Johannes Gutenberg University

Bachelor Thesis

# Automated Data Evaluation of Microscopy Images for Diffusion Model Fitting

*Evgeny Gorelik*

supervised by
Prof. Dr. Elmar Schömer
and
Prof. Dr. Sebastian Seiffert

February 23, 2019

# Contents

# 1 Introduction

This thesis focuses on the algortithm development and implementation for evaluation of experimental diffusion data obtained in *FRAP* (section **??**) experiments. The work is composed of four sections.
The first section describes the theory of diffusion and the method of diffusion coefficient determination from *FRAP* experiments.
The second part is discussed in section 3 which deals with image processing topics such as noise reduction using background images and smoothening filters. Also some methods are presented which aid the calculation in the case of unexpected experiment developments. This includes the problem that the center of the distribution does not align with the center of the image. Furthermore methods are presented which serve the reduction of the data to the minimal amount, which still represents the complete information of the diffusion process. These methods are based on the assumption that the diffusion is isotropic, meaning the substances diffuse in each direction with equal rates. The resulting data can then be fitted using numerical procedures described in section 4.
The basic idea of the fitting procedures described in section 4 is to find parameters for a function so the difference to the data is minimal. This approach is referred to as the calculation of the least squares to the data. It is an effective approach to find optimal parameters for a function, so that it represents the data as precise but as simple as possible.
In part four of the calculation, which is discussed in section 5, the determined parameters of the single images are then used to describe the overall diffusion process, regarding its diffusion rate and its number of components. Calculation methods such as linear regression are presented and the obtained values are transformed to fit physical units.
Different combinations of the presented functions can be used to evaluate different experimental data. In this thesis possible alternatives and their individual advantages or disadvantages are presented. The work flow and the order of procedures is based on the preceding work of S. Seiffert and G. Hauser extended with further methods. Also the thesis provides `MATLAB` code for all mentioned functions and the evaluations of their performance.

## 2 *FRAP* and Diffusion Physics

As stated in "[d]iffusion is a process by which matter is transported from one part of a system to another as a result of random molecular motions."[7][1.1,p. 1] Diffusion occurs in all states of matter, gaseous, liquid and even solid with different diffusion rates. The main topic of this study is diffusion of substances in solution (liquid phase) although the diffusion formalism can also be used for description of gaseous and solid state diffusion processes.
Diffusion is an event based on pure statistical likelihood. If the probabilities of this event equal on every point of interaction the diffusion is referred to as being isotropic. For the experiments which are evaluated by the introduced procedures the diffusion is expected to be isotropic.
The transfer rate $F$ of particles along one dimension is given by FICKs first law

$$F = -D\frac{\delta C}{\delta x} \tag{2.0.1}$$

with $D[\frac{m^2}{s}]$ being the diffusion coefficient, $C$ the concentration of the particles. " The negative sign [...] arises because diffusion occurs in the direction opposite to that of increasing concentration"[7][1.2,p. 2] The first FICKs law describes a snapshot of the particle diffusion while the second law can be used to analyze the diffusion process over time. The second law is defined by equation

$$\frac{\delta C}{\delta t} = -D\frac{\delta^2 C}{\delta^2 x} \tag{2.0.2}$$

Integration over time $t$ of (2.0.2) results in the following equation

$$C(x,t) = \frac{M}{\sqrt{4\pi Dt}^d}\exp(-\frac{x^2}{4Dt}) \tag{2.0.3}$$

with $M$ being the total amount of diffusing particles and $d$ the dimensionality of the diffusion[1].
Equation (2.0.3) corresponds to a normal (Gaussian) distribution.

$$C(x,t) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{x^2}{2\sigma^2}) \tag{2.0.4}$$

with $4Dt$ being equal to $2\sigma^2$ and pre-exponential scaling factor $M$. This means that at any given time the diffusion profile can be described by a Gaussian function. Hence the diffusion coefficient $D$ can be calculated by

$$D = \frac{\sigma^2}{2t} \tag{2.0.5}$$

By evaluating the variance development over time the diffusion coefficient $D$ can be determined.
*Fluorescence Recovery after Photo-bleaching* (short *FRAP*) is a method for analyzing the diffusion behavior of substances. For the *FRAP* experiment the substances must contain a fluorescent tag. The fluorescent signal is recorded using a confocal microscope. A very intensive focus beam is used to bleach the substance in a certain small area. The diffusion of the particles in the probe cause the spread of the bleached area. From a series of snapshot images quantitative information about the diffusion can be gained.
After recording the initial intensity the probe is bleached by focusing a intense light beam on a spot on the surface of the substances. Thereby the layers up to a certain depth from the surface

[1]This study restricted to two dimensional diffusion processes. However the presented procedures in general are applicable to one and three dimensional diffusion.

lose their fluorescent properties and the light intensity is smaller the bleached area in contrast to the unbleached surrounding. At this point the diffusion process between bleached and unbleached regions commences by the exchange of particles in a random manner. From the time needed for the diffusion process to finish, which is the point at which the bleached particles are evenly spread across the whole image, information about the concentration and the amount of the diffusing material can be extracted. As mentioned in [3][1.2,p. 16] the fluorescence is a useful property to measure since the change of the concentration of the substances is proportional to the shift in intensities over time.

These concepts of diffusion outwards from a single point, line or plane in space is possible only in theory.Nevertheless equations which describe the diffusion outwards from these shapes are still applicable to experiments by assuming that the diffusion process initiated from a shape with no volume before the experiment was supervised. This assumption can be used for mathematical purposes as described in [3][2.3.1,p. 39] by determining the development of the diffusion and traced back in time to the point, where the diffusion was theoretically initiated.

In the context of the provided experiments the assumption leads to the conclusion that the first image is not the initial state of the diffusion but the first observed state. The bleaching process therefor is regarded as initiating from a dirac impulse at some point $t_0$ and the first image is only taken after $t_f$ time has passed.

# 3 Image Preprocessing

For the calculation a dataset is required, which consists of images of the diffusing process taken over passing time. As described in [2][1.3,p. 13]each image contains information, which can be described by the formula

$$\overrightarrow{data_t} = \overrightarrow{I}_t + \overrightarrow{N}_t - \overrightarrow{G}_t \tag{3.0.1}$$

with $\overrightarrow{I}_t \in \mathbb{N}^{w \times h}$ being the ground intensity matrix, $\overrightarrow{N}_t \in \mathbb{N}^{w \times h}$ being the noise and $\overrightarrow{G}_t \in \mathbb{N}^{w \times h}$ being the actual Gaussian distribution which contains the information about the state of the diffusion for image $t$. Since determining the specifics of $\overrightarrow{G}$ is the aim of the calculations of the methods in 4 the aim of this section is to annihilate the other parameters so that the content of $\overrightarrow{G}$ is as accessible and undistorted as possible. So this chapter will tackle the process of determining $\overrightarrow{I}$ and reduce the influence of $\overrightarrow{N}_t$ on the calculation. Furthermore methods for extracting the data from the two dimensional image and transforming it into a one dimensional array are presented which serve the efficiency of the consequent determination of $\overrightarrow{G}$.
In order to process images, these have to be loaded into `MATLAB`. Therefor `MATLAB` supports a variety of image formats, including *.tif*. By using `ugetopen()` it is possible to select multiple images and using `imread()` to import the image data into `MATLAB`. The stack of images is then stored in an three dimensional array, where

$$\overrightarrow{stack} \in \mathbb{N}^{w \times h \times T} := \overrightarrow{stack}_{i,j,t} = (\overrightarrow{data_t})_{i,j} \tag{3.0.2}$$

with $\overrightarrow{data_t}$ being the data of image $t$. This accordingly asserts, that all images must be the same width $w$ and the same height $h$. Though not all following algorithms depend on this assertion and `MATLAB` provides the possibility of using `cell` for stacking images with differing dimensions, the format of the images taken by a device like a microscope is expected to be equal at least for the duration of the measurement of a FRAP diffusion process. Especially normalization operations, which have to be performed for all images profit from the array-based data structure.

## 3.1 Background Noise Reduction

Since all images are taken with an optical device, they naturally are exposed to noise which can influence the calculation. A useful method to reduce the overall background noise is to provide a background image taken before the experiment in order to exclude technical inaccuracies of the measurement for example defect pixel sensors of the optical device or unevenly distributed lightning properties of the experiment. If a background image is provided it can be subtracted from the individual images in the stack. When subtracting, the resulting image will contain the difference of a pixel in the image to the pixel at the same location in the background image.

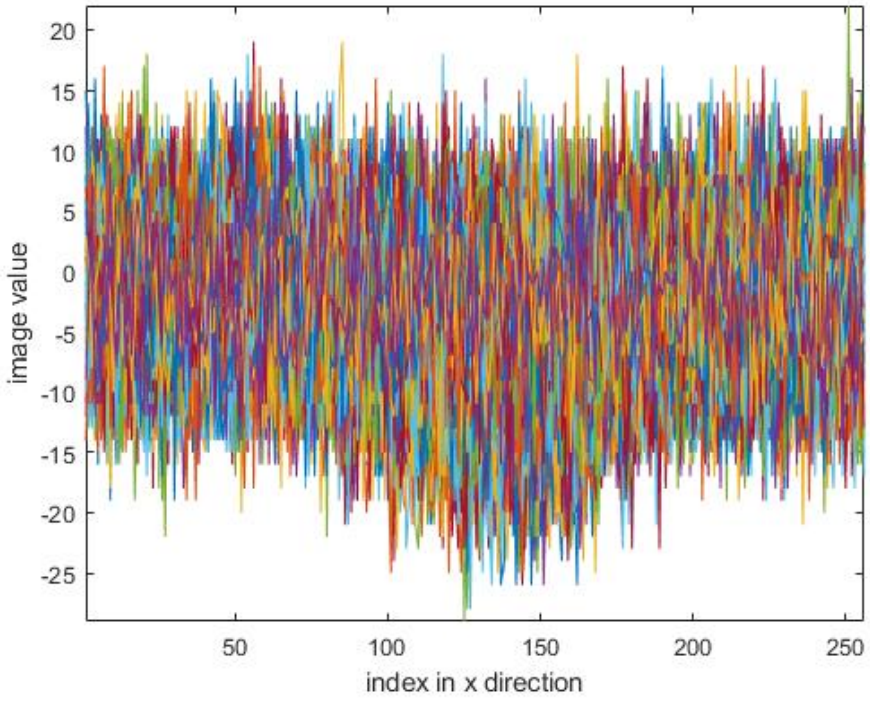



Figure 1: Subtraction of background image from Gaussian image from side perspective

Here Figure 1 shows the result of the subtraction of the background image from the Gaussian distributed image. The resulting values of the image can are expected to be shifted into the negative region. Is this not the case then the background image is not a precise representation of the ground intensity in the experiment or the ground intensity has changed throughout the experiment. A solution regarding this problem is presented in 3.5.1.
Since the values in the result of the subtraction are mostly negative the whole further calculation would be based on determining negative values. For a more statistical approach of gaining results the values are mirrored along the middle of the whole span of the values.

$$(image_{new})_{i,j} = \frac{|\max(\overrightarrow{image}) - \min(\overrightarrow{image})|}{2} - (image_{old})_{i,j} \tag{3.1.1}$$

This has little computational profit except replacing a number of subtractions with additions. But since all other parameters of the calculation are described by positive parameters flipping the values by using (3.1.1) also serves the consistency throughout the calculation and helps to catch negative thus false results without further conditions.

## 3.2 Using Filter to Smooth Image

The images as taken from an optical device like a microscope can be expected to be noise on some level. Especially images where the amplitude of the distribution is approaching the noise level this can affect the calculation. By using image processing methods there are ways to smoothen the noise of the images, so the calculation can still lead to accurate results. In the following two methods to smoothen the images are presented and compared by their advantages regarding the fitting problem.

### 3.2.1 Smoothing Images with Gaussian Filter

The gaussian filter is a common method to smoothen an image that contains noise. The way the method operates is as decribed in [5][3.5.2,p. 87] by iterating over each pixel and assigning it a value which is influenced by its $(k \times k) - 1$ neighbors with $k$ being an odd number starting from 3. This is often refered to as a **box filter**. For a matrix $\overrightarrow{image}$ and a box $\overrightarrow{filter}$ where

$$\overrightarrow{image}_{old} = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix} \in \mathbb{R}^{n \times m}, \overrightarrow{filter} = \begin{pmatrix} f_{1,1} & \cdots & f_{1,k} \\ \vdots & \ddots & \vdots \\ f_{k,1} & \cdots & f_{k,k} \end{pmatrix} \in \mathbb{N}^{k \times k}$$

the algorithm then creates a new image

$$\overrightarrow{image}_{new} = \begin{pmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \cdots & b_{m,n} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

with entries

$$b_{i,j} = \frac{\sum_{l=0}^{k^2-1} a_{(i-1)+(l \bmod k),(j-1)+(l \div k)} \cdot f_{l \bmod k, l \div k}}{\sum_{l=0}^{k^2-1} f_{l \bmod k, l \div k}} \tag{3.2.1}$$

Should an entry $a_{(i-1)+(l \bmod k),(j-1)+(l \div k)}$ not exist because the index is out of range of the matrix the value and the according entry $f_{l \bmod k, l \div k}$ is set to 0 so they don't appear in the calculation and the values closer then $k$ pixels away from the edge still can be calculated. $k$ should be an odd number so the pixel for which the value is determined by equation (3.2.1) is in the exact middle of the $\overrightarrow{filter}$ matrix.
For the Gaussian box filter the values in $\overrightarrow{filter}$ are distributed according to their value in a two dimenstional Gaussian distribution. The entries of $\overrightarrow{filter}$ then are

$$f_{i,j} = \left\lceil \frac{1}{\sqrt{2\pi}} e^{-\frac{(i-\lceil \frac{k}{2} \rceil)^2 + (j-\lceil \frac{k}{2} \rceil)^2}{2\sigma^2}} \right\rceil \tag{3.2.2}$$

Since by its own the Gaussian function used in (3.2.2) does not provide discrete values the results of the function used in (3.2.2) have to be rounded to the next integer by using $\lceil \dots \rceil$. After calculating the values of $\overrightarrow{filter}$ a convolution in the sense of (3.2.1) provides a smoothened image with reduced noise amount.
For `MATLAB` the Gaussian filter is already provided by the function `imgaussfilt(A,'FilterSize',3)`. Although reducing the noise in the image is supposed to provide better results in further calculation using a filter that smoothens the whole image might actually alter the the overall disrtibution of $\overrightarrow{G}$ as used in (3.0.1) by smoothing the top of the distribution. Hence an approach is helpful that targets the high frequency noise in the image and does not alter the low frequency of the distribution.

### 3.2.2 Smoothing Images with Fourier Filter

An alternative to filtering noise with the Gaussian box filter is to use a frequency filter that can operate in the frequency domain and thereby leave the overall form of the Gaussian distribution as unchanged as possible. As described in [5][3.5.2,p. 78] filtering in the frequency domain requires the following procedure:

1. Compute the Fourier transform $\overrightarrow{F}(\overrightarrow{image})$ of the image.
2. Filter frequencies by multiplying $\overrightarrow{F}(\overrightarrow{image})$ with a weight matrix $\overrightarrow{filter}$ elementwise.
3. Compute the inverse Fourier transform of the elementwise product of $\overrightarrow{F}(\overrightarrow{image})$ with $\overrightarrow{filter}$.

For computing the Fourier transform of an image the `MATLAB` function `fft2(image)`[2] that calculates the discrete Fourier transform $\overrightarrow{F}$ with entries

$$F_{p,q} = \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} (e^{-\frac{2\pi i}{m}})^{j(p-1)} (e^{-\frac{2\pi i}{n}})^{k(q-1)} \cdot image_{j+1,k+1} \tag{3.2.3}$$

for $p \in [1, \ldots, m]$ and $q \in [1, \ldots, n]$. Then the function `fftshift` is used sort the vaules so the frequencies increase from the middle of the image outwards. A low frequency filter then is applied by creating a mask with all values in the fourier domain above a thresholded frequency being set to 0 and the circle with radius `thres` around the center of the image being set to 1. This method leaves the Gaussian distribution unaltered since a Gaussian function in the fourier domain is invariant regarding translation. So the values in the middle of the image are asured to be part of the Gaussian distribution. The following `MATLAB` implementation presents a filtering method based on the fourier transformation.

```
function denoise = fourier_denoise_2d(values,thres)
% input: values as an matrix and treshold from which on high
    frequencies will be cut off
% output denoised values, with frequencys over treshold cut off

% transform to fourier
Y = fftshift(fft2(values));
[width,height] = size(Y);
% create grid for the filter
[x_coor,y_coor] = meshgrid(-width/2:width/2-1,-height/2:height/2-1);
% filter values greater than treshold radius
filter = ((x_coor).^2 + (y_coor).^2 <= thres^2);
% reverse transformation of filtered values
denoise = ifft2(ifftshift(Y.*filter));
end
```

Since both methods are parameterised with the sigma value and the filter size for `imgaussfilt` and `thres` for `fourier_denoise_2d` the results may vary for different distributions.

[2] https://de.mathworks.com/help/matlab/ref/fft2.html

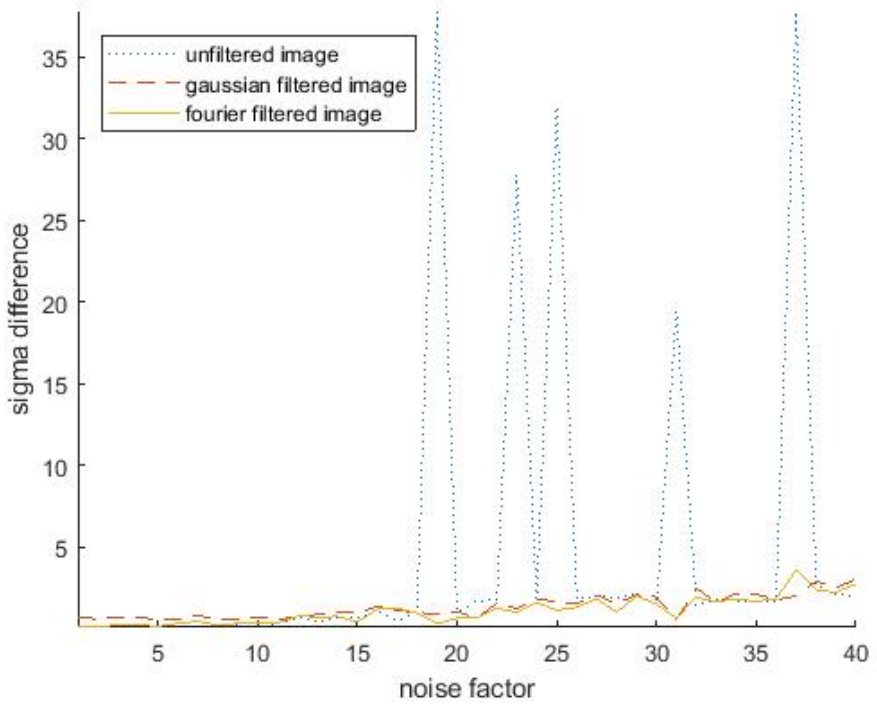



Figure 2: Difference between actual and fitted $\sigma$ values

As seen in Figure 2 `fourier_denoise_2d` achieves slightly better results for the given image series. However the usage of either `imgaussfilt` or `fourier_denoise_2d` improves the results of the fit thus the fitting of an unfiltered image should be avoided.

### 3.3 Finding Image Center

Not all images taken from the microscope have their diffusion center in the center of the corresponding image. This offset has to be determined for further calculations and finally the fitting model. The center of the distribution is part of the gauss function in form of $\vec{c} \in \mathbb{N}^2$

$$G(\vec{x}) = A \cdot e^{-\frac{(\vec{x}-\vec{c})^2}{2 \cdot \sigma^2}} \tag{3.3.1}$$

So an algorithm for finding the distribution center $\vec{c}$ should achieve

1. accuracy
2. noise resistance
3. computation speed.

In the following three methods are presented which depending on the quality of the Gaussian distribution image differ in results regarding accuracy and noise resistance. Since computation speed on the scale of the given input does not differ largely the computation intensity of all three methods is regarded to be almost equal.
Though technically a fitting algorithm could determine the center $\vec{c}$ as part of the fitting process the aim of the following methods is to approximate the values as precise as possible so $\vec{c}$ is no longer part of equation (3.3.1). Hence after finding values for $\vec{c}$ the image must be transformed, so the distribution center corresponds to the image center.

### 3.3.1 Maximal Image Value

For obtaining the offset value which corresponds to the distribution center $\vec{c}$ some further properties of the distribution can be used. Firstly the center $\vec{c}$ corresponds to the maximal value of the function.

$$\forall \vec{k} \in \mathbb{N}^2 \neq \vec{0} : G(\vec{c}) > G(\vec{c} - \vec{k})$$

Hence a simple algorithm idea for determining the center of the distribution is to search the image for its maximal value and to define it as the distribution center $\vec{c}$ for a given image $\overrightarrow{data} \in \mathbb{N}^{w \times h}$.

$$\vec{c}_{max} = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{N}^2, data_{x,y} = \max(\overrightarrow{data}) \tag{3.3.2}$$

An implementation in `MATLAB` is provided by following code.

```
function center = max_detection(image)
% input: image - data matrix for center detection
% output: center - coordinate of point with max value

% find maximal value of the data
[max_val] = max(max(image));
% find coordinates that correspond to maximal value
[x,y] = find(abs(image-max_val)<10^-10);
% vectorize the coordinates
center = [x y];
end
```

As simple as this implementation is its precision is not guaranteed due to noise in the data of the image. Nevertheless the property of the equality of the distributions maximum coordinate with the center coordinate $\vec{c}$ can be used, by taking more data into account.

### 3.3.2 Discrete Integration

As proposed in [2, p. 30] the center can be determined more precisely and noise resistant by gathering the sum of all points along the x axis for each point on the y axis and repeat the procedure analogously for all points along the y axis. This results in two vectors

$$\vec{x}_{sum} = \begin{pmatrix} \sum_{i=1}^{n} data_{1,i} \\ \vdots \\ \sum_{i=1}^{n} data_{w,i} \end{pmatrix} \in \mathbb{N}^w, \vec{y}_{sum} = \begin{pmatrix} \sum_{i=1}^{n} data_{i,1} \\ \vdots \\ \sum_{i=1}^{n} data_{i,h} \end{pmatrix} \in \mathbb{N}^h$$

with the center $\vec{c}$ being determined by

$$\vec{c}_{sum} = \begin{pmatrix} i \in \mathbb{N} : (\vec{x}_{sum})_i = \max(\vec{x}_{sum}) \\ j \in \mathbb{N} : (\vec{y}_{sum})_j = \max(\vec{y}_{sum}) \end{pmatrix} \in \mathbb{N}^2, \tag{3.3.3}$$

Thus the individual entries of $\vec{x}_{sum}$ and $\vec{y}_{sum}$ are discrete integrations over one image line along $x$ or $y$. The entry index for which the entry of $x_{sum}$ contains the maximal value of all integrated

values corresponds to the x coordinate for the distribution maximum. Since the integrated values contain more then one sample, the methods results are not effected by noisy values on the cap of the distribution as much as the method described in 3.3.1. This method of determining the center is therefore more noise resistant.

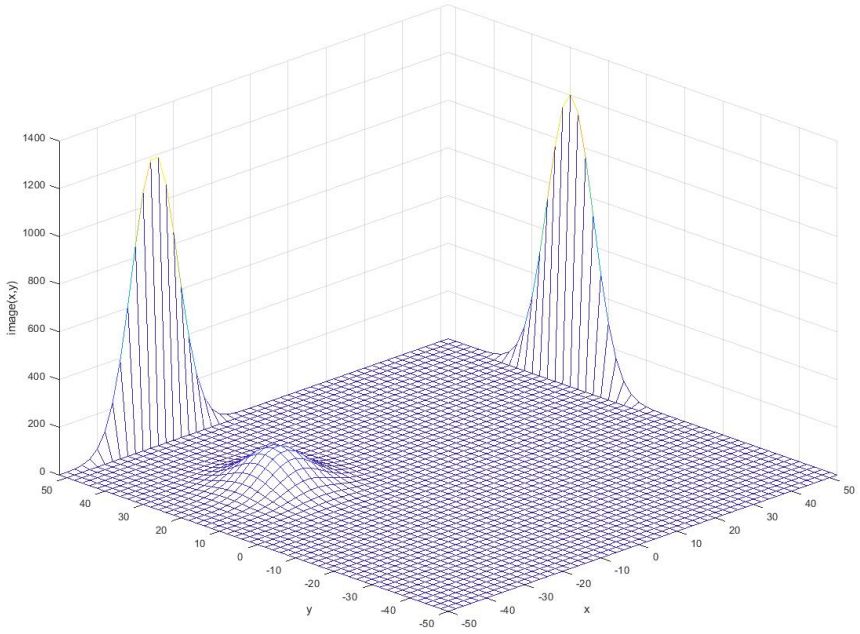



Figure 3: Sum vectors along x and y axis

Figure 3 depicts the idea of (3.3.3) where the profile of the Gaussian distribution in the last row and column of the image visualizes the sum vectors as described in (3.3.3). The maximal point of these sum vectors of this noiseless Gaussian function align with the maximum of the distribution. A `MATLAB` implementation for determining the center by using (3.3.3) is presented in the following.

```
function center = gaussian_detection(image)
% input: image - gaussian distributed data matrix
% output: center center coordinates in image detected by gaussian

% determine image size
[w,h] = size(image);
gauss_x = zeros(w,1);
% generate the sum iterating through rows
for i = 1:h
    gauss_x(i) = sum(image(i,:));
end
gauss_y = zeros(h,1);
% generate the sum iterating through columns
for i = 1:w
    gauss_y(i) = sum(image(:,i));
```

```
16 end
17 % find maximal values of both vectors and their indices
18 [~,x_max_index]=max(gauss_x);
19 [~,y_max_index]=max(gauss_y);
20 center = [x_max_index,y_max_index];
21 end
```

Though noise resistance of this method outperforms the noise resistance of (3.3.2) the limitation this method has lies in the discrete properties of the underlying grid which the data is bound to and therefore center coordinates cannot exceed integer precision.

### 3.3.3 Gaussian Concentrical Circles

To overcome this hurdle further properties of the data can be used. So another useful property of a two dimensional gauss distribution is its point symmetrical structure with regard to the center point $\vec{c}$ of the distribution. Basically the image consists of concentric circles which are distributed along the height dimension of the data.

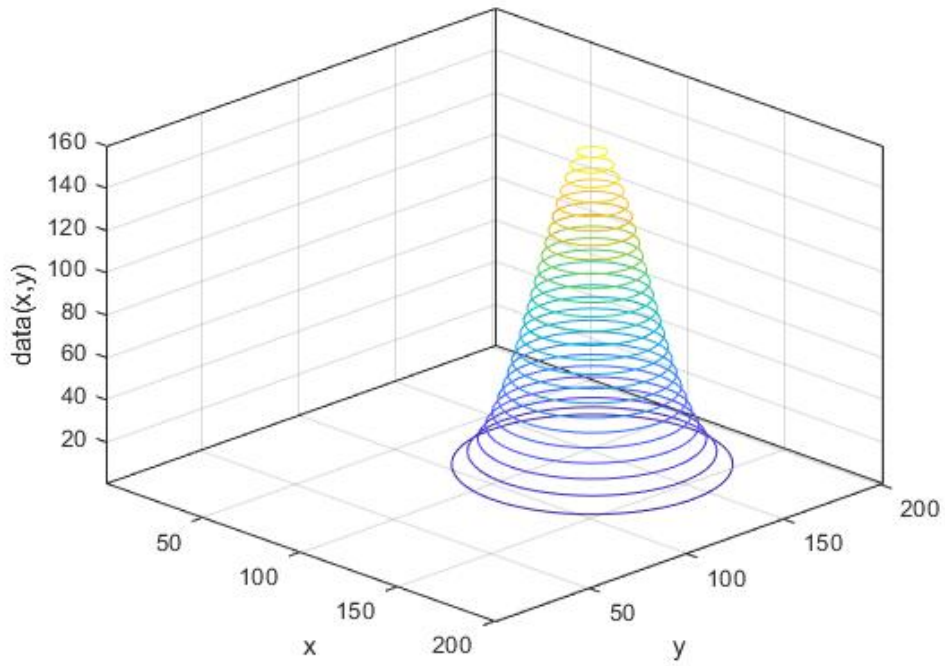



Figure 4: Gaussian Distribution as Concentrical Circles

Hence for a representative height level all points $(x, y)$ with the same value $data_{x,y}$ form a circle. And since all circles of the gauss distribution are concentric, the determination of their center will result in being the center $\vec{c}$ of the gauss distribution.

$$\vec{c}_{circular} = \min_{\vec{x} \in EQ} |\vec{c} - \vec{x}|, EQ := \{\vec{a}, \vec{b} \in \mathbb{R}^2 : \overrightarrow{data_{\vec{a}}} = \overrightarrow{data_{\vec{b}}}\} \tag{3.3.4}$$

A representative height hereby is a value for which the amount of sample points is balanced. Setting the value to high or to close to the maximum of the distribution will result in too few sample points

in the calculation which would increase the precision loss due to noise. If the height value is set to low the majority of the sample points might not be part of the distribution and thereby misplace the center of the circle. Choosing a height value of $\frac{A}{2}$ is hence a good compromise and produces stable results. The calculation can also include a height interval $[\frac{A}{2}-c, \frac{A}{2}+c], c \in \mathbb{N}$ which generates stability for especially noisy data. Definition (3.3.4) is then extended to

$$\vec{c}_{circular} = \min_{\vec{x} \in EQ} |\vec{c} - \vec{x}|, EQ := \{\vec{a} \in \mathbb{R}^2 : |\overrightarrow{data}_{\vec{a}} - \frac{A}{2}| < c\} \tag{3.3.5}$$

Ideally this creates a precise coordinate for the center location. However the more the substances diffuse the closer the amplitude of the Gaussian function $A$ gets to the amplitude of the noise in the image.

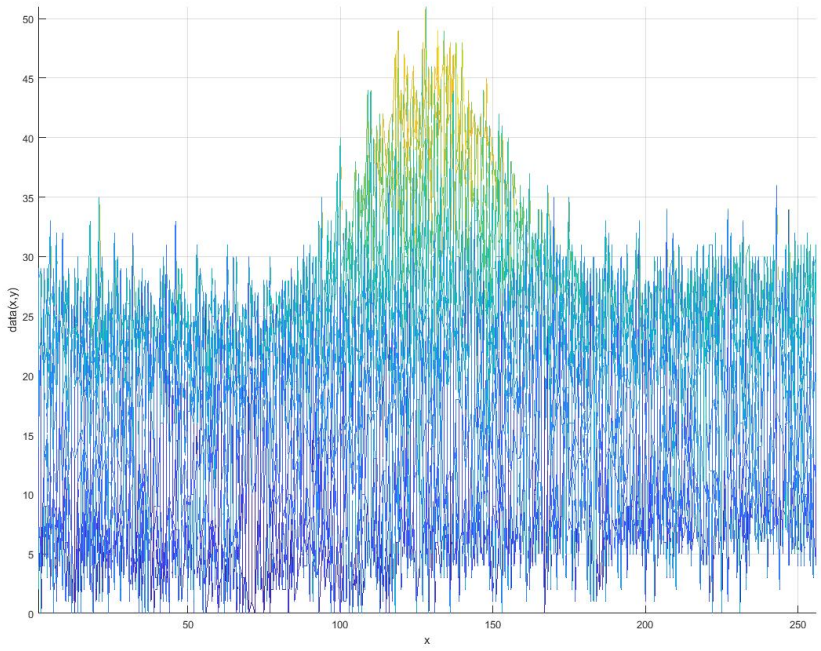



Figure 5: Gaussian Function with Low Amplitude from the Side Perspective

For example Figure 5 shows a Gaussian distribution for which the amplitude $A$ is low and approaches the height of the noise peaks. Though optically the cap of the distribution is still distinguishable from the noise, evaluating the center of all points at half of the maximal value will already include points from the ground noise. As a result the center coordinate $\vec{c}$ will no longer contain precise values for the center of the Gaussian distribution and further calculations will produce false results.
Consequently the height value for evaluation must adapt depending on the progress of the diffusion process. For Figure 5 evaluating at $70-80\%$ would return a more accurate center representation. The following `MATLAB` implementation takes the evaluation level as an optional second input argument and the interval for evaluation as an optional third argument.

```
1 function center = circular_detection(image, eval_height_perc, eval_int)
```

```
% input: image - gaussian distributed data matrix, eval_height_perc -
    evaluation height in percent, eval_int - interval around
    eval_height_perc for calculation
% output: center - concentrically determined center

% default values for optional parameters
if isempty('eval_height_perc') || ~exist('eval_height_perc','var')
     eval_height=max(max(image))/2;
else
     eval_height=max(max(image))*eval_height_perc;
end
if isempty('eval_int') || ~exist('eval_int','var')
     eval_int=1;
end
% find points, which lie in the evaluation interval
eval_ind=find(abs(eval_height(1)-image)<=eval_int);
% gather their values from the indices
[x,y] = ind2sub(size(image),eval_ind);
% define a function of least squares for minimizing
func=@(p)sum(sum((p(1)-x).^2+(p(2)-y).^2));
% initial parameter vector
center = [mean(x) mean(y)];
% find function minimum so precision exceeds integer values
center = fminsearch(func,center);
end
```

Due to the non integer characteristics of this method its accuracy exceeds the ones of the before mentioned. In the case of noise resistance the results depend on the difference of the distribution amplitude and noise amplitude peaks.
So to automate the process of determining `eval_height` an approach from 3.3.2 can be used. By determining the discrete integral values along both image axis the resulting difference of the maximal $V_{max}$ and the minimal $V_{min}$ value gives a representation of the noise to signal ratio in the image. Setting `eval_height` to the middle between the maximum and the minimum $\frac{\frac{V_{max}}{V_{min}}+1}{2}$ and evaluating the points maximal 10 integers above and below `eval_height` provides stable results. Should a 10 integer evaluation interval already include points, which are not part of the distribution or exceed the maximal height `amp` of the image values a evaluation interval of $\texttt{amp} - \texttt{eval\_height} - 1$ is used. With `eval_height` calculated equation (3.3.5) becomes

$$\vec{c}_{circular} = \min_{\vec{x} \in EQ} |\vec{c} - \vec{x}|,$$

$$EQ := \{\vec{a} \in \mathbb{R}^2 : |\overrightarrow{data}_{\vec{a}} - \texttt{eval\_height} < \min(10, \texttt{amp} - \texttt{eval\_height} - 1)\} \tag{3.3.6}$$

An `MATLAB` implementation of a center detection function that utilizes (3.3.6) is presented in the following.

```
function center = circular_detection(image,eval_height_perc,eval_int)
% input: image - gaussian distributed data matrix
% output: center - concentrically determined center

```

```
% determine the amount of signal in the data by summing up the values
    along both axes
total_int = sum(image,1)+transpose(sum(image,2))
% calculate the evaluation height and the evaluation interval around it
eval_perc = (min(total_int)/max(total_int)+1)/2;
eval_height=max(max(image))*eval_perc;
eval_int=min(max(max(image))-eval_height-1,10);
% find points, which lie in the evaluation interval
eval_height_ind=find(abs(eval_height(1)-image)<=eval_int);
% gather their values from the indices
[x,y] = ind2sub(size(image),eval_height_ind);
% define a function of least squares for minimizing
func=@(p)sum(sum((p(1)-x).^2+(p(2)-y).^2));
% initial parameter vector
center = [mean(x) mean(y)];
% find function minimum so precision exceeds integer values
center = fminsearch(func,center);
end
```

As mentioned before this algorithm is capable of determining the center of the distribution more precise then the algorithms described in 3.3.1 and 3.3.2.

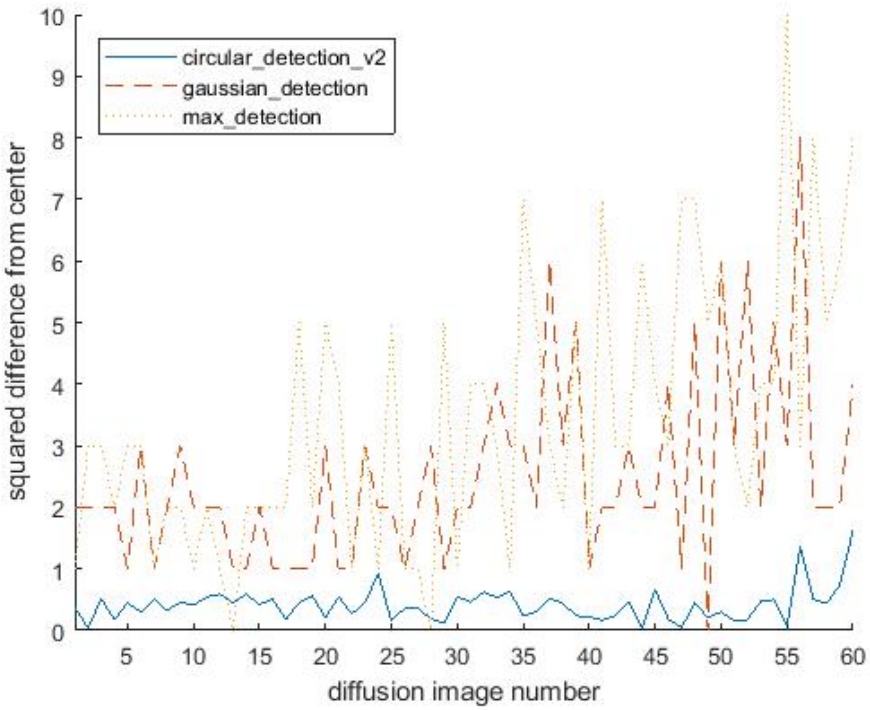



Figure 6: Comparisson of Center Detection Methods

Figure 6 depicts the difference of the center found by the methods from the actual center for a modelled diffusion process. As seen the quality of the other center finding methods decreases for further developments of the diffusion which is due to the approach of the amplitude of the Gaussian

distribution towards the amplitude of the noise. The overall precision of `concentrical_circle_v2` does not alter which shows the methods resistance for bad signal to noise ratios.

### 3.3.4 Rotational Invariance

The center of a Gaussian distribution is invariant to rotation as described in [2][1.2.2, p. 14] so this property can be used to determine the distributions center. As proposed in [2][1.2.2, p. 14] for the center of the distribution $\vec{c}$ the difference of a window around $\vec{c}$ to its values rotated by 90 degrees lies at a local minimum. This can be interpreted as a least squares problem where the minimum of equation

$$f_{\overrightarrow{A},W}(\vec{x}) = \sum_{i=-W}^{W} \sum_{j=-W}^{W} \left(A_{x_1+i,x_2+j} - A_{x_1-j,x_2+i}\right)^2 \tag{3.3.7}$$

for a given matrix $\overrightarrow{A}$ and a window size $W$ around the expected center $\vec{x}$ has to be found. By minimizing $f_{\overrightarrow{A},W}$ the center $\vec{c}$ can be determined if the center is already approximated since as mentioned in [2][1.2.2, p. 18] the regions which are not part of the distribution also are invariant to rotation. For the approximation a simple method as the one described by equation (3.3.1) is sufficient since in only serves the determination of the region, in which $\vec{c}$ is expected to be. Unlike the method described in 3.3.3 the method (3.3.7) can evaluate integer values for $\vec{x}$ only. In order to exceed integer precision the values in the window with window size $W$ around $\vec{x}$ have to be interpolated. Therefor the evaluation function is adapted, so that it interpolates the values inside the window and then calculates the squared difference between the interpolated window and its 90 degree rotation. So the grid upon which the function evaluates is finer and the values are more precise. `MATLAB`s build in function `fminsearch` is then used to minimize the values. The following code presents a possible `MATLAB` implementation.

```
function center = sym_correct(image,center,windowsize)
% input: image - gaussian distributed matrix, center - approximate
    center
% coordinates, windowsize - size of the region that is scoped
% output: center - corrected center coordinates

% initialize optional parameters
if isempty('center') || ~exist('center','var')
     center=size(image)/2;
end
if isempty('windowsize') || ~exist('windowsize','var')
     windowsize=10;
end
% global variables needed for evaluation
global matrix
global w_size
global fact
matrix=image;
fact=0.1;
w_size=windowsize;
% find point that is most invariant to rotation
center=fminsearch(@eval,center);
```

```
22 end
23 % eval function for numerical evaluation
24 function fun_val = eval(center)
25 global matrix
26 global w_size
27 global fact
28 % extract values in a window around center
29 window = matrix(round(center(1))-w_size:round(center(1))+w_size,round(
       center(2))-w_size:round(center(2))+w_size);
30 [X,Y] = meshgrid(center(1)-w_size:center(1)+w_size,center(2)-w_size:
       center(2)+w_size);
31 [Xq,Yq] = meshgrid(center(1)-w_size:fact:center(1)+w_size,center(2)-
       w_size:fact:center(2)+w_size);
32 % interpolate with the finer grid
33 window_p=interp2(X,Y,window,Xq,Yq);
34 fun_val=sum(sum((window_p-rot90(window_p)).^2));
35 end
```

Rotational invariance is a characteristic of the distribution, that has not been used in the previous methods and since `sym_correct` evaluates on a larger area the results are expected to be more stable, provided an appropriate parameter `windowsize`.

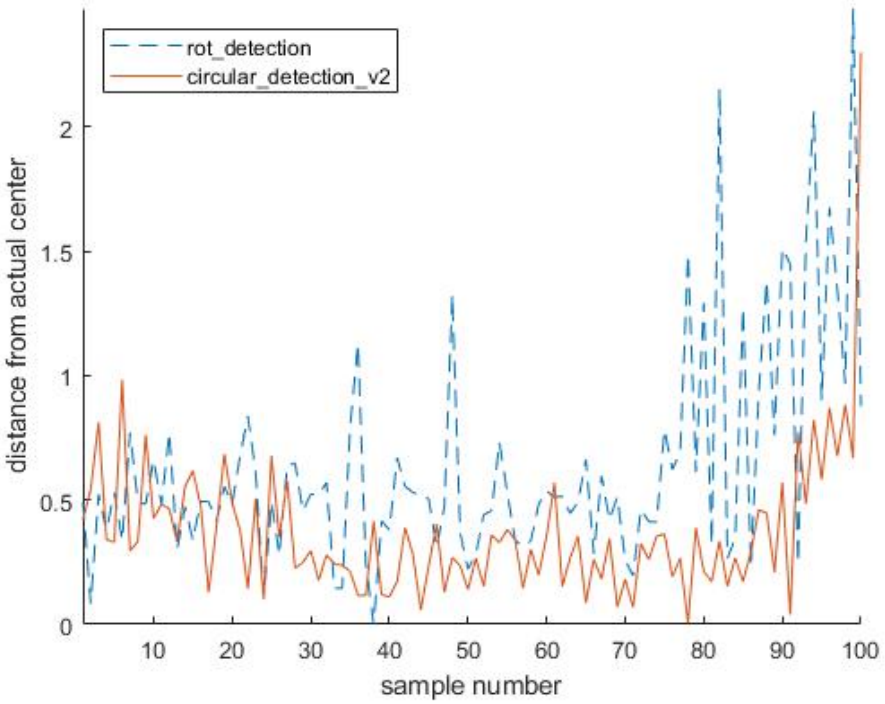



Figure 7: Distance to Actual Distribution Center

As seen in Figure 7 the distance to the actual center of the results calculated by `circular_detection_v2` are on average closer then the ones of `sym_correct` for this series of a modeled diffusion process.

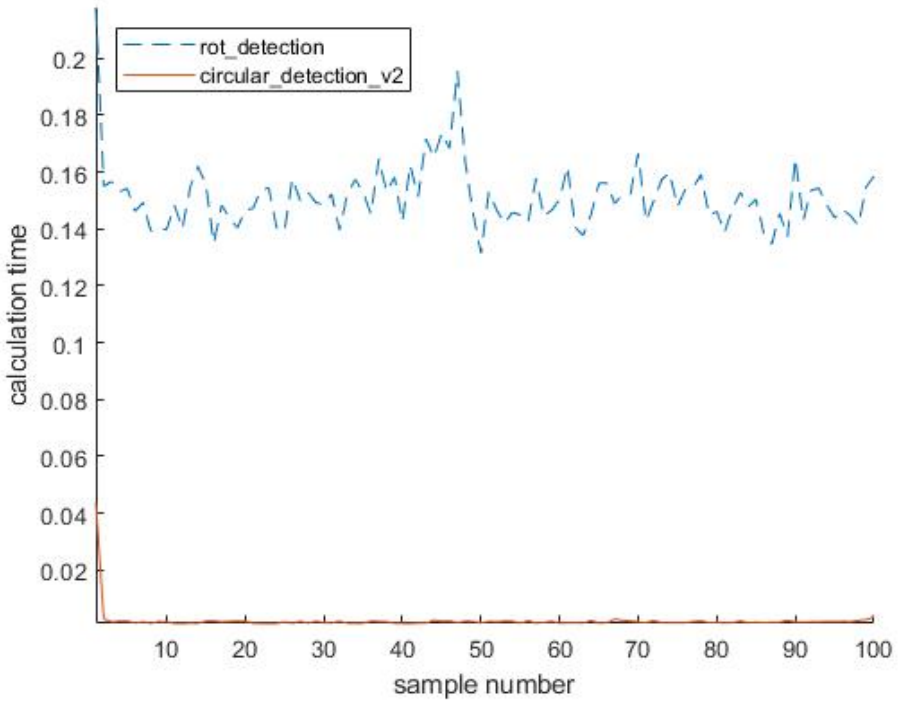



Figure 8: Calculation Time for Distribution Center

Also Figure 8 shows, that the time, which `sym_correct` needs for the calculation exceeds the time needed to calculate the center using `circular_detection` by an average factor of 70. The reason for that is the computation of the interpolated values.
Though the performance of `sym_correct` is suboptimal it might still be used as an alternative if the results of `circular_detection` are unsatisfying, since it holds the possibility of user interaction in the sense of correcting an approximate center coordinate value.

### 3.4 Gaussian Profile

The goal of this step of image preprocessing is to reduce the dataset to the minimal data which still represents the Gaussian distribution of the original image. Thereby the further procedure of fitting Gaussian distribution functions to the data have less over-fitting problems. These problems occur when due to noise the accuracy of a calculation decreases with the inclusion of a larger amount of data in the calculation. To avoid this problem the image in form of a data matrix is reduced to a data vector in certain ways, so that all information for the fitting process is preserved but the data amount is reduced from quadratic $O(n^2)$ to linear dimension $O(n)$.
For the reduction of the data matrix to a data vector the assumption is used, that each data point is point symmetrical to the axis line intersecting the previously calculated distribution center $\vec{c}$. This is not the case for experiments which have compounds with anisotropic[3] features. For these no reduction is possible. But assuming the point symmetry to the axis $\vec{c}_{sym}$ defined by

$$\vec{c}_{sym} = \begin{pmatrix} c_1 \\ c_2 \\ t \end{pmatrix} \in \mathbb{N}^3, \forall t \in \mathbb{N}$$

[3]Having properties that differ according to the direction of measurement.

each point has an inverse point.

$$\forall \vec{a} \in \mathbb{N}^3 : \exists \vec{b} \in \mathbb{N}^3 : \vec{b} = \begin{pmatrix} -(a_1 - c_1) \\ -(a_2 - c_2) \\ a_3 \end{pmatrix} \tag{3.4.1}$$

Assuming (3.4.1) two ways of reducing the data to a Gaussian profile are possible.

### 3.4.1 Concentric Averaging

As proposed in [2][p. 50] one possible solution is to iterate through the concentric circles from the center $\vec{c}$ outwards and to take the average of each circle as the height representing all values in the circle. The algorithm iterates through increasing radius values until the average of all height levels is calculated. The values the radius has in each iteration depends on a chosen sample rate. Since radial iteration includes a transformation from the discrete Cartesian coordinates which the indexes of the $\overrightarrow{data}$ matrix are represented by to polar coordinates a numerical error is not avoidable. The transformation then is calculated by

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos\rho \\ r\sin\rho \end{pmatrix}, r \in \mathbb{R}, \rho \in (0, 2\pi] \tag{3.4.2}$$

All $\overrightarrow{data}$ values with the same radius $r$ from the center point $\vec{c}$ are summed up and divided by their amount. For iterating values $r_i$ this results in a vector $\vec{v}_{circular}$ with entries

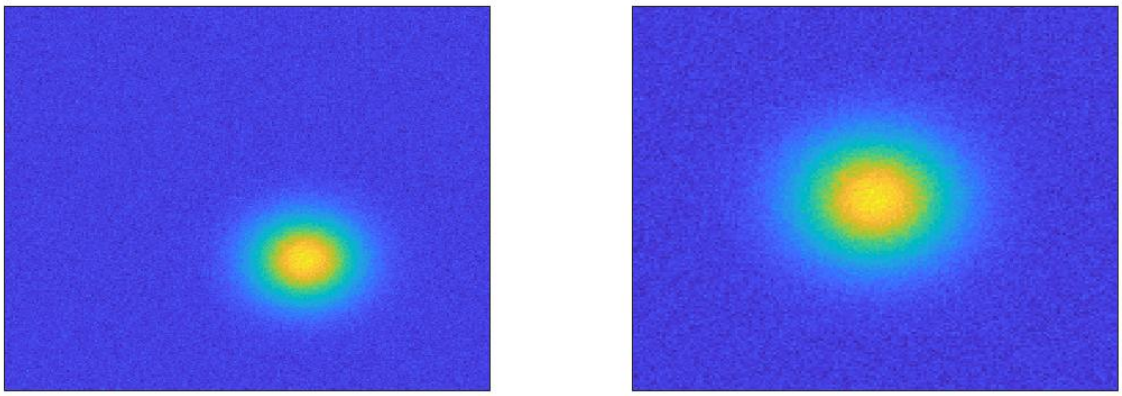
$$EQ_{r_i} := \{\vec{x} \in \mathbb{N}^2 : \vec{x} \approx \begin{pmatrix} r_i\cos\rho \\ r_i\sin\rho \end{pmatrix}, \rho \in (0, 2\pi]\}$$

$$\vec{v}_{circular} = \begin{pmatrix} \frac{\sum_{\vec{x}\in EQ_{r_1}} \overrightarrow{data}_{\vec{x}}}{|EQ_{r_1}|} \\ \vdots \\ \frac{\sum_{\vec{x}\in EQ_{r_m}} \overrightarrow{data}_{\vec{x}}}{|EQ_{r_m}|} \end{pmatrix} \tag{3.4.3}$$

The resulting vector contains a profile of the Gaussian distribution in a cropped region with equal distance to the center coordinate $\vec{c}$ in each direction. The crop size of the region and thereby the length $l$ of the profile vector is defined by

$$l = 2 \cdot \min\left(\begin{pmatrix} c_1 \\ c_2 \\ w - c_1 \\ h - c_2 \end{pmatrix}\right) \tag{3.4.4}$$

This is because for the points in the matrix which are further away from $\vec{c}$ then $\frac{l}{2}$ there are no inverse points as defined in (3.4.1).



(a) Full Gaussian Distribution Image (b) Cropped Gaussian Distribution Image

Figure 9: Cropping a Gaussian Distribution Image

For example Figure 9a has its center $\vec{c}$ offset from the middle of the picture, so the region which can be used to create a concentric averaged profile is reduced to the one displayed in Figure 9b.
To guarantee that the center of the distribution is in the middle of the profile vector, the profile vector length must be uneven. So an image of size $256 \times 256$ can have a maximal total of $255 \times 255$ points, for which (3.4.1) applies. Furthermore the center $\vec{c}$ doesn't have to contain integer values only. Since defining $\vec{c}$ as the center of a concentric circle as in (3.3.4) provides the center up to a decimal precision the fact that the actual distribution center can be located between two image pixels has to be taken into account.
Therefore a grid vector serves as a conversion method to avoid altering the physical size of an image pixel and being able to convert to physical meters after the calculation. The unit size of this vector in relation to the image pixel depends on the decimal number $\min(\vec{c} \bmod 1)$ of the calculated center. The grid covers the symmetrical interval $[-\frac{l}{2}, \frac{l}{2}]$, so that the previous determined center coordinate $\vec{c}$ is shifted to the center of the grid vector. Then the concentric mean values are calculated by using (3.4.3).

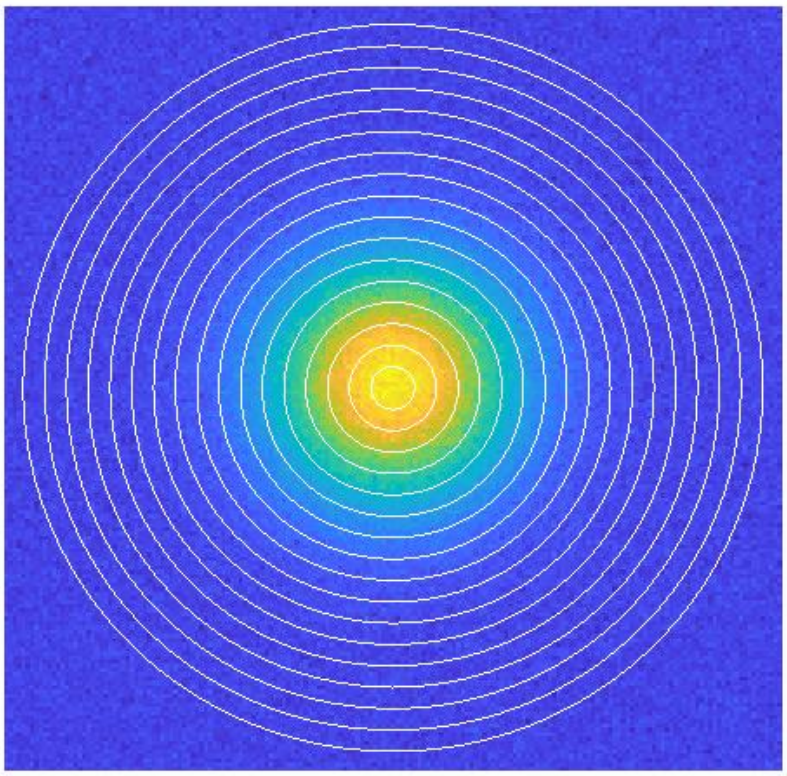


Figure 10: Gaussian Distributed Image with Concentrical Circles

Figure 10 shows a Gaussian distributed image and the concentric circles, along which the mean is calculated. The amount of circles, and thereby the rate at which the image is sampled is then increased to the minimal distance to the edge of the image $\frac{l}{2}$. A `MATLAB` implementation is provided here.

```
function [profile,grid] = concentrical_sum(img,center)
% input: img - data matrix with gaussian distribution, center -
    gaussian
% distribution center coordinates
% output: profile - vector with concentrical averaged values, grid -
    grid vector
% for counteracting size variation

% determine img dimensions and smallest distance from the center to an
% img border
[width,height] = size(img);
radius = min([center(1) width-center(1) center(2) height-center(2)]);
% define stepsize based on precision of the center
step_size=1-mod(radius,1);
% create grid vector and corresponding value vector
```

```
14 grid=-radius:step_size:radius;
15 grid_size=length(grid);
16 profile = zeros(1,grid_size);
17 % define radius vector which stepwise increases its distance to the img
18 % center
19 rho = 0:step_size:radius;
20 % define angle vector for calculation of the polar coordinates
21 theta = 0:1/grid_size:2*pi;
22 for i = 1:length(rho)
23     % translate the polar coordinates defined by radius value rho(i) to
24     % cartesian coordinates
25     [x,y] = pol2cart(theta,rho(i));
26     % round values so they can be used as integer
27     x = uint8(x+center(1));
28     y = uint8(y+center(2));
29     % create mean value for all (x,y) values in the matrix
30     mean_val = mean(img(sub2ind(size(img),x,y)));
31     % assign values to the profile at both sides from the vector center
           so
32     % the center of the matrix stays in the middle
33     profile(round(end/2)+(i-1)) = mean_val;
34     profile(round(end/2)-(i-1)) = mean_val;
35 end
36 end
```

Instead of iterating through the radius values another possibility is to iterate through the angle values $\rho$.

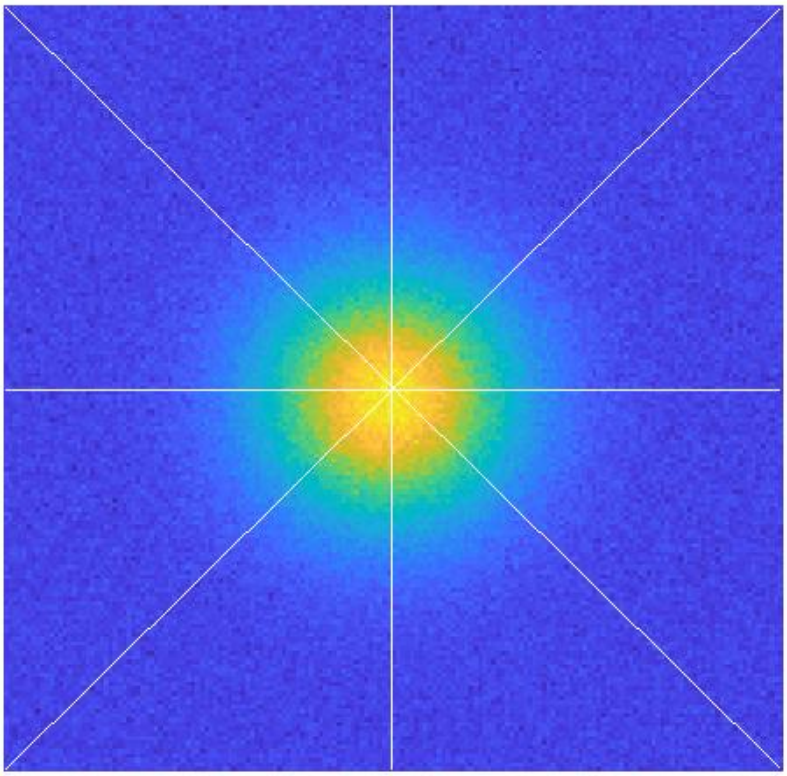


Figure 11: Gaussian Distributed Image with Radial Lines

In Figure 11 the profiles along the intersecting lines are averaged so the rotation value of the line along which the profile is created increases in each iteration. Again the the length of the radial profile is $\frac{l}{2}$ and the amount of profile lines depends on the sample rate.
In the following `MATLAB` implementation the radial profiles are averaged.

```
function [profile,grid] = concentrical_sum_v1(img,center)
% input: img - data matrix with gaussian distribution, center -
    gaussian
% distribution center coordinates
% output: profile - vector with concentrical averaged values, grid -
    grid vector
% for counteracting size variation

% determine img dimensions and smallest distance from the center to an
% img border
[width,height] = size(img);
radius = min([center(1) width-center(1) center(2) height-center(2)]);
% define stepsize based on precision of the center
step_size=1-mod(radius,1);
% create grid vector and corresponding value vector
```

```
grid=-radius:step_size:radius;
grid_size=length(grid);
profile = zeros(1,grid_size);
% define radius vector which stepwise increases its distance to the img
% center
rho = 0:step_size:radius;
% define angle vector for calculation of the polar coordinates
theta = 0:1/grid_size:2*pi;
for i = 1:length(theta)
    % translate the polar coordinates defined by radius value theta(i)
       to
    % cartesian coordinates
    [x,y] = pol2cart(theta(i),rho);
    % round values so they can be used as integer
    x = uint8(x+center(1));
    y = uint8(y+center(2));
    % gather values for (x,y) in the matrix
    mean_val = img(sub2ind(size(img),x,y));
    % create symmetrical profile
    profile(1:round(end/2)) = (fliplr(mean_val)+profile(1:round(end/2))
       );
    profile(round(end/2)+1:end) = (mean_val(2:end)+profile(round(end/2)
       +1:end));
end
% devide by tha amount of sampled values to create the average
profile=profile/length(theta);
end
```

As mentioned before this method of profile creation holds a numerical error in the definition of $EQ_{r_i}$ with $\vec{x}$ being the closest Cartesian integer coordinate to the exact polar coordinate value. As a further disadvantage of this method the average of inner circles has less points included in the calculation, so it is less resistant to noise.

### 3.4.2 Radial Distribution

Besides creating the average of concentric circles around the center it is possible to iterate through each image point and map its value $\overrightarrow{data}_{x,y}$ to its distance from the center point $\vec{c}$. This results in a vector $\vec{I}$ containing the intensity $\overrightarrow{data}_{x,y}$ and a vector $\vec{d}$ containing the distance to $\vec{c}$.

$$\vec{d} = \begin{pmatrix} \left| \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \vec{c} \right| \\ \vdots \\ \left| \begin{pmatrix} w \\ h \end{pmatrix} - \vec{c} \right| \end{pmatrix} \in \mathbb{R}^{w \cdot h}, \vec{I} = \begin{pmatrix} \overrightarrow{data}_{1,1} \\ \vdots \\ \overrightarrow{data}_{w,h} \end{pmatrix} \in \mathbb{R}^{w \cdot h} \tag{3.4.5}$$

As discussed in 3.4.1 the amount of points to be included in the calculation is limited by the distance $l$ as described in (3.4.4). Thus the size of the vectors $\vec{I}$ and $\vec{d}$ can be reduced from $w \cdot h$ to $\frac{l}{2}^2$ which

is the circle of points around $\vec{c}$ with radius $r = \frac{l}{2}$ for which its inverse point (3.4.1) is part of the image. When excluding points further than $r$ away from $\vec{c}$ the vectors $\vec{d}$ and $\vec{I}$ are reduced to

$$\vec{d} = \begin{pmatrix} \left| \begin{pmatrix} c_1 \\ c_2 - r \end{pmatrix} - \vec{c} \right| \\ \vdots \\ \left| \begin{pmatrix} c_1 \\ c_2 + r \end{pmatrix} - \vec{c} \right| \end{pmatrix} \in \mathbb{R}^{r^2}, \vec{I} = \begin{pmatrix} \overrightarrow{data}_{c_1, c_2 - r} \\ \vdots \\ \overrightarrow{data}_{c_1, c_2 + r} \end{pmatrix} \in \mathbb{R}^{r^2} \tag{3.4.6}$$

The values in $\vec{d}$ are not sorted yet so in order to guarantee that the values at the top of the vector are closest to the center $\vec{d}$ and $\vec{I}$ have to be sorted. The order in which the vectors are sorted has to be identical so that $I_i$ still contains the intensity value for a point at distance $d_i$. The algorithm for the creation of the sorted vectors `dist` and `val` is described by the following steps

1. Create vectors `dist` and `val` of size $r^2$ and assign $\texttt{iter} = 1, \texttt{count} = 1$.
2. Assign $d = |\begin{pmatrix} i \\ j \end{pmatrix} - \vec{c}|$ with $i = \texttt{iter} \mod w$ and $j = \lceil \texttt{iter} \div w \rceil$.
3. If $d < r$ set $\texttt{dist}(\texttt{count}) = d$ and $\texttt{val}(\texttt{count}) = data_{i,j}$, increase `count` and `iter` by one. Then repeat from 2.
4. If $\texttt{iter} < w \cdot h$ increase and `iter` by one. Then repeat from 2.
5. Sort the values in `dist` by increasing order and create a permutation matrix [6][3.5,p. 73] which indicates the new position of each element in the vector.
6. Sort the values in `val` by using the created permutation matrix.

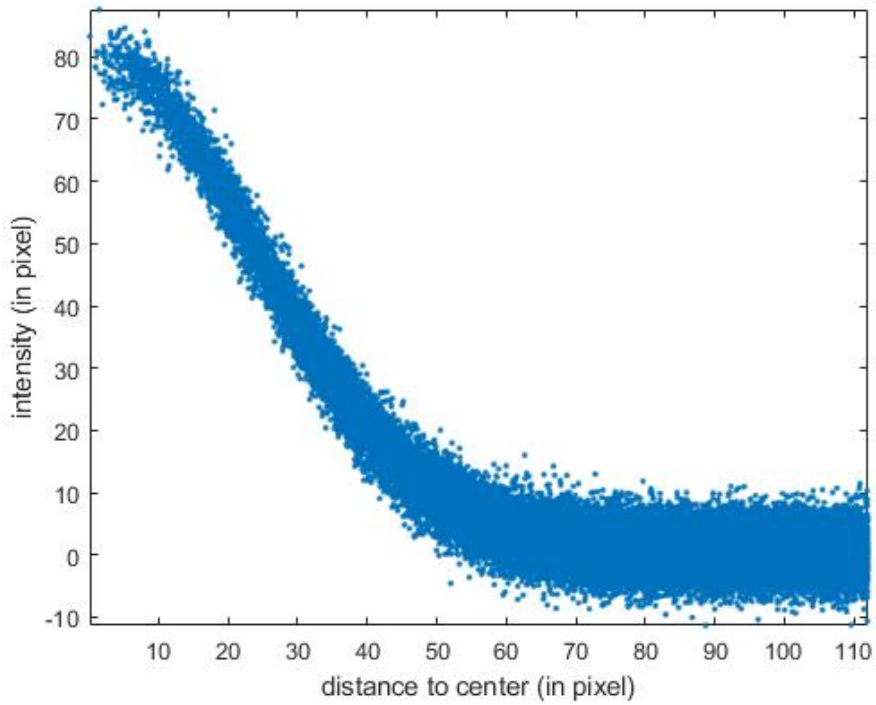



Figure 12: Distribution of values across the distance from the center

Figure 12 is a plot of the intensity distribution for increasing distance from the center. As seen in the depiction the amount of data points per pixel increases with the distance to $\vec{c}$.

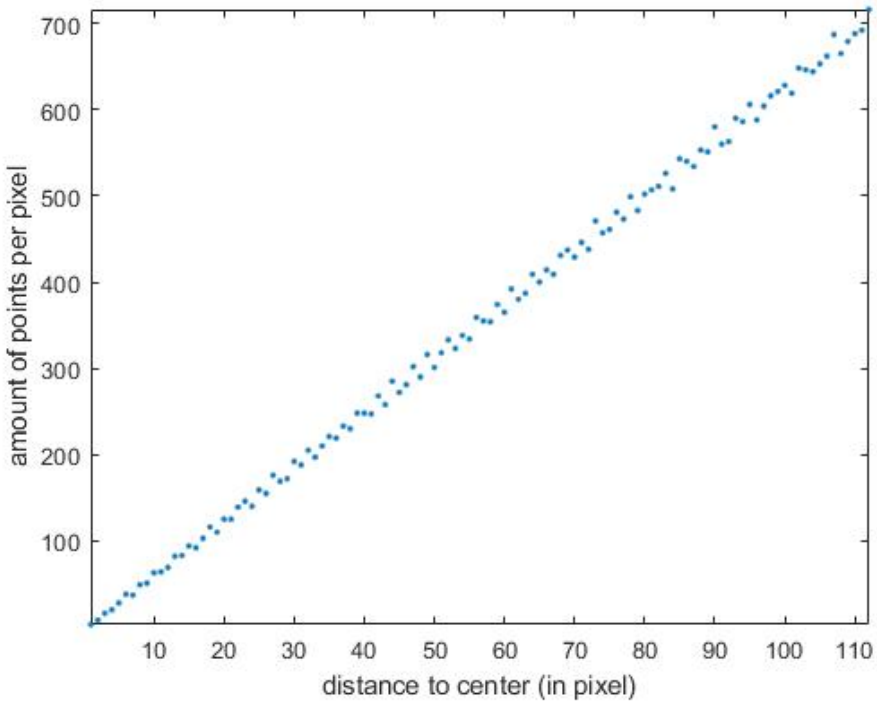



Figure 13: Amount of data points with distance from the center

Figure 13 shows the amount of data samples in the distance of one pixel. This linear increase in the data point amount is caused by the increasing perimeters of the circles with distance to $\vec{c}$ as depicted in Figure 10. The amount peaks at approximately 700 sample points per pixel for the furthest distance and starts at about 5 for the closest. Though generally all information for fitting the data is provided in the vectors $\vec{d}$ and $\vec{I}$ this difference of the amounts of data points results in the pixels furthest from the center dominating the calculation. Since the values close to the center hold the most valuable information for the calculation this difference of data amounts has to be reduced.
A method for this is to cluster the values and assign a representative value for each cluster. The clustering of the values is performed by a binning procedure, where each data point is assigned to the cluster closest to its distance value. The distance value of the cluster is the average distance of all points in the cluster and the intensity value is the average of all intensity values. This results in a matrix $\overrightarrow{C}$ where the first column contains the distance values and the second contains the intensity values of the clusters.

$$k = \begin{pmatrix} 1 \cdot \frac{r}{m} \\ \vdots \\ m \cdot \frac{r}{m} \end{pmatrix} \in \mathbb{R}^m,$$

$$\overrightarrow{C} = \begin{pmatrix} \dfrac{\sum_{i=1}^{k_1} d_i}{k_1} & \dfrac{\sum_{i=1}^{k_1} I_i}{k_1} \\ \vdots & \vdots \\ \dfrac{\sum_{i=k_{m-1}+1}^{k_m} d_i}{k_m - (k_{m-1}+1)} & \dfrac{\sum_{i=k_{m-1}+1}^{k_m} I_i}{k_m - (k_{m-1}+1)} \end{pmatrix} \in \mathbb{R}^{m \times 2} \tag{3.4.7}$$

This approach of clustering all data by fixed bin sizes combined with the idea that valuable information is located closer to the center $\vec{c}$ leads to the method of thresholded binning. The threshold can either be applied to the distance value or to the value distribution where the points in the same pixel are clustered if the cluster holds more than `thresh` points. This way, information at the top of the distribution is not varied and the distant points of the data do not dominate the calculation. The following algorithm describes the thresholded binning procedure with previously created vectors `dist` and `val`.

1. Count the amount of data points for each bin $[0,1), [1,2), \ldots, [m-1, m]$ and save it in a vector `dist_hist`.
2. Find the first bin number $k$ containing at least `thresh` data points by iterating through `dist_hist`.
3. Save the first $\sum_{i=1}^{k} \texttt{dist\_hist}_i$ points of `dist` and `val` in the value vector `profile` and distance vector `grid`.
4. Find the indexes `ind` of the entries for which $k \leq \texttt{dist} < k+1$.
5. Calculate the average of `dist`(`ind`) and save it in `grid`.
6. Calculate the average of `val`(`ind`) and save it in `profile`.
7. If $k < m$ increase $k$ by one and repeat from 4.

The `grid` and `profile` vectors are a combination of the raw distance intensity mapping at the beginning and the binned values further away from the center. This thresholding actually leaves the possibility to cluster almost every data point or none at all though that would decrease performance, since with no clustering the vectors contain the same amount of sample points as the image.

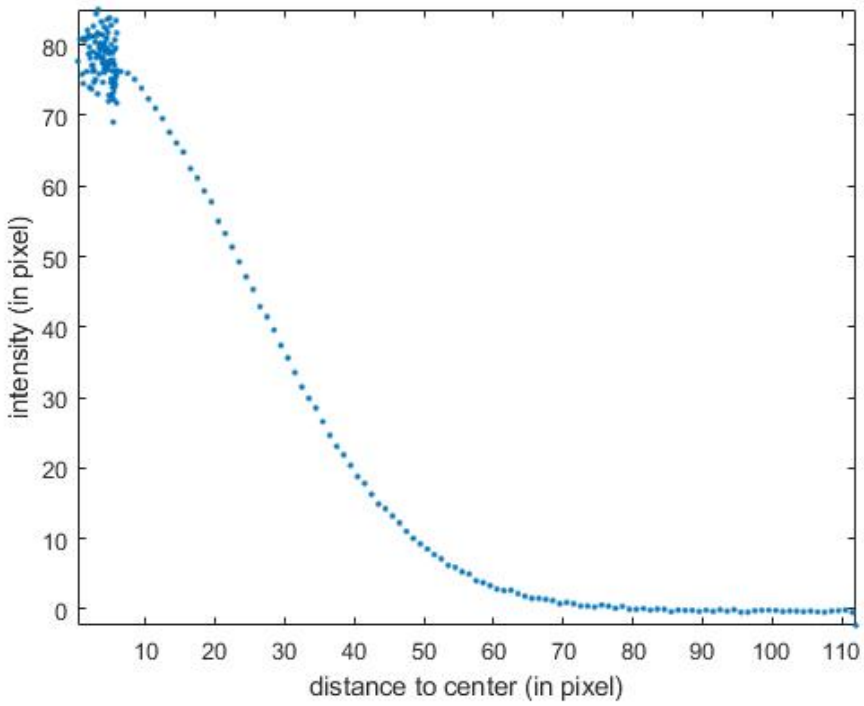



Figure 14: Binned distribution of values across the distance from the center

Figure 14 depicts the distribution seen in Figure 12 with applied thresholded binning for a threshold value of 40. As shown in Figure 14 the data close to the center of the distribution is preserved and can be fitted accordingly. The MATLAB implementation follows.

```
function [profile,grid] = distributed_profile(matrix,center,threshold)
% input: matrix - gaussian distributed data matrix, center -
    distribution
% center, treshold
% output: profile - value vector, grid - underlying grid

% optional default value for threshold
if isempty('threshold') || ~exist('threshold','var')
     threshold_val=20;
else
     threshold_val=threshold;
end
% determine matrix size and the minimal distance from the center to an
    edge
[w,h] = size(matrix);
min_dist = round(min([center(1) w-center(1) center(2) h-center(2)]));
% initialize the vectors
values = zeros(min_dist^2,1);
dist = zeros(min_dist^2,1);
dist_hist = zeros(min_dist,1);
count=1;
```

```
for i=1:h
    for j=1:w
        d=norm([i,j]-center);
        % check weather the index coorinate is in min_dist range from
            the
        % center
        if(d<min_dist)
            % store its distance and its value
            dist(count)=d;
            values(count)=matrix(i,j);
            % increase the amount of points in the bin
            dist_hist(floor(d)+1)=dist_hist(floor(d)+1)+1;
            count=count+1;
        end
    end
end
% sort the values in dist and apply the re-indexing to the values
[sortedKeys, sortedIndices] = sort(dist);
sortedValues = values(sortedIndices);
% determine the first bin containing at least threshold values
threshold_dist=length(find(dist_hist<=threshold_val));
% determine the amount of values
threshold_count=sum(dist_hist(1:threshold_dist));
% initialize the values and vectors
grid_length = threshold_count + min_dist - threshold_dist;
profile=zeros(grid_length,1);
grid=zeros(grid_length,1);
% store the first threshold_count values in the vectors
grid(1:threshold_count)=sortedKeys(1:threshold_count);
profile(1:threshold_count)=sortedValues(1:threshold_count);
for i=threshold_dist:min_dist
    % find indexes within the boundries and calculate the average of
        their
    % values
    bound_a=length(find(abs(sortedKeys<i)));
    bound_b=length(find(abs(sortedKeys<i+1)));
    val_avg=mean(sortedValues(bound_a:bound_b));
    key_avg=mean(sortedKeys(bound_a:bound_b));
    profile(threshold_count+i-threshold_dist)=val_avg;
    grid(threshold_count+i-threshold_dist)=key_avg;
end
end
```

When compared to `concentrical_sum` from 3.4.1 the method runs more stable with regards non integer values for the center, since it does not require to calculate polar coordinates. Also since in `concentrical_sum` polar coordinates have to be calculated, the time that is needed to calculate the profile using `concentrical_sum` is greater than the time `distributed_profile` takes. Nevertheless since both methods use some sort of averaging throughout the calculation both hold a numerical

error which alters the resulting parameter vector slightly.

### 3.5 Intensity Normalization

The created profiles represent the data in a reduced form and almost ready to be fitted. However the profiles may still contain one parameter which can be calculated without a fit. For default Gaussian distributed data

$$\lim_{x\to\infty} G(x) = 0$$

is expected. Due to the physical properties of the experiment $\lim_{x\to\infty} G(x)$ can actually be a constant $C$ with $c \neq 0$. In [2][1.4,p. 40] this is refered to as the base line of the distribution. In a statistical sense the distribution converges towards this value $C$ so it is the value which appears most often in the image. By creating a histogram of all values in the image, binning them into clusters and determining the bin with the most values the constant $C$ can be determined statistically without the need of fitting.

$$C = \max\left(\begin{pmatrix} |B_1| \\ \vdots \\ |B_b| \end{pmatrix}\right) \tag{3.5.1}$$

with $|B_i|$ being the amount of elements in bin $B_i$. This is also the case for the profile which consists of less data points and as long as the still enough points are not part of the distribution but already represent the value $C$ this statistical method will produce usable results. The following code shows an implementation in `MATLAB`.

```
function profile = level_norm(profile)
% input: profile - value vector
% output: profile - input vector shifted maximal appearing element is 0

% bin all values between two integers
[hist_count,hist_binval] = histcounts(profile,round(abs(min(profile)-
    max(profile))));
% deterimine bin with the most elements
[~, max_index] = max(hist_count);
zero = hist_binval(max_index);
% shift profile by its value
profile = profile - zero;
end
```

With the eliminated parameters $\vec{c}$ and $C$ the noise reduced image only contains a Gaussian distribution $G$ the specific parameters of which have to be detected using a fitting algorithm.

# 4 Fitting Parameters

This section deals with the fitting process of the preprocessed images. By fitting the gained data to a predefined function the parameters which best describe the current image series are calculated. In the following methods are presented which on the input of a Gaussian distributed vector or image return a vector which bests describes the distribution with help of parameters. These parameters can later be used to describe the overall process of the diffusion.

## 4.1 Single Component Analysis

After reducing the amount of parameters and the amount of data points the fitting process comes down to solving one formula with 2 parameters. Since all all gauss images now have their distribution center in the center of the image the formula, which describes the sum of the gauss function is

$$A \cdot \exp(\frac{-x^2}{2 \cdot \sigma^2}) \tag{4.1.1}$$

with $A$ and $\sigma$ being the parameters which have to be fitted, and $x$ being the grid vector for the corresponding dataset. The fit algorithm then uses (4.1.1) to minimize the least squares to the data

$$\sum_{k=1}^{N} \left( A \cdot \exp(\frac{-x_k^2}{2 \cdot \sigma^2}) - y_k \right)^2 \tag{4.1.2}$$

For minimizing (4.1.2) a two dimensional minimization problem solver is needed. Since the single component Gaussian function is basically just a special case of the multiple component Gaussian function, (4.1.1) can be described by (4.2.1) for $n = 1$. Hence the fitting algorithm of the multiple component Gaussian function is applicable to minimize (4.1.2) which is equal to (4.2.2) with $n = 1$. Therefore further algorithmic approaches are discussed in 4.2.

## 4.2 Multiple Component Analysis

In comparison to the two parameters which have to be fitted in case of single component data the multiple component Gaussian function adds two more parameters to the fitting process per component. So an overlay of multiple gauss functions can be described by a formula with $2n$ parameters where $n$ is the amount of substances involved in the diffusion process.The formula, which describes the sum of the gauss functions is

$$\sum_{i=1}^{n} A_i \cdot \exp(\frac{-x^2}{2 \cdot \sigma_i^2}) \tag{4.2.1}$$

with $A_i$ and $\sigma_i$ being the parameters which have to be fitted, and $x$ being the grid vector for the corresponding dataset. The fit algorithm then uses (4.2.1) to minimize the least squares to the data

$$\sum_{k=1}^{N} \left( \sum_{i=1}^{n} A_i \cdot \exp(\frac{-x_k^2}{2 \cdot \sigma_i^2}) - y_k \right)^2 \tag{4.2.2}$$

for all parameters $A_i$ and $\sigma_i$ and all $x_k$ and $y_k$. Hereby $N$ is the amount of data points in the profile, $(x, y) \in \mathbb{N}^N \times \mathbb{N}^N$ the vector, which contains all data points, $A \in \mathbb{N}^n$ the vector, which contains all amplitudes of the individual gauss functions and $\sigma \in \mathbb{N}^n$ their corresponding sigma values. The problem of minimizing (4.2.2) can be solved, by using a multidimensional minimization

algorithm. Therefore $A$ and $\sigma$ are compressed to a single vector $\vec{p}$ with the odd indexes containing the amplitudes and the even ones the sigma values.

$$\vec{p} = \begin{pmatrix} A_1 \\ \sigma_1 \\ \vdots \\ A_n \\ \sigma_n \end{pmatrix} \in \mathbb{R}^{2n} \tag{4.2.3}$$

Now (4.2.2) can be written as a function $f$ that calculates the least squares to a given data set for a sum of multiple gauss function, which are defined by a parameter vector $\vec{p}$.

$$f_{\vec{x},\vec{y}} : \mathbb{R}^{2n} \to \mathbb{R} : f_{\vec{x},\vec{y}}(\vec{p}) = \sum_{k=1}^{N} (\sum_{i=1}^{n} p_{2i-1} \cdot \exp(\frac{-x_k^2}{2 \cdot p_{2i}^2}) - y_k)^2 \tag{4.2.4}$$

This function represents the quality of our parameter vector. By evaluating the function $f_{x,y}(\vec{p})$ for certain values in $\vec{p}$ and comparing them to others, the result of (4.2.4) determines whether certain values fit to the dataset better then others. This is necessary since no analytical result can be calculated thus numerical methods have to be used to determine the best values of $\vec{p}$, for which (4.2.4) is minimal.

### 4.2.1 Powell's Method for Multidimensional Minimization

For minimizing functions with multidimensional input parameter, as seen in (4.2.4) there multiple numerical options. The following implementation of a minimization algorithm is based on "Powell's Quadratically Convergent Method" [1, 10.7.2,p. 511]. As described in "Numerical Recipes" the idea of this method is to minimize along each of the $N$ basis vectors in each iteration and thereby approaching the optimal result stepwise. In the case presented by (4.2.4) the parameter vector $\vec{p}$, as defined in (4.2.3), has to be minimized along the basis vectors

$$\vec{e}_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \ldots, \vec{e}_{2n} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \in \mathbb{R}^{2n}$$

A native approach for this algorithm is to choose a fixed step size $\Delta$ and to minimize along each parameter while leaving the other fixated. After determining the optimal value for the current parameter in the current iteration, the value is assigned to the parameter and a minimization along the next parameter can be calculated. This way after each iteration the parameter values are closer to the optimal values then in the iteration before. If the parameters are not improving or the desired precision is reached, the calculation can be stopped and the values saved. Hence the calculation after each step produces a new parameter vector with help of a coefficient vector for $\Delta$. By definition (4.2.3) the parameter vector $\vec{p}$ in iteration $k+1$ is

$$(\vec{p})^{k+1} = (\vec{p})^k + \Delta \cdot (s_1 a_1 \vec{e}_1 + \cdots + s_{2n} a_{2n} \vec{e}_{2n}) \tag{4.2.5}$$

Hereby $\vec{s} \in \mathbb{Z}^{2n}$ represents the signum function

$$sign(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

for the direction in which the next optimal value for $\vec{p}_i$ is expected and $\vec{a} \in \mathbb{N}^{2n}$ the amount of iterations of size $\Delta$ in the direction of that optimal value from the current value. Consequently for the entry $i$ in $(\vec{p})^k$ the vectors $\vec{s}$ and $\vec{a}$ can by calculated by

$$s_i = \begin{cases} 1 & \text{if } p_i^{k+1} > p_i^k \\ 0 & \text{if } p_i^{k+1} = p_i^k \\ -1 & \text{if } p_i^{k+1} < p_i^k \end{cases}$$

$$a_i = \frac{|p_i^{k+1} - p_i^k|}{\Delta}$$

Since in iteration step $k$ the values of $(\vec{p})^{k+1}$ are unknown, $\vec{s}$ and $\vec{a}$ have to be calculated iteratively as well. Thus $\vec{s}$ and $\vec{a}$ have to be calculated without using $(\vec{p})^{k+1}$. For the computation of $\vec{s}$ the evaluation function $f_{x,y}(\vec{p})$ is utilized as defined in (4.2.4). For the following method the assumption that the direction in which $f_{x,y}(\vec{p})$ gets smaller is the direction in which $(\vec{p})_i^k$ is moved towards its optimal value is used. This assumption is not necessarily true, since $(\vec{p})^k$ might be close to a local minimum that leads it away from the global minimum which would be the optimal value for the current iteration.

### 4.2.2 Local and Global Minima

In general minima are points $\vec{x}_{min} \in \mathbb{R}^n$ of a function $f(\vec{x}) : \mathbb{R}^n \to \mathbb{R}^m (m, n \in \mathbb{N})$ for which the value of $f(\vec{x}_m)$ is smaller then the ones of all their neighbors. The consequent idea of (4.2.5) is to move along the declining direction of a function until its values stop declining. If $x_{min} \neq \pm\infty$ this procedure will always result in mimima. Nevertheless there might be multiple minima which are local minima since $f(\vec{x})$ is smallest for their local neighborhood, but there can be only one global minimum, which is the point $\vec{x}_{min_{global}}$ for which $f(\vec{x}_{min_{global}})$ is smaller then for all the others.

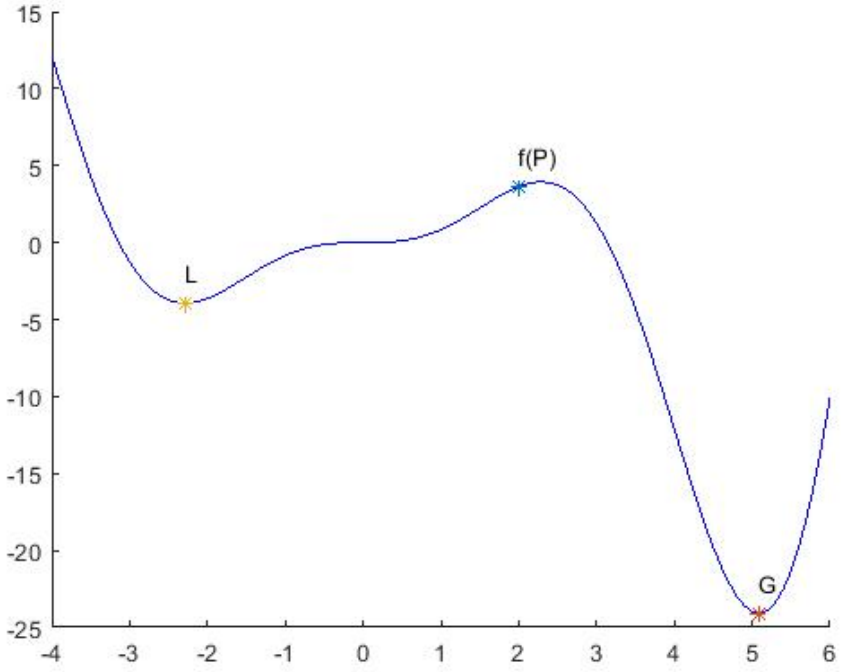



Figure 15: Local Minimum Problem

Figure 15 shows a possible scenario for a problem to occur. In order to find the global minimum $G$ of the function, the point $P$ would have to move right, what as a result would increase the function value $f(P)$. But evaluation of the points to the left and to the right of $P$ indicate, that $P$ will reach a minimum by moving to the left. Thus method (4.2.5) will only be able to reach the local minimum $L$ when starting from point $P$. When Searching the global minimum without prior approximation every local minimum has to be checked, before the global minimum can be determined. But in the case of the function defined in (4.2.2) prior approximation of the parameter vector $p$ as defined in (4.2.3) is possible.
Firstly the amplitudes of the individual gauss functions have are bound to the maximal point of the dataset.

$$\sum_{i=1}^{n} p_{2i-1} = \max(y) \tag{4.2.6}$$

With no further restriction the starting values for the amplitudes of each gauss function can be set to

$$p_{2i-1} = \frac{\max(y)}{n}, \forall i \in \{1, \ldots, n\} \tag{4.2.7}$$

Although the individual amplitude values are expected not to be near their optimal values, (4.2.7) provides a good starting point, from which individual amplitudes can move in certain directions. At the same time the physical properties of the gauss function restricts them from being negative and (4.2.6) shows their dependency from another. The interval for possible values is therefore defined by

$$0 < p_{2j-1} < \max(y) - \sum_{\substack{i=1 \\ i \neq j}}^{n} p_{2i-1}, \forall j \in \{1, \ldots, n\} \tag{4.2.8}$$

This means, that if an amplitude of a gauss function increases, another one will decrease.

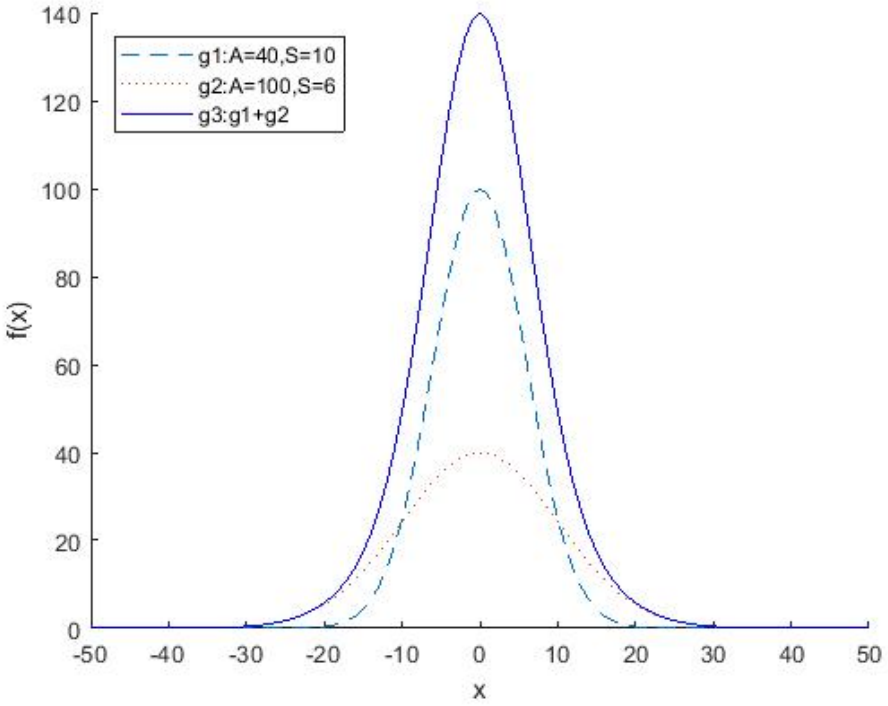



Figure 16: Amplitude Dependency of a Gauss Sum

Figure 16 depicts this dependency since $\max(g3)$ is equal to the sum of the amplitudes of $g1$ and $g2$.
In contrast to the individual amplitudes of the gauss functions their variances do not have any linear dependency from one another. Since a start value has to be provided a simple approach is to treat the sum of the gauss functions (4.2.1) as an own gauss function, calculate its variance and assign it to all variance values. By definition (taken from Wikipedia[4]) the variance of a gauss function $f(x)$ is

$$\sigma = x_w \cdot \sqrt{2 \ln 2}, f(x_w) = \frac{A}{2} \tag{4.2.9}$$

with $A$ being the amplitude and $x_w$ the half width at half maximum. For the given discrete values in the dataset the exact value for $x_w$ might not be included but since (4.2.9) serves as an approximation the $x$ value, which is nearest to $x_w$ can be taken as well. Thus the best approximation $x_{appr}$ can be calculated by

$$x_{appr} = x_k : \min(|y_k - \frac{A}{2}|), \forall k \in \{1, \ldots, N\} \tag{4.2.10}$$

where $N$ is the amount of points in the dataset, $y_k$ being the data values and $x_k$ being their corresponding grid values. The remaining starting values of the parameter vector $p$ can therefore be assigned.

$$p_{2i} = x_{appr} \cdot \sqrt{2 \ln 2}, \forall i \in \{1, \ldots, n\} \tag{4.2.11}$$

[4]https://en.wikipedia.org/w/index.php?title=Full_width_at_half_maximum&oldid=864676968

After the parameter vector $p$ is initialized the iterative optimization process can almost be commenced. Still an appropriate step size $\Delta$ has to be chosen. Since $p$ contains the approximate average for each parameter a fast approach is to divide this average by some chosen integer which represents the desired precision. The following code shows a possible implementation in `MATLAB` of the method described above.

```
function p0 = powell_method(x,y,n,prec)
% input: x - grid for the values, y - value vector, n - substance
    amount, prec - desired precision
% output: p0 - parameter vector for n substances

global subst
global x_coor
global y_coor
subst=n;
x_coor=x;
y_coor=y;

% profile amplitude and variance for p0 initialization
amplitude = max(y);
variance = calc_variance_1D(y);
% p0 initial parameter guess
p0 = zeros(2*n,1);
for i=1:n
    p0(2*i-1) = amplitude/n;
    p0(2*i)= variance;
end
% set residual value for initial p0
residual = eval(p0);
old_residual = residual*2;
%iteration counters
iter = 0;
% maximal iterations in outer loop
ITMAX = 200;
% maximal iterations in inner loop
STEPMAX = 3000;
% stepsize towards optimal parameters
delta=p0/prec;
% iterate until results don't get better
while(old_residual>residual && iter<ITMAX)
    % take best value in each parameter direction
    for i=1:2*n
        p_up = p0;
        p_down = p0;
        p_up(i) = p0(i)+delta(i);
        p_down(i) = p0(i)-delta(i);
        % choose direction, in which to minimize
```

```
41          if(eval(p_up)<eval(p_down))
42              % in this case the eval function decreases by decreasing
                    values in p0(i)
43              local_iter=0;
44              % minimize until the next step worsens results
45              while(eval(p_up)<eval(p0) && local_iter<STEPMAX)
46                  p0(i)=p0(i)+delta(i);
47                  p_up(i)=p_up(i)+delta(i);
48                  local_iter=local_iter+1;
49              end
50              % after while-loop p0(i) contains best fitting value
51          else
52              % in this case the eval function decreases by increasing
                    values in p0(i)
53              local_iter=0;
54              while(eval(p_down)<eval(p0) && local_iter<STEPMAX)
55                  p0(i)=p0(i)-delta(i);
56                  p_down(i)=p_down(i)-delta(i);
57                  local_iter=local_iter+1;
58              end
59          end
60      end
61      % compare the previous residual to the current
62      old_residual = residual;
63      residual = eval(p0);
64      iter = iter + 1;
65  end
66  end
67
68  % the eval function returns the difference to the data-set of the
        function for a given parameter vector
69  function fun_val = eval(p)
70  global subst
71  global x_coor
72  global y_coor
73  val=0;
74  for i=1:subst
75      val = val+p(2*i-1)*exp(-x_coor.^2./(2*p(2*i)^2));
76  end
77  fun_val = sum((val-y_coor).^2);
78  end
79
80  % calculate variance from dataset
81  function variance = calc_variance_1D(values)
82  % input: values - gaussian distributet value vector
83  % output: variance value
84
85  % calculate amplitude, center is index of amplitude
```

```
[amplitude,center] = max(double(vector));
% find point closest to amplitude/2
[~,var_index] = min(abs(values-amplitude/2));
variance = abs(center-var_index)/sqrt(2*log(2));
end
```

This implementation will find parameters which will fit the function in `eval` with `n` gauss functions to a dataset `y` with grid `x` and desired relative step size `prec`.

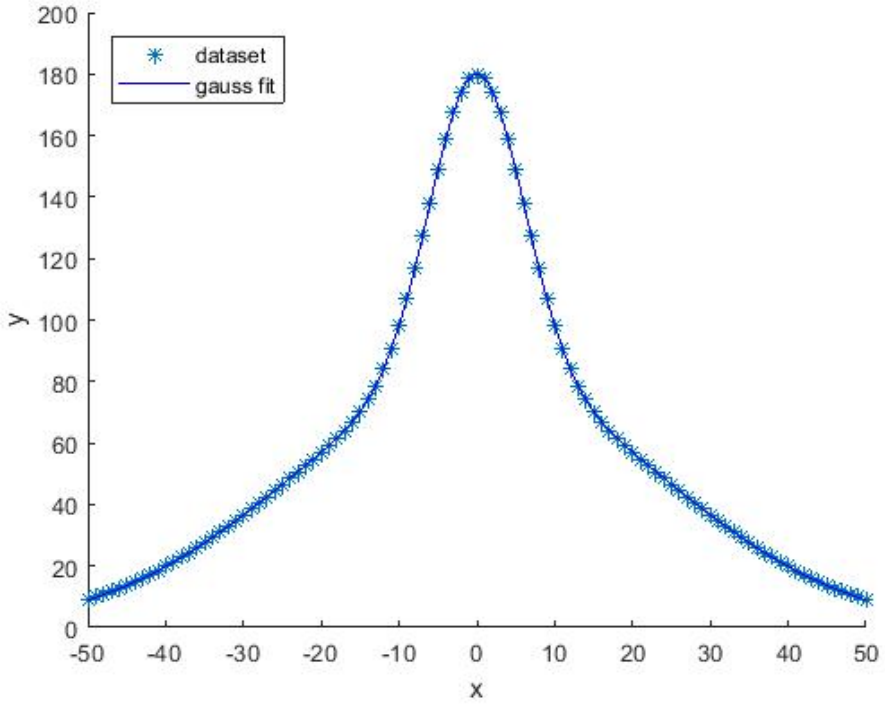



Figure 17: Gauss Fit using `powell_method()`

In the example, shown in Figure 17 the dataset contains a gauss sum described by following function

$$G(x) = A_1 \cdot \exp(\frac{-x^2}{2 \cdot \sigma_1^2}) + A_2 \cdot \exp(\frac{-x^2}{2 \cdot \sigma_2^2}), A_1 = 80, \sigma_1 = 24, A_2 = 100, \sigma_2 = 6$$

For input parameter `prec` $= 1000$ and `n` $= 2$ the result vector `p0` of `powell_method()` is

$$\texttt{p0} = \begin{pmatrix} 80.5500 \\ 23.8982 \\ 99.6300 \\ 5.9605 \end{pmatrix}$$

with $f_{x,y}(\texttt{p0})$ as defined in (4.2.4)

$$f_{x,y}(\texttt{p0}) = 1.3130$$

being reached after `iter` $= 33$ iterations.
Hereby the amount of iterations has to be differentiated. Although the outer loop has been repeated

32 times, the sum of `local_iter` for every outer loop iteration sums up to a total of 564 iterations. When comparing the real values of $G(x)$ the entries in `p0` are off by maximal 0.5 so algorithm approximates $G(x)$, at least by its form. Even though this approach for fitting `p0` produces usable results, the expectation is that for noiseless data as produced by $G(x)$ it should be possible to approximate the real parameters in $G(x)$ much more precise. The step size defines the precision and the step size is dependent from the method parameter `prec`. So the precision should increase by increasing `prec`. After running `powell_method` for increasing parameters `prec` following results can be observed.

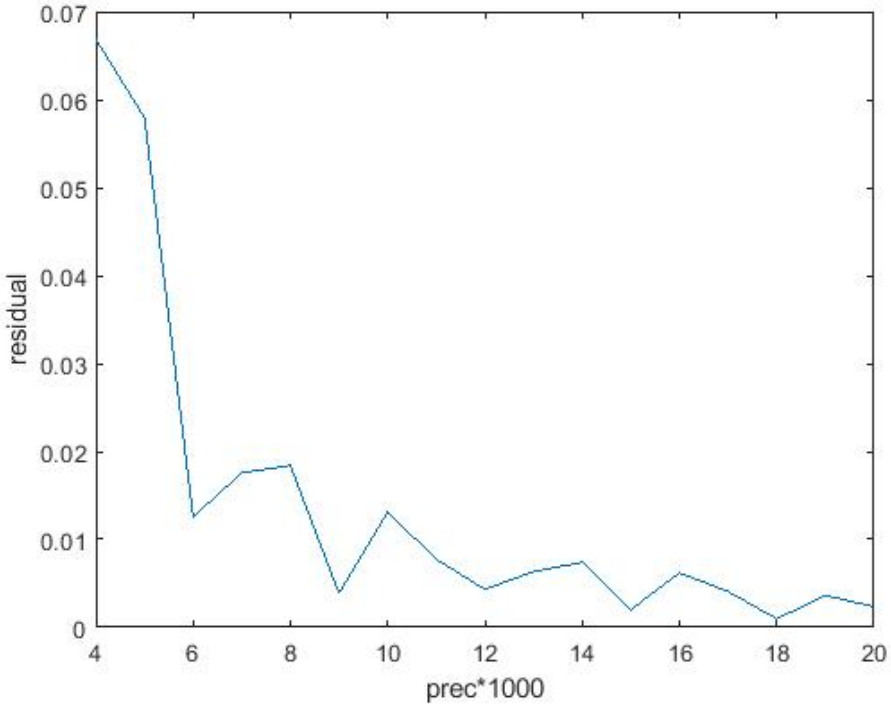



Figure 18: `powell_method()` Residual for Increasing `prec`

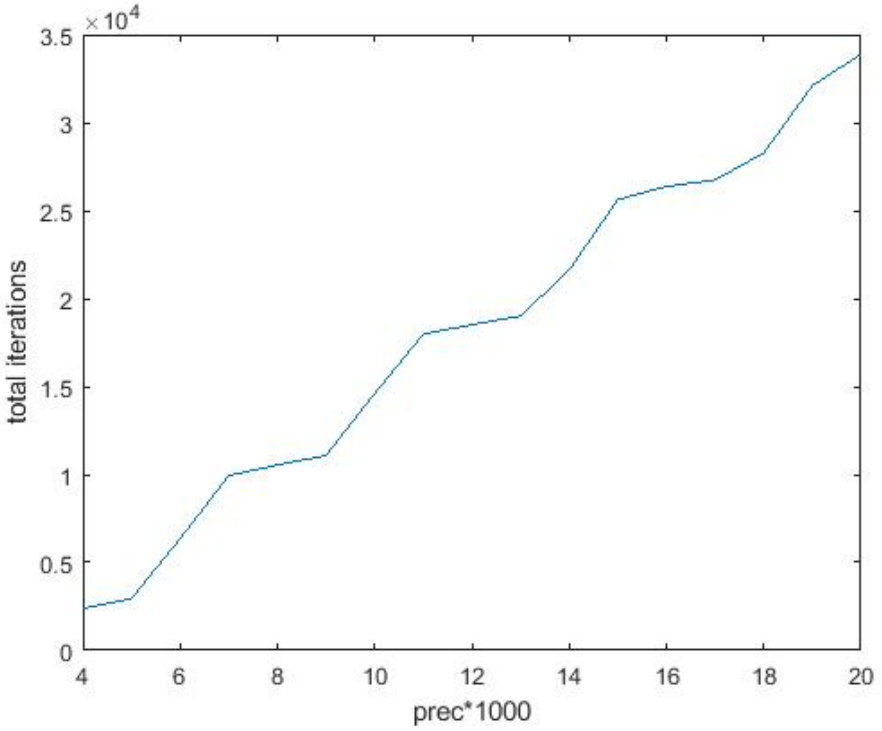



Figure 19: `powell_method()` Iteration Number for Increasing `prec`

While the number of iterations per run increase exponentially when `prec` is increased, the residual function $f_{x,y}(\texttt{p0})$ decreases quadratically. This relation can be explained by looking at the purpose of `prec`. Since $\Delta$ is defined by

$$\Delta = \frac{\texttt{p0}}{\texttt{prec}}$$

with `p0` as the initial guess of the parameter vector $\vec{p}$ (4.2.3), `prec` decreases the step size, but the step size is static and independent from the distance to the optimal parameter value. Hence the usage of `powell_method` results in an unbalanced trade off between precision and computing speed. The solution to this problem calls for an approach, where the step size changes dynamically as the values get closer to the minimum.

### 4.2.3 Bracketing the Minimum

The motivation of dynamic step size is to avoid having to iterate in small steps over great distances and at the same time to avoid losing precision when approaching the minimum. Multiple implementations of dynamically adjusting the step size are possible. It is possible to decrease the step size with each iteration but since simply moving the value towards the optimum provides no information about the distance from it, this implementation can face two kinds of problems. Either the decrease starts to early or is to drastically which results in the optimum not being reached at all, or the decrease starts to late which results in the loss of precision.
Another approach is to bracket the minimum [1, 10.1,p. 490] of the evaluation function for a parameter value. This approach asserts that in each iteration the value *opt* which minimizes $f_{x,y}(p)$ (4.2.4) best has a lower bound $a$ and an upper bound $b$.

$$\forall i \in 1, \ldots, 2n : opt_i \in [a_i, b_i] \land f_{x,y}(opt_i) \leq f_{x,y}(a_i) \land f_{x,y}(opt_i) \leq f_{x,y}(b_i) \tag{4.2.12}$$

Values $a$ and $b$ serve as brackets for the searched value.After setting initial values for $a$ and $b$ the idea is to move them towards each other while always maintaining assertion (4.2.12). The algorithm then proceeds to execute the following steps

1. Bisect the interval $a$ and $b$, so $mid = \frac{b-a}{2}$
2. Evaluate $f_{x,y}(p)$ for neighbors of $mid$
3. Save $mid$ to $\begin{cases} a & \text{if } opt_i > mid \\ b & \text{if } opt_i < mid \end{cases}$
4. If $b - a > \Delta$ repeat

Throughout the algorithm assertion (4.2.12) can only be breached when by chance $mid$ is assigned a value on the slope towards a local minimum. Even if this should be the case, $mid$ might be assigned to the right variable, if it is on the right side of the slope. In total the probability of being trapped in a local minimum is minimized compared to iteration by fixed step size because the function is evaluated in a not continues manner.
In this context $\Delta$ is not the iteration size but serves as measurement for the guarantee, that the approximated minimum is within the distance $\Delta$ to the result. The initial values for $a$ and $b$ still define the efficiency of the algorithm. But compared to the previous approach great distances can be traversed with exponential speed, so if one of the bounds is off by a lot[5] the algorithm will resize the bracketed interval accordingly searching a wider range for possible minima to occur.
In the given case boundary $a$ is defined by physical properties, neither the amplitude nor the variance of a gauss function can be negative. Therefor initial $a$ is set to 0. For the initial value of $b$ the boundaries are unclear, but since the algorithm can handle great distances without losing precision the values are simply set to the quadratic value of the corresponding initial value of $\vec{p}$.
The following `MATLAB` implementation includes the bracketing algorithm into the fitting method.

```
function p0 = powell_method_v1(x,y,n,prec)
% input: x - grid for the values, y - value vector, n - substance
    amount, prec - desired precision
% output: p0 - parameter vector for n substances

global subst
global x_coor
global y_coor
subst=n;
x_coor=x;
y_coor=y;

% profile amplitude and variance for p0 initialization
amplitude = max(y);
variance = calc_variance_1D(y);
% p0 initial parameter guess
p0 = zeros(2*n,1);
for i=1:n
```

[5]Since bounding values might have to be guessed, it is sometimes better to choose them to be further apart than necessary to grant (4.2.12).

```
18      p0(2*i-1) = amplitude/n;
19      p0(2*i)= variance;
20  end
21  % set residual value for initial p0
22  residual = eval(p0);
23  old_residual = residual*2;
24  %iteration counters
25  iter = 0;
26  % maximal iterations in outer loop
27  ITMAX = 200;
28  % maximal iterations in inner loop
29  STEPMAX = 3000;
30  % stepsize towards optimal parameters
31  delta=p0/prec;
32  % iterate until results don't get better
33  while(old_residual>residual && iter<ITMAX)
34      % take best value in each parameter direction
35      for i=1:2*n
36          p_up = p0;
37          p_down = p0;
38          p_up(i) = p0(i)+delta(i);
39          p_down(i) = p0(i)-delta(i);
40          % choose direction, in which to minimize
41          if(eval(p_up)<eval(p_down))
42              % in this case the eval function decreases by decreasing
                    values in p0(i)
43              local_iter=0;
44              bracket_a = p_up(i);
45              bracket_b = p_up(i)^2;
46              p_down = p_up;
47              % minimize until b-a = prec
48              while((abs(bracket_a-bracket_b) > prec) && local_iter<
                    STEPMAX)
49                  % bisect interval [a,b]
50                  mid_p = (bracket_a+bracket_b)/2;
51                  p_up(i) = mid_p+prec;
52                  p_down(i) = mid_p-prec;
53                  % check in which direction the minimum is and assign
                        values accordingly
54                  if(eval(p_up)<eval(p_down))
55                      bracket_a = mid_p+prec;
56                  else
57                      bracket_b = mid_p-prec;
58                  end
59                  local_iter=local_iter+1;
60              end
61          else
62              local_iter=0;
```

```
63                bracket_a = 0;
64                bracket_b = p_down(i);
65                p_up = p_down;
66                %minimize in the other direction
67                while((abs(bracket_a-bracket_b) > prec) && local_iter<
                      STEPMAX)
68                    mid_p = (bracket_a+bracket_b)/2;
69                    p_up(i) = mid_p+prec;
70                    p_down(i) = mid_p-prec;
71                    if(eval(p_up)<eval(p_down))
72                        bracket_a = mid_p+prec;
73                    else
74                        bracket_b = mid_p-prec;
75                    end
76                    local_iter=local_iter+1;
77                end
78            end
79            %after iteration p0(i) has last minimal value
80            p0(i)=mid_p;
81        end
82        % compare the previous residual to the current
83        old_residual = residual;
84        residual = eval(p0);
85        iter = iter + 1;
86    end
87    end
88
89    % the eval function returns the difference to the data-set of the
          function for a given parameter vector
90    function fun_val = eval(p)
91    global subst
92    global x_coor
93    global y_coor
94    val=0;
95    for i=1:subst
96        val = val+p(2*i-1)*exp(-x_coor.^2./(2*p(2*i)^2));
97    end
98    fun_val = sum((val-y_coor).^2);
99    end
100
101   % calculate variance from dataset
102   function variance = calc_variance_1D(values)
103   % input: values - gaussian distributet value vector
104   % output: variance value
105
106   % calculate amplitude, center is index of amplitude
107   [amplitude,center] = max(double(vector));
108   % find point closest to amplitude/2
```

```
109 [~,var_index] = min(abs(values-amplitude/2));
110 variance = abs(center-var_index)/sqrt(2*log(2));
111 end
```

Again in order to count all iterations of the algorithm the inner as well as the outer loops have to be taken into account. The expectation for the `powel_method_v1` algorithm would be to exceed the previous version `powel_method` in terms of convergence speed, precision and the number of iterations.

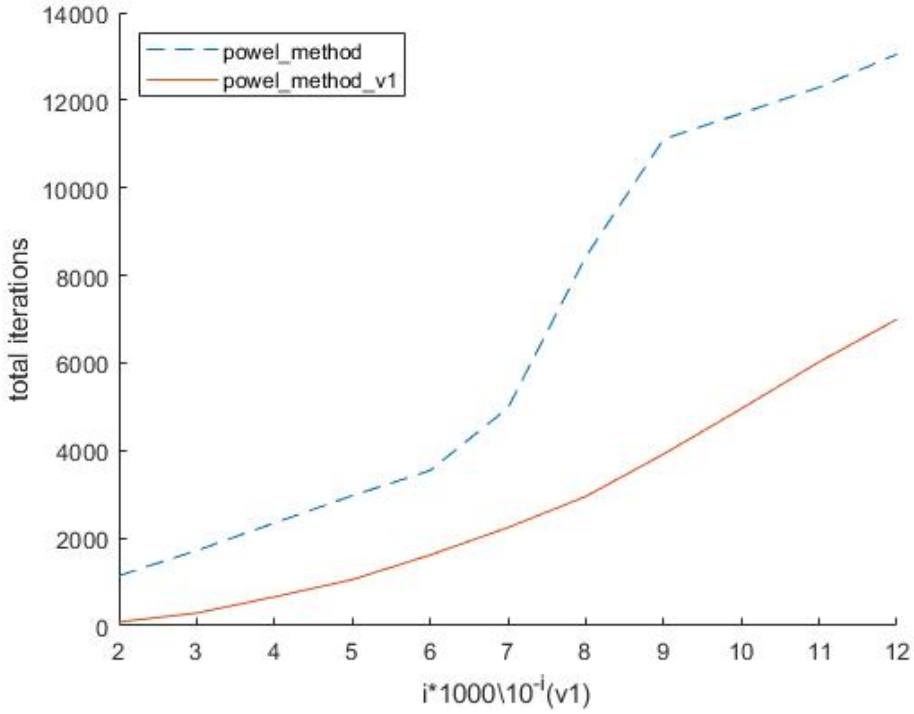



Figure 20: Number of Iterations of `powel_method_v1` compared to `powel_method`

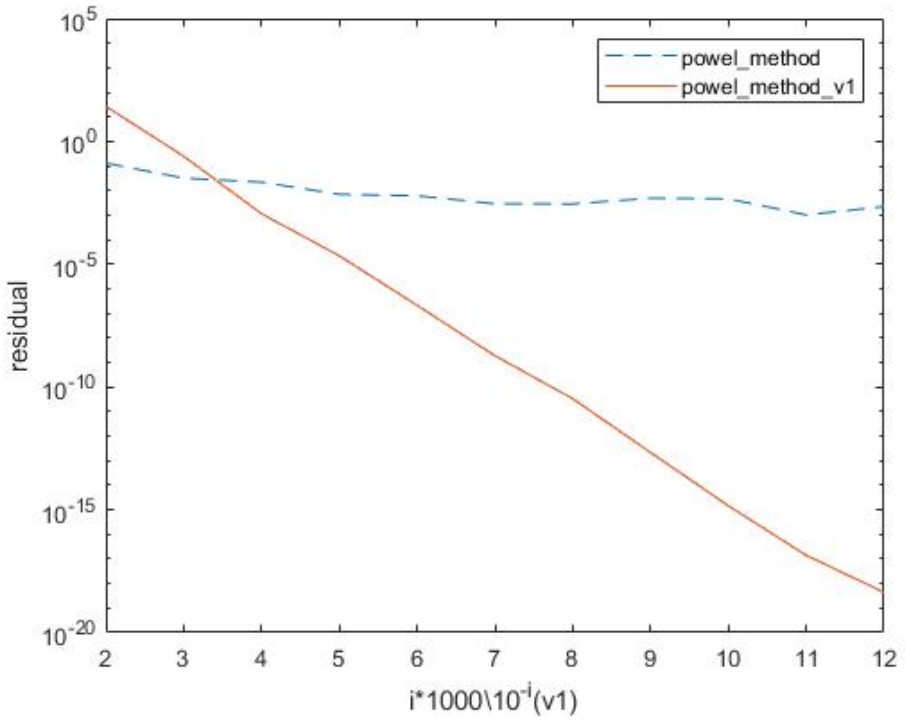



Figure 21: Residual of `powel_method_v1` compared to `powel_method`

Indeed as Figure 20 and Figure 21 show the convergence speed of `powel_method_v1` is exponential while its total number of iterations increases linearly. The algorithms were tested by fitting (4.2.1) to a dataset, which was generated by the function

$$G(x) = A_1 \cdot \exp(\frac{-x^2}{2 \cdot \sigma_1^2}) + A_2 \cdot \exp(\frac{-x^2}{2 \cdot \sigma_2^2}), A_1 = 130, \sigma_1 = 1.534534, A_2 = 100, \sigma_2 = 6$$

and iteratively decreasing the step size for `powel_method` by 1000 and iteratively increasing the targeted precision of `powel_method_v1` by factor 10 in each iteration. For a target precision `prec` of $10^{-12}$ the values of `p0` in `powel_method_v1` are optimized to

$$\texttt{p0} = \begin{pmatrix} 100,000000000574 \\ 5,99999999998222 \\ 129,999999999591 \\ 1,53453399999472 \end{pmatrix}$$

after 6998 iterations with a residual of $4,366351737663584 \cdot 10^{-19}$.
For the step size quotient `prec` $= 12000$ `powel_method` returned the parameter vector

$$\texttt{p0} = \begin{pmatrix} 100,040416666658 \\ 5,99875987543540 \\ 129,978750000006 \\ 1,53387517131983 \end{pmatrix}$$

after iterating 13050 times with a residual of $0,002210117787711$.
As a conclusion bracketing the minimum enhances the performance of the algorithm presented in 4.2.2 drastically.

### 4.2.4 Minimization with `fminsearch`

As an additional method for fitting the parameter vector `p0` from (4.2.3) `MATLAB` provides an own minimization algorithm. It takes a function `fun` and a starting value `x0` as input arguments and calculates the local minimum closest to `x0`. The argument `fun` must be passed as a symbolic function and since there is no symbolic equivalent for $\sum_{i=1}^{n}$ in `MATLAB`[6] a disadvantage of this method is, that the substance amount $n$ can not be parameterized. So the function would have to be hard coded for each $n$.
Alternatively a non symbolic function can be created which returns the value of the least squares from the data for a given `p0`. This `eval` function is then passed to `fminsearch` via a function handle, symbolized by `@` in `MATLAB`. Here the least squares function $f_{x,y}(\vec{p})$ as defined in (4.2.4) serves as the `eval` function. The input parameters needed for the fitting algorithm are

1. the underlying grid vector `x`
2. the value vector `y`
3. the amount of substances `n`.

Unlike for the `powel_method` no precision parameters can be passed to `fminsearch`.
`fminsearch` finds the parameter values which minimize the function values[7] by using the Nelder-Mead simplex algorithm[8].
The algorithm takes the initial parameter vector $\vec{p}_0 \in \mathbb{R}^n$ and creates a new parameter vector by increasing the the value by 5% for each of the $n$ dimensions. When including the initial $\vec{p}_0$ this results in $n+1$ vectors $\vec{p}_0, \ldots, \vec{p}_n$ which create a so called simplex $S$. Then following steps are then repeated

1. Sort all vectors $\vec{p}_0, \ldots, \vec{p}_n$ in the simplex by their evaluated value $f_0 = f(\vec{p}_0) \leq \cdots \leq f_n = f(\vec{p}_n)$ by increasing order and calculate $\vec{m} = \frac{1}{n} \cdot \sum_{i=0}^{n-1} \vec{p}_i$.
2. Replace vector $\vec{p}_n$ with the highest $f(\vec{p}_n)$ with $\vec{p}_r = 2 \cdot (\vec{m} - \vec{p}_n)$ and calculate the according value $f_r = f(\vec{p}_r)$. $\vec{p}_r$ is then the **reflection** of $\vec{p}_n$ along the center of all other vectors $\vec{p}_0, \ldots, \vec{p}_{n-1}$ in $S$.
3. If $f_0 \leq f_r \leq f_{n-1}$ repeat from 1.
4. If $f_r < f_0$ calculate $p_s = \vec{p}_r + \vec{m}$ which is the **expansion** along the vector $\vec{p}_r$ and $f_s = f(\vec{p}_s)$.
   (a) If $f_s < f_r$ replace $\vec{p}_r$ with $\vec{p}_s$ in $S$ and repeat from 1.
   (b) Else leave $\vec{p}_r$ in $S$ and repeat from 1.
5. If $f_r \geq f_{n-1}$ and
   (a) ... if $f_r < f_n$ calculate $\vec{p}_{out} = \vec{m} + \frac{1}{2} \cdot (\vec{p}_r - \vec{m})$ and $f_{out} = f(\vec{p}_{out})$. If $f_{out} < f_r$ include $\vec{p}_{out}$ instead of $\vec{p}_r$ as the **contraction outside** and repeat from 1. Else continue with 6.
   (b) ... if $f_r \geq f_n$ calculate $\vec{p}_{in} = \vec{m} + \frac{1}{2} \cdot (\vec{p}_n - \vec{m})$ and $f_{in} = f(\vec{p}_{in})$. If $f_{in} < f_n$ include $\vec{p}_{in}$ instead of $\vec{p}_r$ as the **contraction inside** and repeat from 1. Else continue with 6.

[6] In versions of `MATLAB` until `R2018`
[7] `fminsearch` provides the local minimum closest to the initial parameters `p0`.
[8] https://de.mathworks.com/help/matlab/math/optimizing-nonlinear-functions.html#bsgpq6p-11

6. If this step is reached, then $S$ covers the minimum and its size has to **shrink**. So all vectors $\vec{p}_1, \ldots, \vec{p}_n$ must be replaced with $\vec{v}_1 = \vec{p}_0 + \frac{1}{2} \cdot (\vec{p}_1 - \vec{p}_0), \ldots, \vec{v}_n = \vec{p}_0 + \frac{1}{2} \cdot (\vec{p}_n - \vec{p}_0)$ then repeat from 1.

After precision criteria are met the algorithm terminates. Overall the algorithm is based on resizing the simplex by expanding it away from the worst value in the current simplex in each iteration or shrinking it if the simplex contains a local minimum. So the algorithm is good in covering great distances on rather linear faces. At the same time it converges once a local minimum is found.

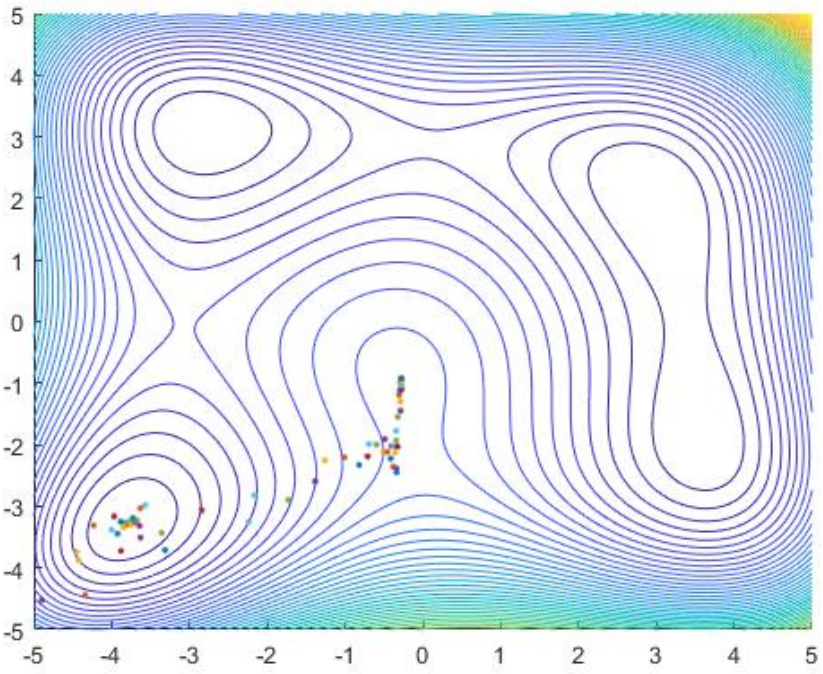



Figure 22: Minimization by using `fminsearch`

22 depicts how `fminsearch` minimizes the Himmelblau's function[9]

$$f(\vec{p}) = (p_1^2 + p_2 - 11)^2 + (p_1 + p_2^2 - 7)^2$$

starting from the local maximum

$$\texttt{p0} = \begin{pmatrix} -0.270845 \\ -0.923039 \end{pmatrix}$$

and expanding towards the closest local minimum. Since the minimum is not contained by the initial simplex, it has to expand first. Then follows the contraction, once the values are close to the minimum.
When choosing $f_{x,y}$ from (4.2.4) as the `eval` function, `MATLAB` method for parameter fitting is provided by the following code.

```
function result = profile_fit(x,y,n)
% input: x - grid vector, y - value vector, n - amount of substances
```

[9] https://en.wikipedia.org/w/index.php?title=Himmelblau%27s_function&oldid=816547958

```
% output: result - parameter vector with 2n entries

% initialize global variables, so they can be used in the eval function
global subst
global x_coor
global y_coor
subst=n;
x_coor=x;
y_coor=y;
% initialize p0 with profile amplitude and variance
amplitude = max(y);
variance = calc_variance_1D(y);
p0 = zeros(2*n,1);
for i=1:n
    p0(2*i-1) = amplitude/n;
    p0(2*i)= variance;
end
% run fminsearch algorithm to determine the parameter values
result=fminsearch(@eval,p0);
end

% eval function for evaluating the quality of a parameter vector
function fun_val = eval(p)
global subst
global x_coor
global y_coor
val=0;
for i=1:subst
    val = val+p(2*i-1)*exp(-x_coor.^2./(2*p(2*i)^2));
end
fun_val = sum((val-y_coor).^2);
end
```

The use of `fminsearch` with the Nelder-Mead algorithm outperforms `powel_method_v1` in the sense of calculation time since it needs less evaluations to reach the minimum.

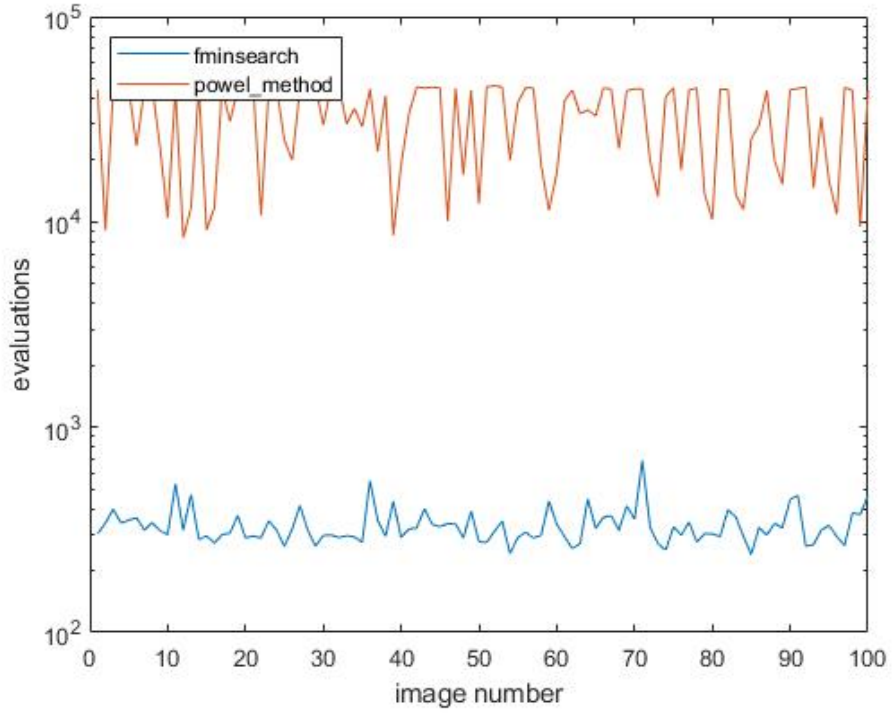



Figure 23: Evaluations for Minimum Determination

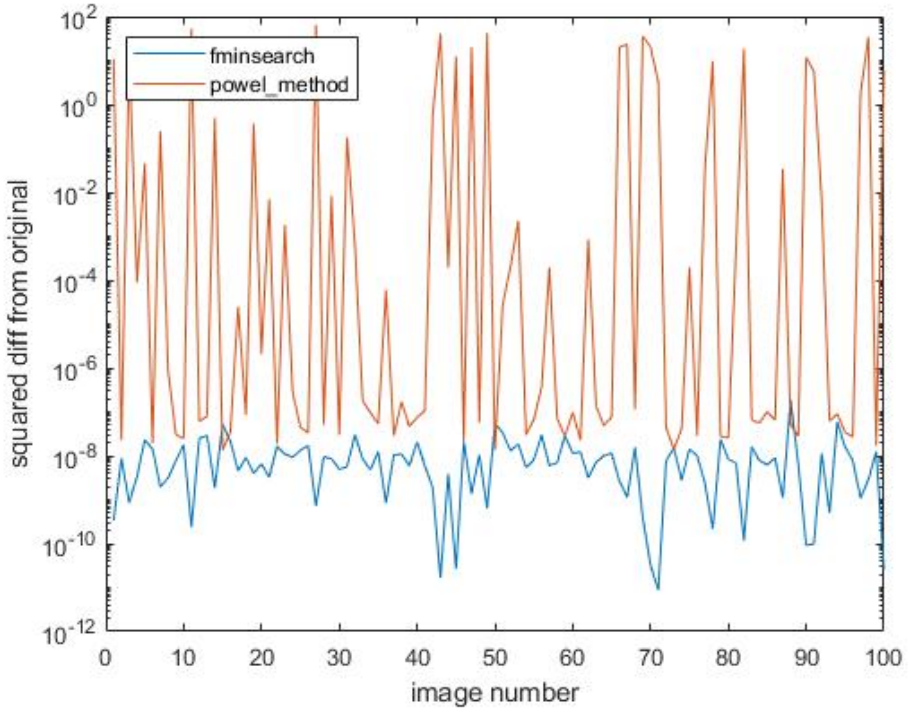



Figure 24: Squared Distance to Data

Figure 23 and Figure 24 depict the fit of 100 generated Gaussian distributed noiseless images with two substances. While `powel_method_v1` requires at least 10.000 evaluations per fit with a `prec` parameter of $10^{-6}$ the method using `fminsearch` only evaluates 200 to 400 times and gets better precision values. However for noisy data the difference between the results of `fminsearch` and `powel_method_v1` approximate each other.

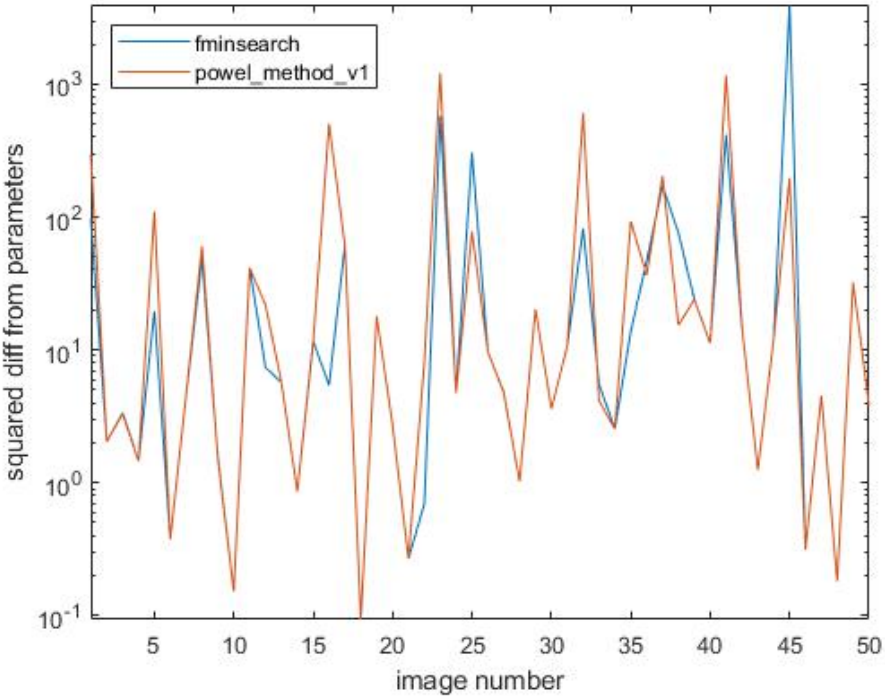



Figure 25: Squared Distance to Original Parameters

As Figure 25 shows the overall outcome of the fitting depends on the given image and how the substances interfere with another. So by observing the fit results of this noisy series of images the use of `fminsearch` seems to be the better choice. Nevertheless `powel_method_v1` is applicable in cases where the results of `fminsearch` don't produce any useful results, especially when one of the parameters is negative. Since no interval for possible values can be defined in `fminsearch` the amplitude algorithm can produce negative amplitudes in order to compensate if the sum of the other values exceeds the total amplitude value. Adding a cost function for negative parameter values does not provide a solution, because the algorithm is designed to search a local minimum and baning negative results will create a local minimum next to the high cost negative values, so the local minimum which would be found in the negative spectrum will shifted to a value close to 0, which again does not present a useful result. In `powel_method_v1` no negative results can be part of the solution, since its search interval for the local minimum $[a, b]$ is defined to be positive. So once `fminsearch` fails to calculate a useful result the outcome of `powel_method_v1` will be taken.

### 4.2.5 Two Dimensional Gauss-Fitting

All mentioned fitting algorithms from 4.2.2, 4.2.3 and 4.2.4 use (4.2.4) as `eval` function. They therefor required a transformation for the data from the image matrix to a concentric averaged profile vector as described in 3.4. In general `eval` can be defined as any function, which returns a

numerical value and the previously mentioned algorithms would determine a parameter vector `p0` for which `eval(p0)` is a local minimum. So instead of reducing the data matrix to a profile vector and using (4.2.4) as `eval` the function

$$x = \begin{pmatrix} -\frac{w}{2} \\ \vdots \\ \frac{w}{2} - 1 \end{pmatrix} \in \mathbb{N}^w, y = \begin{pmatrix} -\frac{h}{2} \\ \vdots \\ \frac{h}{2} - 1 \end{pmatrix} \in \mathbb{N}^h, f_{\overrightarrow{data},\vec{c}} : \mathbb{R}^{2n} \to \mathbb{R} :$$

$$f_{\overrightarrow{data},\vec{c}}(\vec{p}) = \sum_{k=1}^{h} \sum_{l=1}^{w} \left( \sum_{i=1}^{n} p_{2i-1} \cdot \exp(-\frac{(x_l - c_1)^2 + (y_k - c_2)^2}{2 \cdot p_{2i}^2}) - data_{k,l} \right)^2 \tag{4.2.13}$$

can be used as an alternative two dimensional evaluation function for fitting a two dimensional Gaussian function.
Since no reduction of the data is applied this method of fitting is not influenced by numerical reduction errors as described in 3.4. Though this leads to increasing precision, the calculation time increases as well. While in order to get the value of $f_x, y(\vec{p})$ from (4.2.4) the calculation had to iterate through a vector of length $l$ so calculation took $O(l \cdot n)$ the evaluation of $f_{\overrightarrow{data},\vec{c}}(\vec{p})$ takes $O((w \cdot h) \cdot n)$. Especially when using `powel_method_v1` which evaluates more often then `fminsearch` the calculation time increases drastically. When using `fminsearch` the two dimensional fitting results are precise when fitting a single component but increasing the amount of components in the fit is likely to produce unrealistic negative results as discussed in 4.2.4. So the overall use is possible but limited to single component analysis. Nevertheless this method can be modified for application regarding the calculation anisotropic diffusion parameters where the image can not be reduced to a vector without losing valuable information.

### 4.2.6 Determining the Number of Substances

In the sections before possibilities for a fitting algorithm where presented. Besides the definition of a suitable `eval` function the argument `n` which represents the amount of substances in the distribution and therefor is crucial to the determination of the parameters `p` is passed as an argument to the fit function. Nevertheless `n` is not defined at the beginning of the fitting process if it is not already physically determined by the setting of the experiment. So `n` has to be determined by evaluation of the `eval` function for increasing `n` until some criteria is met.
Minimizing the `eval` function for values of `n` is no suitable approach for the given problem, since a sum of more Gaussian functions results in a more accurate fit of the data. From equation (4.2.1) can be seen that any Gaussian distribution can divided in multiple Gaussian functions with partials of the original amplitude but the same $\sigma$ values.

$$A \cdot e^{\frac{-x^2}{2 \cdot \sigma^2}} = \sum_{i=1}^{n} \frac{A}{q_i} \cdot e^{\frac{-x^2}{2 \cdot \sigma^2}}, \sum_{i=1}^{n} q_i = 1 \tag{4.2.14}$$

The termination of the increase in `n` must include the outcome of a fit `p` and compare its values to previous. So an approach to define a termination criteria for determining `n` is to find the maximal amount for `n` for which a pair of sigma values is similar. At which point two values consider being similar requires thresholding of their subtraction one by another. If one of all division results is within a thresholded interval around 1 the calculation is terminated and the parameter vector of the previous iteration without any similar $\sigma$ values is taken. Further an amplitude value for a distribution can approach 0 which indicates, that this distribution is just the result of noisy data

and no more actual distributions can be found. This must also be part of the termination criteria. As mentioned in 4.2.4 `fminsearch` does not exclude negative results, so if a negative result is part of the parameter results `p` the calculation has to be tested with `powel_method_v1`.
For efficient evaluation of the criteria the vector is sorted first and then successive values are compared. If their difference to each other or to zero is smaller than a threshold the loop is terminated and the last valid parameter vector is returned. Also if `fminsearch` produces a negative parameter the result of `powel_method_v1` is taken.

```
function p = sub_num(x,y,params)
% input: x - grid vector, y - value vector, threshold - minimal
    difference
% of two values to terminate
% output: p - parameter vector
global thresh
% set parameters either by user in params class or from defaults
if isempty('params') || ~exist('params','var')
    thresh = params.threshold;
    max_n = params.max_n;
    powel_prec = params.powel_prec;
    n = params.init_n;
else
    thresh = 1;
    max_n = 5;
    powel_prec = 10^-6;
    n = 1;
end
% initial values to start the loop
p_new = [thresh+1;2*thresh+1];
finished = 0;
% loop while two values are not similar
while(n <= max_n && finished ~= 1)
    p = p_new;
    % fit with fminsearch
    p_new = profile_fit(x,y,n);
    if(min(p_new) < 0)
        % in case p contains negative numbers
        p_new = powel_method_v1(x,y,n,powel_prec);
    end
    n = n + 1;
    finished = find_sim(p_new);
end
end
% determine whether values are similar in p
function sim = find_sim(p)
global thresh
% get sigmas from parameters
sigmas = p(2:2:end);
% sort sigma array
```

```
sorted = sort(sigmas);
% if smallest value is too close to zero stop
if(sorted(1) < thresh)
    sim = 1;
    return
else
    % else define values as not similar
    sim = 0;
end
for i = 2:length(sorted)
    % check for each value whether it is similar to its neighbor
    if((sorted(i) - sorted(i-1)) < thresh)
        sim = 1;
        return
    end
end
end
```

By using this algorithm the parameters can be fitted for $\approx 4$ to 5 substances, which with the current algorithm results in being the maximal amount before interference makes two distributions inseparable.

# 5 Physical Data Evaluation

After fittng all images the results are stored in matrices $\overrightarrow{\sigma}, \overrightarrow{A} \in \mathbb{R}^{n\times T}$ wtih $n$ amount of detected substances and $T$ the total amount of images.

$$\overrightarrow{\sigma} = \begin{pmatrix} (\sigma_1)_1 & \cdots & (\sigma_1)_T \\ \vdots & \ddots & \vdots \\ (\sigma_n)_1 & \cdots & (\sigma_n)_T \end{pmatrix} \in \mathbb{R}^{n\times T}, \overrightarrow{A} = \begin{pmatrix} (A_1)_1 & \cdots & (A_1)_T \\ \vdots & \ddots & \vdots \\ (A_n)_1 & \cdots & (A_n)_T \end{pmatrix} \in \mathbb{R}^{n\times T} \tag{5.0.1}$$

Hereby $(\sigma_i)_j$ is the variance of substance $i$ in image $j$ was taken and analogously notation is used for $(A_i)_j$.
Since the order of the outcome of parameter vector `p0` from (4.2.3) has no guaranties of beeing sorted the same way for each image firstly the variances and amplitudes have to be assigned to substances.

## 5.1 Prognostic Assignment

The matrices containing the $\sigma$ and $A$ values have to be created by sorting all vectors $(\vec{\sigma})_1, \ldots, (\vec{\sigma})_t$ and $(\vec{A})_1, \ldots, (\vec{A})_t$. For passing time the variance of a substance increases and the amplitude decreases, hence sorting the values will indicate which $\sigma$ value belongs to which substance. Due to the fitting procedure, the amplitude $A$ and the sigma value $\sigma$ are always linked in a picture in the context that image $j$ contains a substance with amplitude $A$ and sigma value $\sigma$. Therefor when assigning $\sigma$ to a substance $A$ is automatically assigned as well. So only assigning one of the two values to a substance is necessary. Though sorting the $\sigma$ values by order and assigning the $i$-th highest $\sigma$ value to substance $i$ solves the problem for most cases, still some cases might exist, where the diffusion of one substance is faster then the other ones but starts later. So the development of the variances might unfold like in the constructed example depicted in Figure 26.

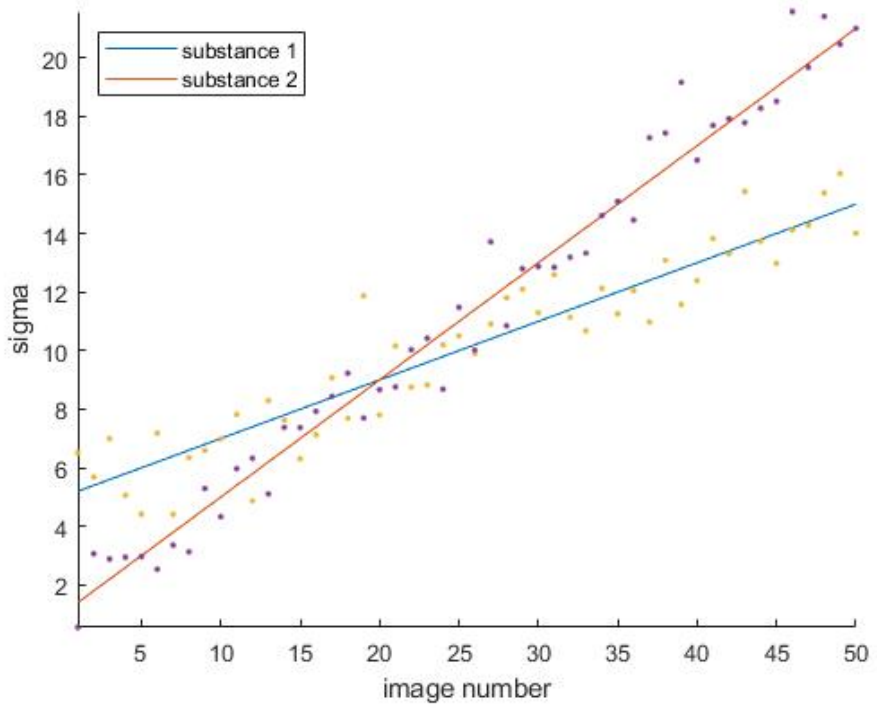



Figure 26: Two Substances with Intersecting Sigma Values

For Figure 26 simply sorting the variances by value would not solve this problem. So another approach would be prognostic sorting, where a linear function is fitted through the given variance values $(\sigma_i)_1, \ldots, (\sigma_i)_j$ of substance $i$ and the value in the next parameter vector is assigned to $(\sigma_i)_{j+1}$ which is closest to the expected value. This prognostic assignment also solves the problem of falsely fit values which would falsify the calculation. A solution would be to discard values which are further away than some threshold from the expected value. In [2][1.4,p. 43] a default value for discarding a sigma value is when it has changed more than 60% to the previous value. In the case of prognostic assignment a value is only regarded as useful if it lies within an interval of $[\sigma_{prog} - \sigma_{prog} \cdot a, \sigma_{prog} + \sigma_{prog} \cdot a]$ with $a$ being the percentage of allowed deviation. Naturally values can only be prognosticated from at least two given values.
Since a prognosis is a forecast for an unknown value by the before gathered data it has to be determined at which time intervals the images were taken. This requires further input data in form of a unique time stamp for each image. If no time stamps are provided the diffusion values can still be calculated with a temporal offset between each image. So either with provided values or with a general temporal offset a line is fitted through the sigma values using a statistical based linear regression.

## 5.2 Regression Model for Fitted Parameters

The relation between the sigma values $\vec{\sigma}$ and the diffusion coefficient $D$ is described in equation (2.8) of [3, 2.3.1,p. 39].

$$\sigma_t^2 = 2 \cdot Dt \tag{5.2.1}$$

With $t$ being the passing time which can be determined by the time stamp of the image.
Solving (5.2.1) for $D$ results in

$$D = \frac{\sigma_t^2}{2 \cdot t} \tag{5.2.2}$$

So $D$ is the gradient, by which $\frac{\sigma_t^2}{2}$ increases depending on $t$.
Also as described [3, 2.3.1,p. 39] some time of the diffusion has passed when the first image is taken so an offset $\sigma_0$ has to be included in equation (5.2.1).
For determining $D$ the function

$$\sigma_t^2 = 2 \cdot Dt + \sigma_0 \tag{5.2.3}$$

is fitted to the determined variances $\overrightarrow{\sigma}$ for each of the $n$ substances. Using the definition (5.0.1) all $n$ diffusion factors can determined by applying linear regression models to the variance values. Fitting the values to $y = a + mx$ as described in [4, 1.5,p. 21] leads to

$$m = \frac{\sum_{j=1}^{T}(x_j - \bar{x}) \cdot (y_j - \bar{y})}{\sum_{j=1}^{T}(x_j - \bar{x})^2}, b = \bar{y} - m \cdot \bar{x} \tag{5.2.4}$$

with $\bar{x}$ and $\bar{y}$ being the average values of all $x_j$ and $y_j$.

$$\bar{x} = \frac{\sum_{j=1}^{T} x_j}{T}, \bar{y} = \frac{\sum_{j=1}^{T} y_j}{T}$$

When applied to the entries of the diffusion vector

$$\vec{D} = \begin{pmatrix} D_1 \\ \vdots \\ D_n \end{pmatrix}$$

and with the usage of (5.0.1) the diffusion coefficients can be calculated by

$$D_i = \frac{\sum_{j=1}^{T}(t_j - \bar{t}) \cdot ((\sigma_i)_j - \bar{\sigma}_i)}{\sum_{j=1}^{T}(t_j - \bar{t})^2} \tag{5.2.5}$$

with $D_i$ being the diffusion factor of substance $i$. So $D_i$ and $(\sigma_i)_0$ can be calculated by passing the sigma values of a substance and the time stamps of the images to following `MATLAB` function.

```
function [m,b] = elr(x,y)
% input: y - vector containing values, x - grid vector
% output: m - gradient of the line, b - y value at x=0

% calculate averages
x_avg = mean(x);
y_avg = mean(y);
s_xy = sum((y-y_avg).*(x-x_avg));
s_yy = sum((y-y_avg).^2);
% calculate parameters for linear regression
m = s_yy/s_xy;
b = y_avg - m*x_avg;
end
```

Thereby all values $D_i$ can be calculated. Should a value differ suspiciously from the rest of the surrounding as described in subsection 5.1 it can be left out of the calculation with its according `x` value by leaving it out from the vector which is passed to `elr`. The function will then calculate the regression only including valid sigma values. Since throughout the calculation all transformations kept the sizes of the grid at the original unit of a pixel the value of the diffusion coefficient after the regression is pixel per time. For transforming the diffusion coefficient to a physical unit information regarding the size of a pixel is required. This is either given directly from the pixel size of the images taken by the microscope or can be calculated by dividing a given image size by the amount of pixels in the image. Is no pixel size provided the results remain in the abstract unit of a microscope image pixel per time.

## 5.3 Determining Substance Mass

A further factor that has to be calculated is the total amount of each substance. As described in [3][2.3.2,p. 40] the total amount $M_i$ of a substance are proportional to the values of the amplitude for an image over time $t$. The proportionality of $M_i$ and the amplitude values $(A_i)$ is shown by equation

$$(A_i)_t = \frac{M_i}{\sqrt{(4\pi D_i \cdot t)}^d} \tag{5.3.1}$$

with $d$ being the dimensionality of the diffusion and $D_i$ being its diffusion coefficient. The dimensionality indicates in how many dimensions the material diffuses and in general only three values are possible here, for which the diffusion developments are depicted in **??**, **??** and **??**. As equation (5.3.1) shows $M_i$ is dependent from the diffusion coefficient $D_i$ and the dimensionality $d$. Whereas methods to calculate $D_i$ for substance $i$ are presented in 5.2 the dimensionality $d$ is needed to calculate $M_i$. Since the methods presented so far are mainly applicable for two dimensional diffusion the $d$ parameter is set to 2. For alternative calculations of one or three dimensional diffusion the parameter can be adapted so it fits the set up of the experiment. A calculation of the dimensionality is possible and a method therefor is mentioned in [3][3.3,p. 55], though for unknown values $M_i$ and $d$ the calculation is hindered by the fact that the values of $(A_i)_t$ are strongly dependent on both parameters.
After just one parameter $M_i$ is left to be calculated a fitting value for $M_i$ can be determined by applying a least squares solving algorithm like `fminsearch` to equation:

$$f_{(\vec{A}_i),d,D_i,(\vec{t}_i)}(M_i) = \sum_{k=1}^{T} \left( (A_i)_k - \frac{M_i}{\sqrt{(4\pi D_i \cdot (t_i)_k)}^{d_i}} \right)^2 \tag{5.3.2}$$

for the given amplitude vector $(\vec{A}_i)$ as well as the values $d$ and $D_i$. So by using this equation as evaluation function for `fminsearch` a `MATLAB` implementation is:

```
1 function M = det_dim(A,dimens,D,timestamps)
2 % input: A - amplitudes, M - substance amount, D - diffusion
      coefficient
3 % output: d - dimensionality of diffusion
4
5 % define global parameters to pass them to eval
6 global amps
7 global dimension
8 global dif
```

```
global times
amps = A;
dimension = dimens;
dif = D;
times = timestamps;
% find dimensionality
M = fminsearch(@eval,2);
end
% eval function serves as quality of the determined value
function value = eval(M)
global amps
global dimension
global dif
global times
% calculate least squares
value = 0;
for i = 1:length(times)
    value = value + (amps(i)-M/(sqrt(4*pi*dif*times(i))^(dimension)))
        ^2;
end
end
```

After gaining the $M_i$ values for all substances the evaluation of the experiment is completed. All fitted values are then presented to the user and stored to a table structured document for usage. Should the results seem suspicious in any form the process can be restarted with a different choice for a noise filter, a different method for the center detection, a different profile creation method or another fitting algorithm. Due to enhanced computation speed the calculation is expected to last a fraction of the duration of an *AFA* evaluation. This way when choosing a suboptimal set of parameters the calculation can be started again after a rather short duration.

# 6 Conclusion and Outlook

In the course of this study a number of algortihms where developed for image preproessing including the detection of the center, fitting algorithms and the evaluation of the fitting procedure. It was found that the optimal procedure for the data processing includes the following steps:

1. Noise filtering with Fourier low pass filter
2. Center detection using `circular_detection_v2` (described in **??**)
3. Profile creation using `radial_distribution` (described in **??**)

The precision of the diffusion coefficient determination using this procedure is ...
The proposed fitting algorithms allow to determine the number of substances in the probe. Based on the multicomponent analysis, methodology was developed for detection of multiple diffusing species in the probe and independent determination of their diffusion coefficients.
A topic which has not been adressed in this study is the dimensionality of the diffusion process. In this respect, a sorrowful analysis of the evolution of the pre-exponential factor over time deserves further research.
All algorithms developed in this study are based on numerical calculation approach. Interpreting the given problem from a different perspective the employment of machine learning methodology can be beneficial. The implementation of these algorithms requires a large amount of evaluated and labeled data, eventually based on diffusion simulation data.

# 7 Acknowledgment

I would like to thank Prof. E. Schömer for fruitful discussions on algorithmic approaches and supervision. I'm also thankful to professor S. Seiffert for providing the assignment and support regarding physical chemistry. Additionally I would like to thank E. Chesnokov (Novosibirsk University) for his mathematical support and proof reading. Furthermore I am grateful to T. Gorelik for conceptional discussions.

# References


[1] Numerical Recipes, William H. Press and Saul A. Teukolsky and William T. Vetterling and Brian P. Flannery, Cambridge University Press, 3rd edition, 2007.

[2] AFA Documentation, S. Seiffert, G. Hauser, University Clausthal, 2007.

[3] Ermittlung von Diffusionskoeffizientenverteilungen mittels Fluorescence Recovery after Photobleaching, G. Hauser, University Clausthal, 2008.

[4] Wahrscheinlichkeitstheorie und Statistik, N. Henze, D. Kadelka, KIT Karlsruhe, 2010.

[5] Image Engineering, Yu-Jin Zhang, De Gruyter, 2017.

[6] Numerik für Ingenieure und Naturwissenschaftler, Wolfgang Dahmen, Arnold Reusken Springer-Verlag Berlin Heidelberg, 2006

[7] The Mathematics of Diffusion, J. Crank Brunel University Uxbridge, 1975