_wells==rpwells(j,i,simstep)), i+1, simstep
85 % will get a seed of colour i.
86 for simstep=1:simstep_max
87     for i=1:3
88         for j=1:threshwell(i)
89             [j1 j2]=find(indexed_wells==rpwells(j,i,simstep));
90             wells(j1,j2,i+1,simstep)=1;
91             wells(:,j1,j2,i,simstep)=wells(:,j1,j2,i,simstep); % Adjust area of ...
92             interest for all simulations
93         end
94     end
95 end
96 iniwells=wells;
97
98 threshedge=round(mu*totaledgelocations);
99
100 % We use the random permutations rpedges so that the proportion of ...
101 %   open edges is very accurately the respective parameter.
102 % rpedges(1:threshedge,simstep); -- Gives index values; the places in ...
103 %   indexed_edges where these values are found are those that will get ...
104 %   the open edges.
105 % The entries of edges with these four coordinates:
106 % ind2sub([nrows,ncols,3], find(indexed_edges==rpedges(i,simstep))), ...
107 %   simstep
108 % will get an edge. Here we cannot use find on its own as there are 3 ...
109 %   coordinates, not only 2. The way we use it gives a linear index.

```

```

103 for simstep=1:simstep_max
104     for i=1:threshedge
105         [i1 , i2 , i3]=ind2sub([nrows,ncols,3], ...
106             find(indexed_edges==rpedges(i,simstep)));
107         edges(i1,i2,i3,simstep)=1;
108     end
109     edges3d=edges(:,:,simstep); % Transform edges into 3D to give ...
110         correct input for contamination.m
111 % Find all the contaminated wells
112 if simstep==1 % For visualization of the first simulation
113     [edgelist , components1]=contamination(edges3d , nrows , ncols , ...
114         shiftedrows , originalrows);
115 else
116     [~, components1]=contamination(edges3d , nrows , ncols , ...
117         shiftedrows , originalrows);
118 end
119 wells3d=wells(:,:,simstep); % Transform wells for each ...
120     simulation into 3D
121 for i=1:size(components1,2) % Loop for all connected components of ...
122     contamination
123     for j=2:4 % Loop for all colors RGB
124         comp=components1{i}+(j-1)*nrows*ncols; % (j-1)*nrows*ncols ...
125             because of dimension of wells => 'Jumps' into next matrix
126         wells3d(comp)=max(wells3d(comp)); % Apply contamination ...
127             for connected component i and color j
128     end
129 end
130 wells(:,:,simstep)=wells3d; %Transform wells back into 4D
131 end
132 % Visualize first simulation before and after contamination (if optimal
133 % estimator already found)
134 if visualize==1
135     % Create grid with hexa function
136     hexa(nrows,ncols,wells(:,:,1),iniwells(:,:,1),shape,0);
137     for l=1:2
138         subplot(1,2,l); % Plot 1: wells before contamination; Plot 2: ...
139             wells after contamination
140     if l==1
141         title('First simulation before contamination')
142     else
143         title('First simulation after contamination')
144     end
145     hold on
146     for i=1:size(edgelist,2) % Add open edges/connected components ...
147         to figures
148         if shape==0
149             plot([edgelist(2,i) + 0.5-0.5 * (1-mod(edgelist(1,i), ...
150                 2)), edgelist(4,i) + 0.5-0.5 * ...
151                 (1-mod(edgelist(3,i), 2))], [edgelist(1,i), ...
152                 edgelist(3,i)], 'w', 'Linewidth', 1.5); % Adjust x-axis
153         else % shape==1

```

```
146         plot([edgelist(2,i) + 0.5-0.5 * (mod(edgelist(1,i), ...
            2)), edgelist(4,i) + 0.5-0.5 * (mod(edgelist(3,i), ...
            2))], [edgelist(1,i), edgelist(3,i)], 'w', ...
            'Linewidth', 1.5); % Adjust x-axis
147         end
148     end
149     hold off
150 end
151 end
152
153 end
```

```

1  function [edgelist, components1]=contamination(edges, nrows, ncols, ...
        shiftedrows, originalrows)
2  % CONTAMINATION.m creates from the matrix edges the list of edges in a
3  % 4-row matrix, edgelist and a cell array of connected components by
4  % breadth-first search.
5  %
6  %
7  % EXPLANATION OF VARIABLES
8  %
9  % edges          simstep_max matrices of size (nrows, ncols, 3) matrix
10 %                whose entries (i,j,k) are indicator variables of edges
11 %                between well (i,j) and its neighbor to the (k=1) right,
12 %                (k=2) right down, (k=3) left down.
13 % edgelist        List of the edges (connected neighbors) of size ...
        (4, no.
14 %                of edges). Each column stores an edge (v_1, v_2) as
15 %                [rowindex(v_1); colindex(v_1); rowindex(v_2); ...
        colindex(v_2)].
16 % originalrows    Rows that are 'further left'. See explanation of shape
17 %                in msm.m.
18 % shiftedrows     Rows that are 'further right'. See explanation of shape
19 %                in msm.m.
20 % edgelist1        edgelist converted to size (2, no. of edges) by
21 %                sub2ind, ie. each vertex is indexed by a linear index.
22 %                Each column stores a [linearindex(v_1); ...
        linearindex(v_2)].
23 % components1{i}   Matrix which contains in one row the elements of the
24 %                i'th connected component (linear indexing). Its purpose
25 %                is that each well within a connected component will
26 %                ultimately have the same color.
27 % edgelistq1        Queue of edges. Created as a copy of edgelist1, the
28 %                visited edges are removed from it until it is empty.
29 % vtov             Vertices to visit. A row vector of vertices which have
30 %                been discovered as neighbors of an already visited
31 %                element of the current components1{i}.
32 %
33 % Felix Beck, Bence Melykuti (University of Freiburg, Germany)
34 % 12/8/2015
35
36
37 edgelist=zeros(4,0);
38
39 % For right-arrows
40 [i, j]=find(edges(:,:,1)); % Find row and column indices of all ...
        nonzero elements of edges(:,:,1)
41
42 % 1st coordinate: row indices of 1st vertices of edges
43 % 2nd coordinate: column indices of the same vertices
44 % 3rd coordinate: row indices of 2nd vertices of edges
45 % 4th coordinate: column indices of the same vertices
46 edgelist=[edgelist [i';j';i';j'+1]];
47
48
49 % Analogue for right-down arrow
50 [i, j]=find(edges(shiftedrows(:),:,2));
51 edgelist=[edgelist [shiftedrows(i);j';shiftedrows(i)+1;j'+1]];
52

```

```

53 [i, j]=find(edges(originalrows(:),:,2));
54 edgelist=[edgelist [originalrows(i);j';originalrows(i)+1;j']];
55
56 % Analogue for left-down arrow
57 [i, j]=find(edges(shiftedrows(:),:,3));
58 edgelist=[edgelist [shiftedrows(i);j';shiftedrows(i)+1;j']];
59 [i, j]=find(edges(originalrows(:),:,3));
60 edgelist=[edgelist [originalrows(i);j';originalrows(i)+1;j'-1]];
61
62
63 % Converting the representation of wells in edgelist from (row, col) ...
    to (col-1)*nrows+row (from 2 coordinates to 1).
64 % The breadth-first search (BFS) implemented this way is faster.
65 edgelist1=[sub2ind([nrows ncols], edgelist(1,:), edgelist(2,:)); ...
    sub2ind([nrows ncols], edgelist(3,:), edgelist(4,:))];
66
67
68 % We start with the first edge in edgelistq1. Put its endpoints into both
69 % components1{i} and vtov. Remove this edge from edgelistq1.
70 %
71 % While vtov not empty: Take first element from vtov. k2: all edges from
72 % edgelistq1 which have vtov(1) in the second row. Restrict k2 to k which
73 % are those where the neighbor of vtov(1) in the first row is not in
74 % components1{i} yet. Put these neighbors into both components1{i} and
75 % vtov. Remove the k2 edges from edgelistq1 as these have been traversed.
76 % Repeat with k1: all edges from edgelistq1 which have vtov(1) in the first
77 % row. Remove vtov(1) from vtov. When vtov is empty, then the ...
    component has
78 % been fully explored. If there is still an edge left in edgelistq1, then
79 % it starts a new component.
80
81 components1=cell(1);
82 edgelistq1=edgelist1;
83 i=1;
84
85 while size(edgelistq1,2)>0
86     % Start the new component with endvertices of first edge of edgelistq1
87     components1{i}=[edgelistq1(1,1) edgelistq1(2,1)]; vtov=components1{i};
88     edgelistq1(:,1)=[];
89     while length(vtov)>0
90         k2=find(edgelistq1(2,:)==vtov(1)); % Elements in edgelistq1 ...
            where the vertex vtov(1) is the second vertex
91         k=[];
92         for l=1:length(k2) % Look through the neighbors k2 and only ...
            keep the ones that are not in components1{i}
93             if any(edgelistq1(1,k2(l))==components1{i})==0
94                 k=[k k2(l)]; % k has the indices of all neighbors of ...
                    vtov(1) that are not yet in components1{i} when ...
                    vtov(1) is in the second row of edgelistq1
95             end
96         end
97         components1{i}=[components1{i} edgelistq1(1,k)];
98         vtov=[vtov edgelistq1(1,k)];
99         edgelistq1(:,k2)=[];
100
101     % Do this also when vtov(1) is in the first row of edgelistq1

```

```

102     k1=find(edgelistq1(1,:)==vtov(1)); % Elements in edgelistq1 ...
        where the vertex vtov(1) is the first vertex
103     k=[];
104     for l=1:length(k1) % Look through the neighbors k1 and only ...
        keep the ones that are not in components1{i}
105         if any(edgelistq1(2,k1(l))==components1{i})==0
106             k=[k k1(l)]; % k has the indices of all neighbors of ...
                vtov(1) that are not yet in components1{i} when ...
                vtov(1) is in the first row of edgelistq1
107         end
108     end
109     components1{i}=[components1{i} edgelistq1(2,k)];
110     vtov=[vtov edgelistq1(2,k)];
111     edgelistq1(:,k1)=[];
112
113     vtov(1)=[];
114 end % of while length(vtov)>0
115     i=i+1;
116 end % of while size(edgelistq1,2)>0
117 end

```

```

1  function epsilon_new=optim( param, nrows, ncols, shape, simstep_max, ...
    dsRGB, dstwocol, dsnb, max_edges, totalwells, indexed_wells, ...
    rpwells, wells, totaledgelocations, indexed_edges, rpedges, edges )
2  % OPTIM.m computes the objective function for the optimization of parameter
3  % values with given values of lambda and mu. The matrices for the ...
    simulation
4  % (ie. wells) are received from simulation.m and the moments needed for
5  % the optimization from simcalcs.m.
6  %
7  % NOTE: The difference between optim.m and optim_output.m is the ...
    number of
8  % outputs (and the variable visualize). This is because the 'real'
9  % optim.m that delivers the objective function for the optimization can
10 % only have limited outputs.
11 %
12 %
13 % EXPLANATION OF VARIABLES
14 %
15 % param                Current values of [lamda,mu]
16 % simstep_max          Number of simulations
17 % originalrows         Rows that are 'further left'. See explanation of
18 %                      shape in msm.m.
19 % shiftedrows          Rows that are 'further right'. See explanation of
20 %                      shape in msm.m.
21 % wells                Matrix of size (nrows,ncols,4,simstep_max). Each
22 %                      simulation step has one matrix of size
23 %                      (nrows,ncols,4). (:,:,1) represents the
24 %                      area of interest. (:,:,2:4) represent the ...
    wells of
25 %                      the data set that can be filled with R, G or B
26 %                      (e.g. if well (i,j) of the first simulation is
27 %                      filled with green color, wells(i,j,3,1)=1, else
28 %                      0).
29 % totalwells           Number of wells that could possibly have a color.
30 % indexed_wells        wells(:,:,1,:) has indicator variables 0 and 1;
31 %                      indexed_wells will index the locations of 1s in
32 %                      increasing order, 0s will remain 0
33 % rpwells              Permutation of totalwells
34 % edges                simstep_max matrices of size (nrows, ncols, 3)
35 %                      matrix whose entries (i,j,k) are indicator
36 %                      variables of edges between well (i,j) and its
37 %                      neighbor to the (k=1) right, (k=2) right down,
38 %                      (k=3) left down.
39 % totaledgelocations   Number of edges that could possibly be open.
40 % indexed_edges        max_edges has indicator variables 0 and 1;
41 %                      indexed_edges will index the locations of 1s in
42 %                      increasing order, 0s will remain 0.
43 % rpedges              Permutation of totaledgelocations.
44 % nrows                Number of rows
45 % ncols                Number of columns
46 % shape                Shape of the grid
47 % max_edges            Matrix of size (nrows,ncols,3) that carries
48 %                      information about if there can possibly be
49 %                      contamination between well (i,j) and its neighbors.
50 %                      The last dimension represents the three directions
51 %                      of possible contamination. Edge direction: 1=right,
52 %                      2=right down, 3=left down. E.g. if there could be

```



```

53 % contamination between well (i,j) and its right down
54 % neighbor, max_edges(i,j,2)==1. If there cannot be
55 % contamination (ie the neighbor lies outside the
56 % area of interest), max_edges (i,j,2)==0.
57 % d_RGB Normalized squared distances of averages of single
58 % colors [R,G,B] over all wells, between data set
59 % and simulations
60 % d_twocol Normalized squared distances of averages of
61 % multiple colors [RG,RB,GB] in one well over all
62 % wells, between data set and simulations (currently
63 % decided not to be important and set to zero)
64 % d_nbRGB Normalized squared distances of averages of same
65 % colored [R,G,B] neighbors over all possible
66 % edges, between data set and simulations
67 % dsRGB [R,G,B]-vector of well-average of color for data
68 % set
69 % dstwocol [RG,RB,GB]-vector of well-average of multiple
70 % colors in one well for data set
71 % dsnb [R,G,B]-vector of well-average of same colored
72 % neighbors for data set
73 % abst (3,3) matrix with all the above distances
74 % epsilon_new Sum of all distances (which is to be minimized)
75 %
76 %
77 % Felix Beck, Bence Melykuti (University of Freiburg, Germany)
78 % 16/7/2015
79
80
81 lambda=param(1:3);
82 mu=param(4);
83
84
85 % Start simulation(s)
86 [wells,shiftedrows,originalrows]=simulation(nrows,ncols,shape,...
      lambda,mu,0,totalwells,simstep_max,indexed_wells,rp wells,...
      wells,totaledgelocations,indexed_edges,rp edges,edges);
87
88 % Get averages from simcalcs
89 [avRGB,avtwocol,avn b]=simcalcs(...
      nrows,ncols,shiftedrows,originalrows,wells,max_edges); % ...
      Calculations for simulations
90
91 % Presetting for absolute distances between data set and simulations
92 d_RGB=zeros(1,3);
93 d_twocol=zeros(1,3);
94 d_nbRGB=zeros(1,3);
95
96 %(squared) distances between data set and simulations
97 for i=1:3
98     if dsRGB(i)~=0
99         d_RGB(i)=((dsRGB(i)-avRGB(i))/dsRGB(i))^2;
100     else
101         d_RGB(i)=(dsRGB(i)-avRGB(i))^2;
102     end
103     if dstwocol(i)~=0
104         d_twocol(i)=0*((dstwocol(i)-avtwocol(i))/dstwocol(i))^2;
105     else

```

```
106         d_twocol(i)=0*(dstwocol(i)-avtwocol(i))^2;
107     end
108     if dsnb(i)~=0
109         d_nbRGB(i)=(dsnb(i)-avnb(i))/dsnb(i)^2;
110     else
111         d_nbRGB(i)=(dsnb(i)-avnb(i))^2;
112     end
113 end
114
115 % optimize the maximum of the calculated distances to get the best possible
116 % estimator for lambda and mu
117 abst=[d_RGB;d_twocol;d_nbRGB];
118 epsilon_new=sum(abst(:));
119
120 end
```

```

1  function hexa (nrows,ncols,wells,iniwells, shape, plotmode)
2  % HEXAm visualizes simulations. It first creates a hexagonal grid and
3  % then adds the corresponding color to each spot.
4  %
5  %
6  % EXPLANATION OF VARIABLES
7  %
8  % shr          Shrinkage parameter is in (0,1], shows how much the spots
9  %              are shrunk relative to the circumscribed spots
10 % nrows        Number of rows
11 % ncols        Number of columns
12 % shape        Shape of the grid
13 % plotmode      Mode of plotting:
14 %              0: Plot the first simulation before and after
15 %              contamination
16 %              1: Plot original image and recognized grid with colors
17 %              to check if fit is good enough for user's needs
18 % iniwells      If plotmode==0, iniwells is the same as the variable wells
19 %              before contamination.
20 %              If plotmode==1, iniwells contains three matrices containing
21 %              RGB data with the corresponding coordinates in the image
22 %              (see I_original in
23 %              automatic_grid_fitting_perspective_click.m)
24 % wells         Consists of four different 2-dimensional
25 %              matrices with the column and row size of the
26 %              data set grid. The second, third and fourth
27 %              matrix represent R, G and B. Whenever a spot
28 %              is filled with a color, the corresponding
29 %              matrix entry is set to 1. Whenever there is no
30 %              color in the spot, the matrix entry remains
31 %              0. The first matrix indicates the 'area of
32 %              interest' (initially all the entries are 1).
33 %              E.g. it can happen that some parts of a column
34 %              or row are not within the limits of our
35 %              original image. If that happens, the
36 %              corresponding area of interest matrix entry is
37 %              set to 0. Whenever there is a 0 entry in the
38 %              area of interest matrix, the corresponding
39 %              R/G/B matrix entries will be ignored for any
40 %              further calculations. E.g. let us assume
41 %              wells(1,5,1)=0. If that happens,
42 %              wells(1,5,2:4) will be ignored in any further calculations
43 %              and the grid will consist of one grid point less.
44 % colors        Vector with row and col numbers of colored wells
45 % color         Vector containing the corresponding color to each ...
46 %              entry in
47 %              colors
48 %
49 % Felix Beck, Maja Temerinac-Ott, Bence Melykuti (University of ...
50 %              Freiburg, Germany)
51 % 16/7/2015
52
53
54
55 shr=0.8 ;

```

```

56
57 [X, Y] = meshgrid(-1:ncols+1,0:nrows+1); % Create well-centers
58 m = size(X,1); % == nrows+2
59 n = size(X,2); % == ncols+3
60
61 % Adjust grid according to shape
62 if shape == 0
63     if mod(m,2)==0
64         X = X + repmat([0; 0.5],[m/2,n]); % Shift x-axis
65     else
66         X = X + [repmat([0; 0.5],[floor(m/2),n]); zeros(1,n)];
67     end
68 else % shape==1
69     if mod(m,2)==0
70         X = X + repmat([0.5; 0],[m/2,n]); % Shift x-axis
71     else
72         X = X + [repmat([0.5; 0],[floor(m/2),n]); 0.5*ones(1,n)];
73     end
74 end
75
76 % Prepare plots
77 figure;
78 subplot(1,2,1); % First grid shows wells before contamination
79 if plotmode==0 % If plotmode==0, the first plot is the first ...
80     simulation before contamination
81     % [XV, YV] = voronoi(X(:),Y(:)); % voronoi can be activated to ...
82     visualize the (hexagonal) grid
83     % plot(XV,YV,'w')
84 else
85     imshow(uint8(iniwells)); title('Original image') % If plotmode==1, ...
86     the first plot is the original image
87     subplot(1,2,2); % Second grid shows recognized grid with colors
88     hold on
89     title('The location of recognized spots')
90     set(gca,'XTickLabel',[]) % Remove labels from second plot
91     set(gca,'YTickLabel',[])
92     hold off
93 end
94
95 for l=2:-1:1
96     if l==1 % Before contamination
97         wells1=iniwells;
98     else % After contamination
99         wells1=wells;
100     end
101     colors=zeros(2,0);
102     color='';
103
104     % Find matrix entries with red, green or blue color and save into ...
105     colors
106     [rrow, ...
107         rcol]=find(wells1(:,:,2).*(1-wells1(:,:,3)).*(1-wells1(:,:,4))); ...
108     % Find red entries
109     colors=[colors [rrow rcol]'];

```

```

107 color= repmat('r',[1 length(row)]);
108 [grow, ...
    gcol]=find((1-wells1(:,:,2)).*wells1(:,:,3).*(1-wells1(:,:,4))); ...
    % Find green entries
109 colors=[colors [grow gcol]'];
110 color=[color repmat('g',[1 length(grow)])];
111 [brow, ...
    bcol]=find((1-wells1(:,:,2)).*(1-wells1(:,:,3)).*wells1(:,:,4))); ...
    % Find blue entries
112 colors=[colors [brow bcol]'];
113 color=[color repmat('b',[1 length(brow)])];
114
115 % Find matrix entries with two or three colors and save into colors
116 [yrow, ...
    ycol]=find(wells1(:,:,2).*wells1(:,:,3).*(1-wells1(:,:,4))); % ...
    Find yellow (red&green) entries
117 colors=[colors [yrow ycol]'];
118 color=[color repmat('y',[1 length(yrow)])];
119 [crow, ...
    ccol]=find((1-wells1(:,:,2)).*wells1(:,:,3).*wells1(:,:,4))); % ...
    Find cyan (green&blue) entries
120 colors=[colors [crow ccol]'];
121 color=[color repmat('c',[1 length(crow)])];
122 [mrow, ...
    mcol]=find(wells1(:,:,2).*(1-wells1(:,:,3)).*wells1(:,:,4))); % ...
    Find magenta (red&blue) entries
123 colors=[colors [mrow mcol]'];
124 color=[color repmat('m',[1 length(mrow)])];
125 [wrow, wcol]=find(wells1(:,:,2).*wells1(:,:,3).*wells1(:,:,4))); % ...
    Find white (red&green&blue) entries
126 colors=[colors [wrow wcol]'];
127 color=[color repmat('w',[1 length(wrow)])];
128
129 % Prepare color variable for plotting colors into corresponding grid
130 subplot(1,2,1); % Select grid
131
132 % Presettings for subplots in the loop
133 set(gca,'color','k','XAxisLocation','top','YDir','reverse');
134 axis([0 ncols+1.5 0 nrows+1]);
135 daspect([sqrt(3),2,1]); % Determine the relative scaling of the ...
    data units along the axes
136
137 for k=1:size(colors,2) % For all the color entries
138
139     % Fill spots with corresponding color
140     i=colors(1,k); % Row number of current spot
141     j=colors(2,k); % Column number of current spot
142
143     % Add corresponding color to current spot by using the ...
    coordinates of the current spot and filling it with patch
144     if shape==0
145         if mod(i,2)==0
146             patch([j, j+shr*0.5, j+shr*0.5, j, j-shr*0.5, ...
                j-shr*0.5], [i-shr*5/8, i-shr*3/8, i+shr*3/8, ...
                i+shr*5/8, i+shr*3/8, i-shr*3/8], color(k))
147         else

```

```

148         patch([j, j+shr*0.5, j+shr*0.5, j, j-shr*0.5, ...
                j-shr*0.5] + 0.5*ones(1,6), [i-shr*5/8, i-shr*3/8, ...
                i+shr*3/8, i+shr*5/8, i+shr*3/8, i-shr*3/8], color(k))
149     end
150
151     else % shape==1
152         if mod(i,2)==0
153             patch([j, j+shr*0.5, j+shr*0.5, j, j-shr*0.5, ...
                    j-shr*0.5] + 0.5*ones(1,6), [i-shr*5/8, i-shr*3/8, ...
                    i+shr*3/8, i+shr*5/8, i+shr*3/8, i-shr*3/8], color(k))
154         else
155             patch([j, j+shr*0.5, j+shr*0.5, j, j-shr*0.5, ...
                    j-shr*0.5], [i-shr*5/8, i-shr*3/8, i+shr*3/8, ...
                    i+shr*5/8, i+shr*3/8, i-shr*3/8], color(k))
156         end
157     end
158 end
159
160
161 if plotmode==1 % If plotmode==1, the first plot stays untouched; ...
    Exit the loop
162     break
163 end
164
165 end
166 end

```

Bibliography

- [D'E12] John D'Errico. *fminsearchbnd*, *fminsearchcon*. 2012. URL: <http://de.mathworks.com/matlabcentral/fileexchange/8277-fminsearchbnd--fminsearchcon/content/FMINSEARCHBND/fminsearchbnd.m>.
- [Dok12] Jiro Doke. *Custom GINPUT*. 2012. URL: <http://de.mathworks.com/matlabcentral/fileexchange/38703-custom-ginput/content/ginputc.m>.
- [FH63] HL Frisch and JM Hammersley. "Percolation processes and related topics". In: *Journal of the Society for Industrial & Applied Mathematics* 11.4 (1963), pp. 894–918.
- [GB06] Ulrike Genschel and Claudia Becker. *Schließende Statistik: Grundlegende Methoden*. Springer-Verlag, 2006.
- [GM90] Christian Gourieroux and Alain Monfort. "Simulation based inference in models with heterogeneity". In: *Annales d'Economie et de Statistique* no. 20/21 (1990), pp. 69–107.
- [GM97] Christian Gourieroux and Alain Monfort. *Simulation-based econometric methods*. Oxford University Press, 1997.
- [Gri99] Geoffrey Grimmett. *Percolation*. Springer, 1999.
- [Han82] Lars Peter Hansen. "Large sample properties of generalized method of moments estimators". In: *Econometrica: Journal of the Econometric Society* (1982), pp. 1029–1054.
- [Hof+12a] Jochen Hoffmann et al. "Solid-phase PCR in a picowell array for immobilizing and arraying 100000 PCR products to a microscope slide". In: *Lab on a Chip* 12.17 (2012), pp. 3049–3054.
- [Hof+12b] Jochen Hoffmann et al. "Universal protocol for grafting PCR primers onto various lab-on-a-chip substrates for solid-phase PCR". In: *RSC Advances* 2.9 (2012), pp. 3885–3889.
- [Mic08] Stephen Michael. *KD Tree Nearest Neighbor and Range Search*. 2008. URL: <http://kr.mathworks.com/matlabcentral/fileexchange/7030-kd-tree-nearest-neighbor-and-range-search>.