

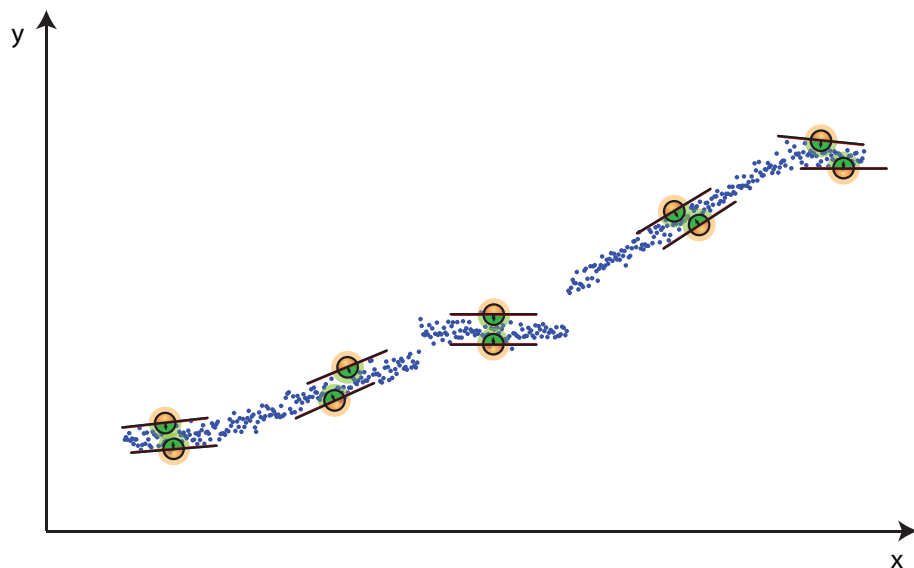
MASTER OF SCIENCE THESIS

Soap-Particle Clustering

for Non-linear System Identification

S.J. Thus BSc

September 29, 2009



Soap-Particle Clustering

for Non-linear System Identification

MASTER OF SCIENCE THESIS

For obtaining the degree of Master of Science in Systems and Control
at Delft University of Technology

S.J. Thus BSc

September 29, 2009

Faculty of Mechanical, Maritime and Materials Engineering
Delft University of Technology



Delft University of Technology

Copyright © Delft Center for Systems and Control

All rights reserved.

DELFT UNIVERSITY OF TECHNOLOGY
DELFT CENTER FOR SYSTEMS AND CONTROL

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering for acceptance a thesis entitled “**Soap-Particle Clustering**” by **S.J. Thus BSc** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: September 29, 2009

Supervisor:

Prof.dr. R. Babuška

Readers:

Ir. J. van Ast

Prof.dr.ir. J. Hellendoorn

Dr.ir. D. de Ridder

Summary

In this thesis report we propose a novel clustering method to be used to create non-linear models in black-box system identification by decomposing the operating range into a number of local operating regimes. We use Soap Particle Clustering on a data matrix containing samples of time signals and delayed signals to find the affine local subsystems: each cluster corresponds with a local subsystem.

When we use current clustering algorithms, we first need to choose the shape and number of clusters. We propose a method that estimates this shape with artificial soap particles, so that this shape does not have to be chosen a priori. The number of clusters can be derived from the shape, so that the number of clusters also does not have to be predefined. The proposed method uses localized and uncoupled movement rules for the soap particles. Localized rules have the advantage that the method is robust to outliers. Additionally, the uncoupled movement rules can be parallelized and the localized movement are based on only a small subset of the data set: the neighbourhood of the soap particle. These last two features are beneficial for large data sets.

The Soap Particle algorithm consists of three stages: Soap Particle Hull Finding, Soap Particle Clustering, and Soap Particle Non-linear System Identification. These three are applied to the data set in this order to obtain a non-linear model.

Soap Particle Hull Finding

The goal of the Soap Particle Hull Finding stage is to find the locally convex hull of the data set. The locally convex hull and its boundary set have the following set of requirements:

- All edges of the boundary are smaller than $e_{\max}$
- The hull is minimal
- The boundary encloses the data

The global requirements inspire the following movement rules for each soap particle:

Data selection Apply the other rules only to the data closer than $\frac{e_{\max}}{2}$

Translation Move the soap particle as close as possible to the data points

Rotation Rotate the soap particle such that the margin between the soap particle is maximized

First the soap particle needs to be initialized in the space of the data set, such that its neighbourhood is not empty. Then, the movement rules are iterated until the soap particle stops moving, which indicates the soap particle is located on the boundary of the locally convex hull. Multiple soap particles are initialized at different positions within the space of the data set. When a sufficient number of soap particles is used, and these soap particles are initialized adequately, eventually the full hull will be described by these soap particles. Each soap particle will then describe a single face of the hull.

The algorithm is compared with the QHull algorithm, that identifies the globally convex hull of a data set. The experimental results show that the converged positions of the soap

particles are generally on boundary of the hull. The Soap Particle Hull Finding algorithm gives solutions similar to the optimal solution of the QHull algorithm, with the added advantage that it can identify intrinsic convex substructures in the data sets: the locally convex hull instead of the globally convex hull. While the algorithm did not find all the faces of the hull; the performance is still good enough for the next two stages.

Soap Particle Clustering

In the training section of the Soap Particle Clustering stage, a data set is partitioned into convex clusters. With the Soap Particle Hull Finding algorithm the locally convex hull is found, and using an adapted maximum clique finder the largest convex regions are found. These largest convex regions correspond with the clusters.

With the Soap Particle Labelling algorithm of the clustering stage, the cluster membership degree of a data point can be determined. This fuzzy membership degree depends on the minimal margin between the cluster boundary and the data point.

The thus found partitioning of the data set is compared to Gustafson-Kessel Fuzzy c -means clustering. Both algorithms result in similar ROC-ratios. The calculations of the Soap Particle Clustering algorithm were faster in our computer experiments, but it also needed more reruns due to an incomplete description obtained from the Soap Particle Hull Finding stage. The need for a rerun is indicated by the Soap Particle Clustering algorithm by a large number of outliers; this is an advantage compared with the c -means clustering, which does not indicate the need for a rerun.

Soap Particle Non-linear System Identification

For system identification, a regressor matrix and regressand are formed from the measurement signals and delayed signals. The Soap Particle Non-linear System Identification uses the Soap

Particle Clustering algorithm to find the local affine subsystems in the combined regressor matrix and regressand. Each subsystem is then modelled separately with a linear least squares estimate. This forms the model.

A 1-step ahead prediction can be made by appending the regressor, that is made from the given model inputs, with the estimated model output from each subsystem. We determine the membership degree to the corresponding clusters for each combined given model input and estimated model output. The sum of these model outputs weighted according to their associated membership degrees, is the global model output. A simulation can be performed by using this estimated model output as one of the inputs for the next time step.

The model obtained is found to have a variance accounted for comparable with Gustafson-Kessel fuzzy c -means clustering with local linear consequents for the simulations of the systems tested in our computer experiments.

Conclusions and recommendations

The Soap Particle Non-linear System Identification performs comparable to Gustafson-Kessel fuzzy system identification and has the advantage that the number of clusters and the shape of these clusters are determined by the algorithm adaptively. The robustness to outliers and the suitability for large data sets due to parallelization and localized calculations have not been tested, but have been shown theoretically.

Future research should focus on the following elements: the number of soap particles can be made adaptive, the generalization parameter can be redefined, and methods with a global optimization function can be used for estimating the affine subsystems.

Table of Contents

Summary	v
Nomenclature	xiii
1 Introduction	1
1.1 Problem statement	3
1.2 Motivation	3
1.3 Proposed solution	6
1.4 Assumptions	7
1.5 Structure	8
2 Soap Particle Hull Finding	9
2.1 Introduction	9
2.2 Global objective	10
2.2.1 Globally convex hull	10
2.2.2 Locally convex hull	12

2.2.3	Maximum edge length	13
2.2.4	Summary of the objectives	14
2.3	Soap Particle definition	16
2.4	Local objectives	18
2.5	Soap Particle movement (local)	19
2.5.1	Data selection	20
2.5.2	Rotation	20
2.5.3	Translation	21
2.5.4	Iteration and stopping criterion	22
2.6	Hull finding algorithm (global)	25
2.6.1	Initializion	26
2.6.2	SP Movement	26
2.6.3	Stopping conditions	27
2.7	Validation	28
2.7.1	Computer experiments	28
2.7.2	Results of the experiments	29
2.7.3	Conclusions of the experiments	30
2.8	Conclusions and recommendations	32
2.8.1	Conclusions	32
2.8.2	Recommendations	32
3	Soap Particle Clustering	35
3.1	Introduction	35
3.2	Training	36
3.2.1	Objective	36

3.2.2	Input, output, and assumptions	39
3.2.3	Soap Particle Cluster Training	39
3.3	Labelling	44
3.3.1	Objective	44
3.3.2	Input, output, and assumptions	45
3.3.3	Soap Particle Labelling Algorithm	46
3.4	Validation	48
3.4.1	Performance measures	48
3.4.2	Comparison with Gustafson-Kessel clustering	50
3.4.3	Computer experiments	51
3.4.4	Results of the experiments	51
3.4.5	Conclusions of the experiments	54
3.5	Conclusions and recommendations	56
3.5.1	Conclusions	56
3.5.2	Recommendation	56
4	Soap Particle Non-linear System Identification	57
4.1	Introduction	57
4.2	Model building	58
4.2.1	Objective	58
4.2.2	Input, output and assumptions	60
4.2.3	SP Modelling Algorithm	61
4.3	Simulation and prediction	64
4.3.1	Objective	64
4.3.2	Input, output and assumptions	64

4.3.3	SP Prediction Algorithm	65
4.4	Validation	67
4.4.1	Performance measures	67
4.4.2	Comparison	68
4.4.3	Computer experiments	68
4.4.4	Results of the experiments	69
4.4.5	Conclusions of the experiments	72
4.5	Conclusions and recommendations	73
4.5.1	Conclusions	73
4.5.2	Recommendations	73
5	Conclusions and Recommendations	75
5.1	Conclusions	75
5.2	Recommendations	76
	Bibliography	79
A	Recommended Approaches to Improvements	81
A.1	Convergence	81
A.2	Initialization, Redundancy, and Completeness	82
A.3	Noise	86
A.4	Low dimension	87

Nomenclature

Sets

$\mathcal{C}$	Cluster
$\mathcal{G}$	Convex Hull
$\mathcal{H}$	Locally Convex Hull
$\mathcal{S}$	Set of Soap Particles
$\mathcal{V}$	Set of Vertices of the Boundary Set
$\mathcal{Z}$	Data-set
Φ	Mapped Space
$\mathbb{R}$	Set of the Real Numbers
Conn	Set of Vertices Connected to a Particular Vertex

Variables

ϵ_t	White Noise
ϵ	Machine Epsilon
λ	Real Valued Weight
μ	Membership Degree

$\bar{\mu}$	Not Normalized Membership Degree
s	Soap Particle
c	Number of Clusters
d	Dimensionality
$e_{\max}$	Maximum Edge Length
e	Edge Length
g_{label}	Generalisation Parameter for Labelling
g_{train}	Generalisation Parameter for Training
k	Time-step
m	Margin
N	Number of Data Points
n	Number of Particles
o	Order of the System
R^2	Coefficient of Determination
r	Radius
t	Time
u	Input
y	Output

Matrices and Vectors

$\mathbf{c}$	Constant vector
$\mathbf{o}$	Orientation
$\mathbf{s}$	Position
$\mathbf{v}$	Vertex of the Optimal Hull
$\mathbf{x}$	Independent variable
$\mathbf{x}$	Point in $\mathbb{R}^d$
$\mathbf{y}_{\text{est}}$	Estimated output
$\mathbf{y}$	Dependent variable
$\mathbf{z}$	Data Point

A	Adjacency Matrix
U	Partition matrix
w	Weight Matrix
X	Regressor matrix

Subscripts and Indices

0	Initial
i	Index of Soap Particle
j	Index of Other Soap Particle
l	Index of Cluster
p	Index of Data Point
q	Index of Other Data Point
nb	Neighbourhood

Acronyms

ARX	AutoRegressive with eXogenous input
FPR	False Positive Rate
GK	Gustafson-Kessel Clustering Algorithm
LTI	Linear Time-Invariant
MISO	Multiple-Input Single-Output
NARX	Non-linear AutoRegressive with eXogenous input
ROC	Receiver Operating Characteristic
SLHC	Single Linkage Hierarchical Clustering
SP	Soap Particle
TPR	True Positive Rate
VAF	percentage of Variance Accounted For

Chapter 1

Introduction

Maybe, one day you can buy a cheap set of tiny multi-purpose robots at the mall. You could just let loose these robots at home and they would clean and repair your house, hold your stuff when you need that third hand, and locate these items that always seem to get lost. While this might be a distant future, we already can work at some of the challenges that will occur.

For example, it is useful to create mathematical models of these robots (and other systems). Such models can be used for the analysis of the robots, for example for engineering and design, without the need to build the robots prototypes. Models can also be used for simulation and prediction; testing scenarios that would otherwise be too expensive. We can use models to filter sensor inputs, and estimate states that are not measured. All these applications are also useful when the model is used to create a controller: to determine the sensor outputs that are required; to tune a controller offline, with less risk; to make an inverse controller; or to acts as part of the controller as an observer or state estimator.

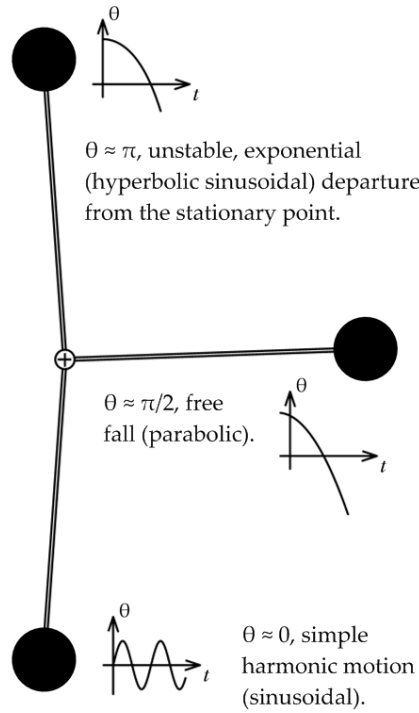


Figure 1-1: A pendulum is a system with nonlinear dynamics and has different behaviours around different operating points

When we want to create classical linear models of most robots, we are faced with a challenge: modelling hinge joints. The hinge joints are similar to a pendulum (Figure 1-1). When we formulate the equations of motion, we see that a sine is present, which is an example of a nonlinear phenomenon:

$$\ddot{\theta} + g \sin \theta = 0 \quad (1-1)$$

Hinges are not the only non-linear phenomena, other examples are aerodynamic forces, friction, gyroscopic effects, and saturation.

A traditional way to deal with these non-linearities is by starting from first principles: for the pendulum these are the mechanistic laws. However not always the model structure is known in sufficient detail. A solution to this problem is to use black-box modelling: making a model directly from measurements on the system.

1.1 Problem statement

The goal of this work is to find a novel algorithm for non-linear black-box system identification. The algorithm should create a model of a dynamical system from the measurements on the inputs u and outputs y of the system.

We will focus on discrete time models, as measurements are generally discrete time. The model input vector then consists of lagged input and output measurements, for example for a second-order model with two inputs at time instant k , the input vector is:

$$\mathbf{x}(k) = [y(k) \quad y(k-1) \quad u_1(k) \quad u_1(k-1) \quad u_2(k) \quad u_2(k-1)] \quad (1-2)$$

The model then becomes:

$$y(k+1) = \Phi(\mathbf{x}(k)) \quad (1-3)$$

These observations are the starting point for the algorithm.

The model that is provided should be able to be used for all the mentioned uses. We will not provide examples for all these applications, but we will use the simulation and prediction as examples of approaches to use the model, while similar approaches for the other applications of the model can be followed.

Summarizing, the goal of this work is to show a novel method to create a non-linear model from series of observations on system inputs and system outputs, and also give examples of this model used for prediction and simulation.

1.2 Motivation

A key point in system identification is finding the right model structure. Many problems can be described nicely with linear models, such as the finite impulse response (FIR) model (Eq. 1-4), however they remain limited in their operating range (Sjöberg et al. 1995).

$$y(t) = b_1 u(t-1) + \dots + b_n u(t-n) + e(t) \quad (1-4)$$

Non-linear models on the other hand give that much freedom, that a good model structure is hard to find. One solution is to generate a piecewise linear model, which has the advantage that on the resulting simpler local models existing control techniques can be applied.

A critical step in the local linear approach, is the decomposition of the operating range and the identification of the local regimes. An effective technique is to use clustering methods to find these premises (Babuška and Oosterom 2001).

Current clustering algorithm have drawbacks when applied to system identification, these include:

- The number of clusters need to be defined beforehand
- The cluster shape needs to be defined beforehand
- Large datasets result in a high computational complexity
- The algorithm are often not robust to outliers

This work aims to solve these four drawbacks.

For the algorithm, we should incorporate all available information. When clustering is used for identifying linear subsystems, there is prior information about the cluster shape: all clusters should be reasonable similar to linear manifolds within a limited domain. However, currently there are few clustering algorithms that focus on this condition. Furthermore, the affine subsystems can have different noise levels, which for example for Gaussian noise result in different cluster thicknesses. Our algorithm will adapt to these different shapes. The *c*-means algorithm results in ellipsoid shaped clusters, which have some similarity with linear manifolds: research has shown that even this limited similarity can be sufficient for extracting these linear subsystems. Our algorithm adapts itself to any shape an affine subsystem can have, which will result in better extraction of the affine subsystems.

The field of swarm intelligence is now applying principles from nature to solve problems in which cooperation is important. Biology is not the only field that can be an inspiration. The essence of complex macroscopic behaviour emerging from simple rules on the microscopic

scale can also be found in physical processes. This formed the basis of a literature study to find physical situations that can help existing problems in the field of systems and control (Thus, van Ast, and Babuška 2008).

Out of the research two promising examples were developed: soap particles as clustering algorithm and air particles as optimization algorithm. This report will focus on the soap particles as clustering algorithm to be used in system identification.

The main target of this work is to develop a novel algorithm, evaluate the potential benefits of the method and identify any drawbacks and needed improvements. It is not our aim to immediately deliver a ready-to-use toolbox; such a toolbox can be created as future work on the basis of this report.

While the aim of this report is to create a clustering algorithm for system identification, the algorithms should also be applicable to other fields.

1.3 Proposed solution

This section only describes the core idea of the proposed solution and a more in depth discussion is the subject of the subsequent chapters.

The core task is to find linear subsystems in the data hyperspace. These linear subsystems are the largest convex shapes in the data. Therefore, to identify the local linear systems, we can look for the largest convex shapes. These convex shapes we can find by describing the contour of the data, and then split this contour into the convex parts. Thus, we propose to accomplish the task to find the linear subsystems in three stages: first find the contour of the data, then find the largest convex shapes within this contour, and finally create a model on the basis of this partitioning. We have illustrated the stages in Figure 1-2.

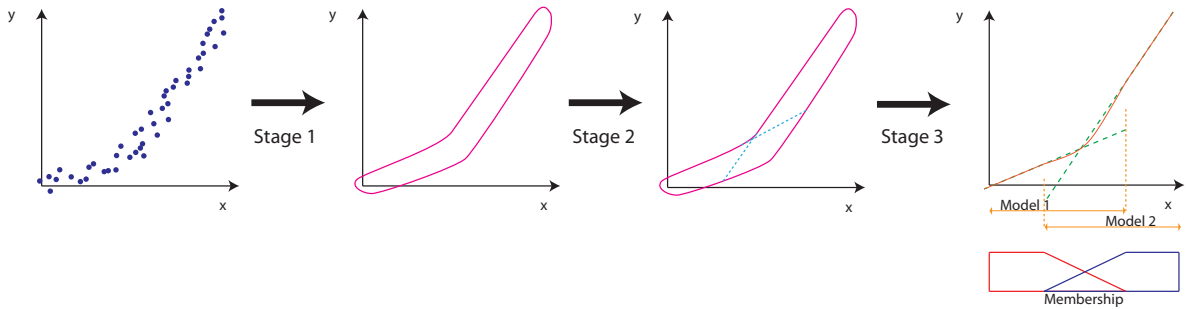


Figure 1-2: The approach consists of three stages: SP Hull finding, SP Clustering, and SP System Identification

A key element of the proposed algorithm is to find the contours in the data. In nature, there are some examples of processes involving finding contours available. Soap is such an example.

Soap is a surfactant. Soap particles have a head that is hydrophilic: the head will be attracted to water. The soap particles also have a tail, which is hydrophobic: it is repelled by the water. Due to the balance between these two forces, the soap particle will –in a emulsion of oil in water– position themselves at the interface between the oil and the water (Figure 1-3). This is similar to the cluster boundaries in clustering problems and this inspired us to create artificial soap particles and use them as a clustering algorithm.

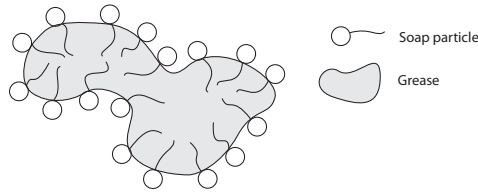


Figure 1-3: Soap particles

1.4 Assumptions

To provide more clarity, we have made some choices in the cases we will treat. These are however not limitations on the algorithm itself.

- Discrete time dynamic systems
- Limited background noise (some outliers are allowed)
- Low to medium system order
- System order is known
- Multi-Input, Single Output (MISO)

These assumptions are not intrinsic to the algorithm and the algorithm can straightforwardly be extended to include cases the do not conform to these assumptions.

1.5 Structure

In the following discussion we will show the proof-of-principle of the algorithm, we will indicate how the algorithm solves mentioned four drawbacks of current algorithms and we will find out opportunities and the main sections for improvement. Also, we provide some suggestions to approach these improvements.

The soap particle clustering algorithm for system identification can be split in 3 distinct stages: describing the data, clustering the data, and identifying the system within the data. Therefore, we will treat these stages in different chapters: Chapter 2, Chapter 3, and Chapter 4. Each of these chapters will describe the objective of the respective stage and the preparation needed first. Then we will outline the algorithm itself. The last two sections of these chapters will consist of the validation and conclusions. Finally we will summarize the found conclusions and recommendations in Chapter 5.

Chapter 2

Soap Particle Hull Finding

2.1 Introduction

Soap can be used to wash, for example, dirty cloths. When we zoom in to the microscopic level, we see that the soap actually consists of molecules: small particles. Those small particles move through the water towards the fatty stain, and end up at the interface of the fat and the water. Soap particles have a tail that is attracted to the fat: it is hydrophobic. Therefore, they move towards the fat. But they also have a head that is attracted by the water and repelled by the fat, therefore the soap particles do not fully immerse in the fat, but remain on the interface: the tail in the fat and the head outside of the fat. Eventually, the soap particles together cover the full surface of the fatty stain.

In analogy to this solution of nature to find the surface of a stain, we use artificial soap particles to find the surface (hull) of a data set. In particular, the first stage of our algorithm to create a non-linear mathematical model (Chapter 1), involves finding clusters a the data set, by finding the hull of a data set; we use artificial soap particles to find this hull.

2.2 Global objective

The goal of this stage is to find the hull of the data, as depicted in Figure 2-1.

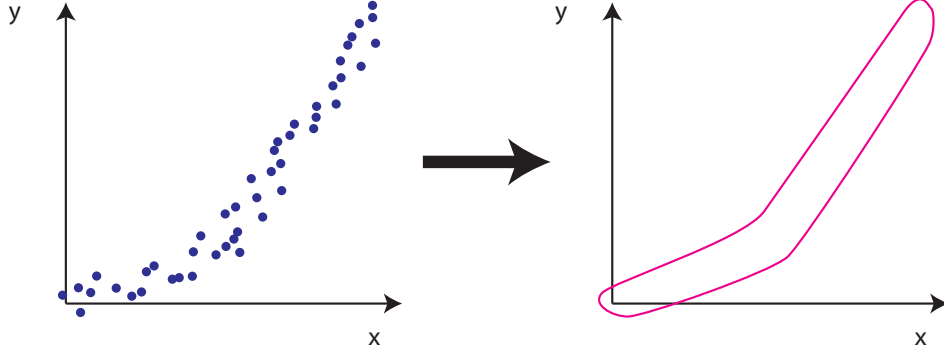


Figure 2-1: The first stage: finding the hull

Formally, given a data set $\mathcal{Z}$ with N data points $z_p \in \mathbb{R}^d$

$$\mathcal{Z} = \left\{ z_p \mid p = 1, \dots, N \right\} \quad (2-1)$$

the objective is to find the locally convex hull $\mathcal{H}$ of $\mathcal{Z}$ ¹, which is the smallest locally convex set which includes all z_p given a certain measure of locality. The convex hull is a well known concept in computational geometry, in the text below we extend this concept with the notion of locality.

2.2.1 Globally convex hull

Let us first start with the definition for the globally convex hull $\mathcal{G}$ of $\mathcal{Z}$ (Preparata and Shamos 1985). $\mathcal{G}$ has the following requirements:

- $\mathcal{G}$ includes $\mathcal{Z}$
- $\mathcal{G}$ is convex
- $\mathcal{G}$ is minimal

¹ $\mathcal{H}(\mathcal{Z})$ is unique when the locality measure is specified.

The hull has a boundary, which consists of faces: polytopes. These polytopes have vertices, which are also the extreme points of both the hull and the data set. The set of vertices which describe the boundary of the hull $\mathcal{G}$, we denote by $\mathcal{V}(\mathcal{G})$.

In the two dimensional case, the convex hull has an enclosing contour (boundary) with a minimal length: a good analogy is an elastic band that is spanned around the points. In higher dimensions a hull with minimal surface is not necessarily convex: due to saddle surfaces and the like.

$\mathcal{G}$ is convex and includes $\mathcal{Z}$ means that any data point $\mathbf{z}_p$ in $\mathcal{Z}$ can be expressed as a convex combination of vertices $\mathbf{v}$ of $\mathcal{V}(\mathcal{G})$, whereby Carathéodory's theorem (Eckhoff 1993) states that this combination needs a maximum of $d + 1$ points:

$$\forall \mathbf{z}_p \in \mathcal{Z} : \mathbf{z}_p = \sum_{q=1}^{d+1} \lambda_q \mathbf{v}_q \quad \text{such that} \quad \mathbf{v}_q \in \mathcal{V}(\mathcal{G}), \lambda_q \in \mathbb{R}, \lambda_q \geq 0, \sum_{q=1}^{d+1} \lambda_q = 1 \quad (2-2)$$

In the two dimensional case this means that any data point is either contained in a triangle formed by three vertices of the convex hull, or is on a line between two of those vertices, or the data point is a vertex of the convex hull.

The hull $\mathcal{G}$ is the same as the set spanned by the convex combinations of the data points $\mathcal{Z}$, as that is the minimal convex set containing $\mathcal{Z}$.

$$\begin{aligned} \mathcal{G} &= \bigcup \left\{ \sum_{p=1}^{d+1} \lambda_p \mathbf{v}_p \mid \mathbf{v}_p \in \mathcal{H}, \lambda_p \in \mathbb{R}, \lambda_p \geq 0, \sum_{p=1}^{d+1} \lambda_p = 1 \right\} \\ &= \bigcup \left\{ \sum_{p=1}^{d+1} \lambda_p \mathbf{z}_p \mid \mathbf{z}_p \in \mathcal{Z}, \lambda_p \in \mathbb{R}, \lambda_p \geq 0, \sum_{p=1}^{d+1} \lambda_p = 1 \right\} \end{aligned} \quad (2-3)$$

For example in the two dimensional case, this means that the union of all triangles, lines and points formed with each combination of data points spans the convex hull.

The result of this is that all the points $\mathbf{v}_p$ of $\mathcal{V}(\mathcal{G})$ are data points $\mathbf{z}_p$ of $\mathcal{Z}$. Furthermore, to ensure convexity we should only include the data points that are in a boundary face of $\mathcal{G}$; these are the extreme points of $\mathcal{G}$. A boundary face of $\mathcal{G}$ is a hyperplane that has all the data

and all other vertices in one closed half space. Half spaces are convex, and the intersection of two convex sets is a convex set; therefore the set bounded by the vertices is a convex set. It follows that the convex set $\mathcal{G}$ with vertices $\mathcal{V}(\mathcal{G})$ is the minimal set of points that includes $\mathcal{Z}$. We can complete the definition of $\mathcal{V}(\mathcal{G})$ as follows:

$$\begin{aligned} \mathcal{V}(\mathcal{G}) &= \left\{ \mathbf{v}_p \mid \mathbf{v}_p \in \mathcal{Z} \right\} \\ &\text{such that} \\ &\|\mathcal{V}\| \text{ is minimal} \\ &\forall \mathbf{z}_p \in \mathcal{Z} : \mathbf{z}_p = \sum_{p=1}^{d+1} \lambda_p \mathbf{v}_p \mid \lambda_p \in \mathbb{R}, \lambda_p \geq 0, \sum_{p=1}^{d+1} \lambda_p = 1 \end{aligned} \tag{2-4}$$

For clustering, the globally convex minimal hull is not appropriate, as we would like to find multiple convex clusters. A globally convex hull would result in a single convex cluster. Therefore, we introduce the locally convex hull $\mathcal{H}$.

2.2.2 Locally convex hull

A locally convex set around data point $\mathbf{z}_p$ is the smallest convex set that contains the set of data points within a certain distance $e_{\max}$ of $\mathbf{z}_p$. The locally convex set $\mathcal{G}_{p,e_{\max}}$ around data point $\mathbf{z}_p$ is then defined as:

$$\begin{aligned} \mathcal{G}_{p,e_{\max}}(\mathcal{Z}) &= \bigcup \left\{ \sum_{q=1}^{d+1} \lambda_q \mathbf{z}_q \mid \mathbf{z}_q \in \mathcal{Z}, \lambda_q \in \mathbb{R}, \lambda_q \geq 0, \sum_{q=1}^{d+1} \lambda_q = 1 \right\} \\ &\text{such that} \quad \|\mathbf{z}_p - \mathbf{z}_q\| \leq e_{\max} \end{aligned} \tag{2-5}$$

In two dimensions, the locally convex inner set of a data point can be formed by taking the union of all the triangles and lines between that point and all points closer than $e_{\max}$.

The locally convex hull $\mathcal{H}$ is the minimal hull including the union of the locally convex sets around each of the N data points $\mathbf{z}_p$ in the data set $\mathcal{Z}$:

$$\mathcal{H} = \bigcup_{p \in N} \mathcal{G}_{p, e_{\max}}(\mathcal{Z}) \quad (2-6)$$

The set of vertices that form a face of the boundary set together with vertex $\mathbf{v}_p$, we call the connected set Conn_p of $\mathbf{v}_p$. Eq. 2-6 implies that any edge $e_{p,q}$ between two connected vertices $\mathbf{v}_p$ and $\mathbf{v}_q$ of $\mathcal{H}$ is not longer than the maximum edge length $e_{\max}$.

$$\|e_{p,q}\| = \|\mathbf{v}_p - \mathbf{v}_q\| \leq e_{\max} \forall \mathbf{v}_q \in \text{Conn}(\mathbf{v}_p) \quad (2-7)$$

The vertices $\mathcal{V}(\mathcal{H})$ of the locally convex hull $\mathcal{H}$ are defined as:

$$\mathcal{V}(\mathcal{H}) = \left\{ \mathbf{v}_p \mid \mathbf{v}_p \in \mathcal{Z} \right\} \quad (2-8)$$

such that

$\|\mathcal{V}\|$ is minimal

$$\begin{aligned} \forall \mathbf{z}_p \in \mathcal{Z} : \mathbf{z}_p &= \sum_{p=1}^{d+1} \lambda_p \mathbf{v}_p \mid \lambda_p \in \mathbb{R}, \lambda_p \geq 0, \sum_{p=1}^{d+1} \lambda_p = 1 \\ \forall \mathbf{v}_p \in \mathcal{V}(\mathcal{H}) : \|e_{p,q}\| &= \|\mathbf{v}_p - \mathbf{v}_q\| \leq e_{\max} \forall \mathbf{v}_q \in \text{Conn}(\mathbf{v}_p) \end{aligned} \quad (2-9)$$

Each data point of $\mathcal{Z}$ can still be obtained as a convex combination of vertices of the boundary of $\mathcal{H}$, but $\mathcal{H}$ is no longer globally convex.

Note, that many vertices of the different $\mathcal{G}_{p, e_{\max}}$ are interior in the full $\mathcal{H}$ and are therefore not vertices of the boundary of $\mathcal{H}$ and consequently are not in $\mathcal{V}(\mathcal{H})$.

2.2.3 Maximum edge length

Limiting the edge length results in concave regions in the hull, see Figure 2-2. The magnitude of $e_{\max}$ controls how concave the hull is. A smaller $e_{\max}$ results in smaller and more concave regions in $\mathcal{H}$. When the $e_{\max}$ becomes so small that the distance between a certain data point

and any other data point is larger than the given $e_{\max}$, the hull has disjoint regions. These disjoint regions can be used as –separable– clusters. A large $e_{\max}$ results in a convex hull encompassing all data points, a very small $e_{\max}$ results in a single data point per cluster, as each disjoint region of the hull encompasses only a single data point, because all edges are larger than $e_{\max}$.

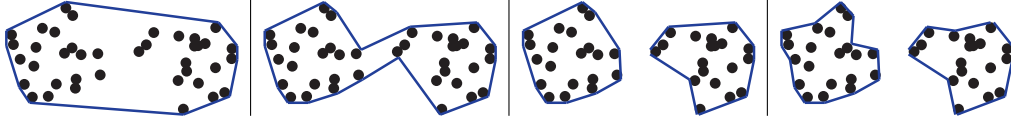


Figure 2-2: Examples of locally convex hulls with different $e_{\max}$, $e_{\max}$ decreases from left to right

$e_{\max}$ is the most important parameter in this objective, as it represents a trade-off between the compactness of the clusters versus the number of clusters. In this respect it is similar to the cut-off distance criterion in Single Linkage Hierarchical Clustering (SLHC) (Webb 2002).

Our locally convex hull adds shape information to the separable clusters, which can be used to even further partition the data, resulting in –possibly overlapping– convex clusters. More detail on this can be found in Chapter 3. When the convex clusters are to be found, the $e_{\max}$ is important as it is a parameter for the concaveness of the hull, and a hull with more concave regions results in more convex clusters. We have thus found a single parameter, that links the criterion in SLHC for separable clusters, with the criterion for convex clusters in our algorithm!

2.2.4 Summary of the objectives

Summarizing, the global objective is to find the target hull $\mathcal{H}$, a locally convex hull, which satisfies the following requirements:

R1.1 The hull is minimal

R1.2 The hull includes the data

$$\mathcal{Z} \subseteq \mathcal{H}$$

R1.3 The hull is locally convex

$$\mathcal{H} = \bigcup_{\mathbf{z}_p \in \mathcal{Z}} \mathcal{G}_{p, e_{\max}}(\mathcal{Z})$$

And this leads to the following requirements on the set of extreme points of the hull $\mathcal{V}$, which are equal to the set of vertices of the boundary of the hull:

R2.1 The vertices are points of the data set

$$\mathcal{V} \subseteq \mathcal{Z}$$

R2.2 The boundary encloses the data

$$\forall \mathbf{z}_p \in \mathcal{Z} : \mathbf{z}_p = \sum_{q=1}^{d+1} \lambda_q \mathbf{v}_q \mid \mathbf{v}_q \in \mathcal{V}, \lambda_q \in \mathbb{R}, \lambda_q \geq 0, \sum_{q=1}^{d+1} \lambda_q = 1$$

R2.3 The cardinality is minimal

$$\|\mathcal{V}\| \text{ is minimal}$$

R2.4 All edges are shorter than $e_{\max}$

$$(\|\mathbf{v}_p - \mathbf{v}_q\| < e_{\max} \forall \mathbf{v}_p, \mathbf{v}_q \in \mathcal{V}, \mathbf{v}_q \in \text{Conn}(\mathbf{v}_p))$$

Some of these requirements are equivalent, for example: given R2.3; R2.1 follows from R1.1, R2.2 is equivalent to R1.2 and R2.4 follows from R1.3.

We will focus on the following set of requirements:

R3.1 All edges are smaller than $e_{\max}$ (R1.3 and R2.4)

R3.2 The hull is minimal (R1.1 and R2.1)

R3.3 The boundary encloses the data (R1.2, R1.3, R2.2, and R2.3)

which cover all the requirements mentioned.

2.3 Soap Particle definition

Inspired by natural soap particles, the goal is to identify the hull associated with the global objective using artificial soap particles (SPs). The SPs follow movement rules based on local information to approximate each face of the optimal hull with one SP. We first describe the necessary parameters to define an artificial soap particle. We translate the global target hull into local objectives, which control the soap particle movements.

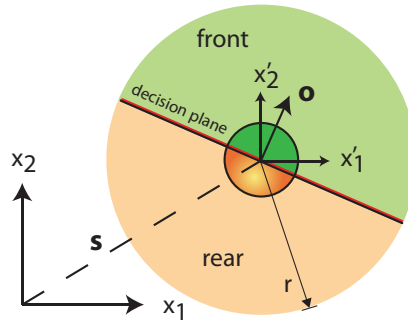


Figure 2-3: Soap Particle definition

Each SP has a position $s(k)$ and an orientation $o(k)$ at step k , see Figure 2-3. The position is defined as a vector in the global coordinate system, which equals the coordinate system of the data set.

The orientation is defined by a unit vector o_i in the local coordinate system of each SP, which is obtained by translating the global coordinate system with s_i .

Real world soap particles have only a limited neighbourhood around them in which the hydrophilic and hydrophobic forces are in effect. We can simulate this region by defining a neighbourhood $\mathcal{Z}_{nb}$ for our SPs. To avoid that the neighbourhood is orientation dependent, we use a spherical neighbourhood. The sphere is centred at the position of the SP, and has a radius of r .

An important measure, as we will see below, is the margin $m_i(x)$ between the SP s_i and a point x in $\mathbb{R}^d$. This can be any point: first we use the margin $m_i(z_p)$ to a data point z_p , but later we also use the margin $m_i(s_j)$ to another soap particle s_j . The margin $m_i(x)$ is the

projection of that point $\mathbf{x}$ onto the orientation vector $\mathbf{o}_i$, that is the signed distance between the affine subspace through the SP perpendicular to its orientation $\mathbf{o}_i$:

$$m_i(\mathbf{x}) = (\mathbf{x} - \mathbf{s}_i)^T \mathbf{o}_i \quad (2-10)$$

The concept of margin is illustrated in Figure 2-4.

Using the margin we can designate a front and a rear of the SP, which are separated by the local null space of the orientation vector. We call this null space the decision plane, as we later use this plane for labelling data points. A data point $\mathbf{z}_p$ at the front of SP $\mathbf{s}_i$ has a positive margin $m_{i,p}$ and a data point at the rear has a negative margin. A data point on the decision plane has a margin of zero.

front $m_{i,p}(\mathbf{z}_p) > 0$

rear $m_{i,p}(\mathbf{z}_p) < 0$

decision plane $m_{i,p}(\mathbf{z}_p) = 0$

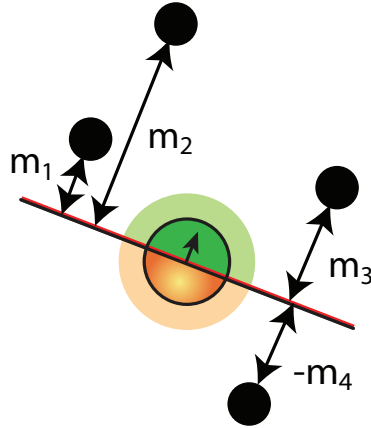


Figure 2-4: Example of margins

2.4 Local objectives

The global objective is to find the locally convex hull: the target hull (Eq. 2-8). From this global objective we derive multiple local objectives, with the goal that the problem can be converted into multiple smaller problems that can be solved in parallel. Each requirement on the target hull, can be translated into a local objective: the requirements on the final placement of the SP. The objective for the SP is to find one face of the target hull.

Limit edge length As each edge should be shorter than $e_{\max}$, we know that two vertices more than $e_{\max}$ apart, cannot be connected. Therefore faces have to be formed with vertices within $e_{\max}$ of each other. We can guarantee that no face is formed with data points too far from each other, by defining a neighbourhood (nb) with a radius r of $\frac{e_{\max}}{2}$ in which we consider data points to form a face. The set of data points within this neighbourhood are $\mathcal{Z}_{\text{nb}}$.

Minimal hull The requirement that the target hull is minimal, means that the hull should be as close as possible to the data points: any space between vertices and the enclosing hull results in a larger hull and therefore the hull cannot be minimal.

Enclose data The target hull should enclose all data, therefore no data points should be outside of the hull. For the SP this means that there should be no data points on the rear side of the SP.

global requirement	local requirement	
R3.1 limit edge length	neighbourhood	$r = \frac{e_{\max}}{2}$
R3.2 minimal hull	closest to data	$\min \left(\left\{ m_i(\mathbf{z}_{\text{nb},p}) \mid \mathbf{z}_{\text{nb},p} \in \mathcal{Z}_{\text{nb}} \right\} \right) = 0$
R3.3 enclose data	no data at the rear	$m_i(\mathbf{z}_{\text{nb},p}) \geq 0 \ \forall \ \mathbf{z}_{\text{nb},p} \in \mathcal{Z}_{\text{nb}}$

These local objectives do not necessarily ensure that the global objective is attained: while every face of $\mathcal{H}$ satisfies these local requirements, not all the faces that satisfy the local requirements have to be a face of $\mathcal{H}$. Therefore, the local objective are necessary but not sufficient conditions for the global objectives. Once we have defined the movement of the soap particles, we shall verify whether the algorithm results in finding the full optimal hull.

2.5 Soap Particle movement (local)

The local objectives can be obtained by creating the right movement rules for each soap particle.

global requirement	local requirement	movement rule
R3.1 limit edge length	neighbourhood	data selection
R3.2 minimal hull	closest to data	translation
R3.3 enclose data	no data at the rear	rotation

These movement rules together form the core elements of the SP movement algorithm. In the sections below we will provide more detail on these three rules.

For the algorithm we define the following interface:

Program Interface 2.A	
Input	Dataset Z
Parameters	• Maximum edge length $e_{\max}$ • Initial position $\mathbf{s}_0$ and orientation $\mathbf{o}_0$
Output	Position $\mathbf{s}$ and orientation $\mathbf{o}$ of the SP
Assumptions	• Data points are in general position

The assumption that the data points are in general position, which means that the number of intrinsic dimensions of the data is equal to the dimension of the data space. The parameter $e_{\max}$ determines the size of the neighbourhood of the SP, and the initial position and orientation determine the final placement of the SP, as the movement rules are deterministic.

2.5.1 Data selection

Only the data points within the neighbourhood of the SP are important. We implement the aforementioned spherical neighbourhood:

$$\mathcal{Z}_{\text{nb}}(\mathbf{s}) = \left\{ \mathbf{z}_p \in \mathcal{Z} \mid \|\mathbf{z}_p - \mathbf{s}\| < r \right\} \quad (2-11)$$

The subsequent rules are only based on these data points.

The p -th point in the neighbourhood $\mathcal{Z}_{\text{nb}}$ will be denoted by $\mathbf{z}_{\text{nb},p}$.

2.5.2 Rotation

The method for the position, which is described below, only brings the SP as close as possible to a single data point, while each face of the optimal hull has at least $d + 1$ (the number of dimensions plus one) vertices: data points in its decision plane.

When we rotate the SP such that the $d + 1$ data points with the lowest $m_{i,p}$ has the largest margin possible, a subsequent step by this margin – as done in the translation rule – ensures that the SP is as close as possible to the data points in its neighbourhood, and have $d + 1$ data points in its plane. Therefore, in the algorithm, we first rotate the particle, and then translate the particle.

Given the definition of the margin (Eq. 2-10) we can state the following optimization problem for the rotation:

$$\max_{\mathbf{o}} \min_{\mathbf{z}_{\text{nb},p} \in \mathcal{Z}_{\text{nb}}} (m_i(\mathbf{z}_{\text{nb},p})) = \max_{\mathbf{o}} \min_{\mathbf{z}_{\text{nb},p} \in \mathcal{Z}_{\text{nb}}} ((\mathbf{z}_{\text{nb},p} - \mathbf{s})^T \mathbf{o}) \quad \text{such that} \quad \|\mathbf{o}\| = 1 \quad (2-12)$$

We optimize the direction of $\mathbf{o}$, such that the margin is maximized.

The constraint on $\mathbf{o}$ (it should be a unit vector) and the 'min'-function make the optimization problem difficult to solve, however we can transform the problem into a quadratic programming problem (Eq. 2-15).

We can replace $\mathbf{o}$ with a non-unit vector to remove the constraint:

$$\mathbf{o} = \frac{\mathbf{o}^*}{\|\mathbf{o}^*\|} \quad (2-13)$$

which gives

$$m^* = \min \left((z_{\text{nb},p} - \mathbf{s})^T \frac{\mathbf{o}^*}{\|\mathbf{o}^*\|} \right) = \frac{\min ((z_{\text{nb},p} - \mathbf{s})^T \mathbf{o}^*)}{\|\mathbf{o}^*\|} \quad \forall z_p \in \mathcal{Z}_{\text{nb}} \quad (2-14)$$

then

$$\max_{\mathbf{o}^*} m^* = \max_{\mathbf{o}^*} \frac{\min ((z_p - \mathbf{s})^T \mathbf{o}^*)}{\|\mathbf{o}^*\|}$$

is equivalent to

$$\min_{\mathbf{o}^*} \|\mathbf{o}^*\| \quad \text{such that} \quad \min ((z_p - \mathbf{s})^T \mathbf{o}^*) = 1 \quad \text{or any positive constant}$$

substituting $\|\mathbf{o}^*\|$ by $\mathbf{o}^{*T} \mathbf{o}^*$ does not change the solution

$$\min_{\mathbf{o}^*} \mathbf{o}^{*T} \mathbf{o}^* \quad \text{such that} \quad \min ((z_p - \mathbf{s})^T \mathbf{o}^*) = 1$$

finally replace the 'min' by an inequality

$$\min_{\mathbf{o}^*} \mathbf{o}^{*T} \mathbf{o}^* \quad \text{such that} \quad (z_p - \mathbf{s})^T \mathbf{o}^* \geq 1 \quad (2-15)$$

which is in the standard form of a quadratic programming problem (Gill, Murray, and Wright 1981). We can retrieve $\mathbf{o}$ with Eq. 2-13.

2.5.3 Translation

To obtain the minimal hull, the hull should be very tight around the data set, and each face of the hull should be as close as possible to the data points, without any data points at the rear. This means that we need to move the SP close to the data, with all the data in the neighbourhood in front of the SP. Therefore, if the SP has data points behind it; it should move backwards, until all the data points are in front of the SP.

Once we know the minimal² margin $m_{i,\min}$ we can translate the SP exactly by that amount (Eq. 2-17), so that the SP is as close as possible to the data, given that we do not allow

²The minimal element is not larger than any other element in the set: e.g. closest to negative infinity.

rotation.

$$m_{i,\min} = \min(m_i(z_p)) \quad (2-16)$$

$$\mathbf{s}_i(k+1) = \mathbf{s}_i(k) + m_{i,\min}(k) \quad (2-17)$$

The optimization problem for the rotation can become ill-defined –when implemented on a computer– due to floating point inaccuracies, when the data points are located exactly in the decision plane ($m = 0$). To avoid this situation, in the translation step we keep a very small margin ϵ between the vertices and the decision plane:

$$\mathbf{s}_i(k+1) = \mathbf{s}_i(k) + m_{i,\min}(k) + \epsilon \quad (2-18)$$

2.5.4 Iteration and stopping criterion

The step involving the translation of the SP, changes the position of the SP. As the position is used in the calculation of the neighbourhood, this neighbourhood is changed and has to be calculated again. As this can invalidate our previous conclusions, we have to iterate the rotation and translation too.

When the change in position does not result in a change in the neighbourhood, the SP does not move after the above steps. When the SP is in the same position and orientation as the previous iteration, the SP is converged. This is the stopping criterion.

After a SP has stopped moving, it is on the optimal hull $\mathcal{H}$, when the following two conditions are true:

C3.1 On the hull, the SP movement rules do not change the position of the SP;

C3.2 Not on the hull, the SP movement is non-zero.

Convergence needs a third condition:

C3.3 There are no cycles present in the particle movement.

Condition C3.1 holds. On the hull, all the data within the neighbourhood is on the front of the SP, and the vertices are intersecting the decision plane, which avoids translation and rotation.

Condition C3.2 does not always hold. The $e_{\max}$ is defined from vertex to vertex, while in the data selection we use the smallest neighbourhood, thereby ignoring possible vertices. See Figure 2-5 for an example, where v_1 and v_2 are within $e_{\max}$ of other data points in the neighbourhood, and therefore should not be at the rear of the soap particle.

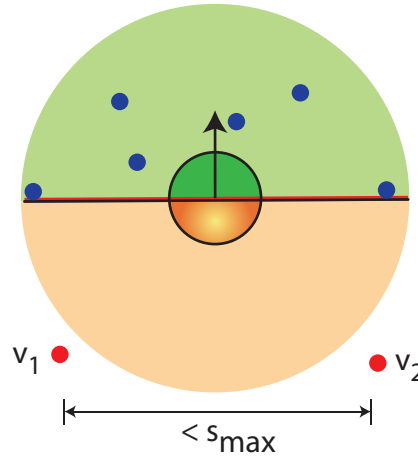


Figure 2-5: Incorrectly converged SP, as v_1 and v_2 are ignores

Currently we ignore this situation, however it should be handled in the future. By verifying the validity of all the data points in the neighbourhood of identified vertices, we can recognize this situation and take appropriate action.

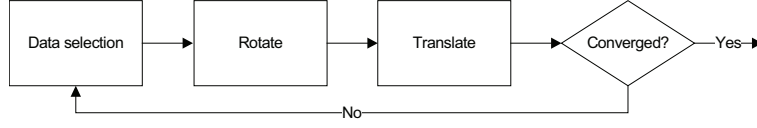
Finally, condition C3.3 can be verified in future research. Empirical evidence gathered by our implementation suggests that cycles do not occur.

Soap Particle movement algorithm

The complete algorithm is presented below:

Algorithm 2.A (*Soap Particle movement*)

Given a data set find a face of the target hull



A-1 [Data selection] Select all the data points within the neighbourhood with radius r

$$\left(\mathcal{Z}_{\text{nb}} \leftarrow \left\{ z_p \mid z_p \in \mathcal{Z} \right\} \quad \text{such that} \quad \|z_p - \mathbf{s}\| < \frac{\epsilon_{\text{max}}}{2} \right)$$

A-2 [Rotate] No data points should be at the rear

$$\left(\min_{\mathbf{o}^*} \mathbf{o}^{*T} \mathbf{o}^* \quad \text{such that} \quad (\mathbf{z}_p - \mathbf{s})^T \mathbf{o}^* \geq 1 \right)$$

A-3 [Translate] Move the SP along its normal until the nearest data point

$$(\mathbf{s} \leftarrow \mathbf{s} + m_{\text{min}})$$

A-4 [Converged?] If new position is equal to previous, then stop, else go to step **A-1** ■

2.6 Hull finding algorithm (global)

Each SP only estimates a single face of the target hull. Using more SPs, we can identify more faces. When sufficient SPs are introduced which are spread well, each face of the optimal hull is represented with at least one SP.

Program Interface 2.B

Input Dataset Z

Parameters

- Maximum edge length $e_{\max}$
- Number of soap particles n

Output Hull $\mathcal{H}$, defined by a set of soap particles $\mathcal{S}$

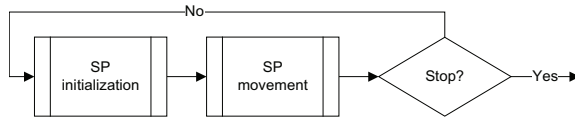
Assumptions

- Data points are in general position
- Low to medium dimension ($d \leq 10$)
- Sufficient number of soap particles n

The resulting algorithm:

Algorithm 2.B (*Soap Particle hull finding*)

Given a data set find the target hull



B-1 [New SP] Create a new SP

B-2 [SP movement] Move the SP until it has converged

B-3 [Stop?] If the required number of SP has been created, then stop, else go to step **B-1**

■

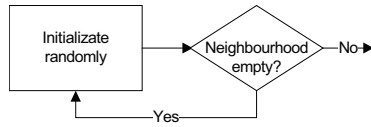
2.6.1 Initialization

For now, we initialize the SPs randomly; this ensures that when more and more SPs are introduced, in the limit every location is visited. Therefore, when sufficient soap particles are introduced, each face of the optimal hull should be approximated with a soap particle.

A random initialization could mean that the soap particle position is infeasible: it has an empty neighbourhood. In this case we need to reinitialize. Assuming $\mathcal{Z}$ is not an empty set, eventually an initialization should result in a non-empty neighbourhood. We present the random algorithm below:

Algorithm 2.C (*Soap Particle initialization*)

Initialize a soap particle



C-1 [Initialize] Initialize the s randomly

($s \leftarrow \text{random}, o \leftarrow \text{random}$)

C-2 [Empty neighbourhood?] If the neighbourhood $\mathcal{Z}_{\text{nb}}$ is empty, go back to step **C-1**

(if $\mathcal{Z}_{\text{nb}} = \emptyset$ then **C-1**) ■

In Section A.2 some other initialization methods are discussed.

2.6.2 SP Movement

Each SP moves according to the same rules as mentioned above in Section 2.5. These are deterministic movement rules, therefore the final position and orientation of each SP is fully determined after their initialization.

2.6.3 Stopping conditions

Ideally, each SP should estimate one face of the target hull. However, we do not know a fast method to determine the number of faces of the target hull in advance. Furthermore, we cannot guarantee that each face of the target hull has only one soap particle on it. Therefore, the exact number of SPs needed is difficult to predict for general problems. Currently, we define in advance the number of SPs, in the recommendation more advanced methods are discussed.

2.7 Validation

We have implemented the algorithm in MATLAB, and we have created some artificial data sets. Even with artificial data sets, it is difficult to verify whether all the soap particles have converged onto a face of the target hull, and to verify whether all face have been found, therefore we use the QHull algorithm of Barber, Dobkin, and Huhdanpaa (1996), which is guaranteed to output the globally convex hull, to compare the SP hull finding algorithm with. The QHull algorithm cannot handle the $e_{\max}$, and only gives the globally convex hull. Due to these limitations of the QHull algorithm, we mainly create data sets with an intrinsic convex hull; but to evaluate the hulls with concave regions, we have made one data set with an intrinsic concave hull by adding two intrinsic convex sets. The QHull algorithm will then be applied to both intrinsic convex sets separately.

2.7.1 Computer experiments

We test the algorithm on four different artificial data sets, with three different numbers of the dimensions, on five performance measures.

The data sets:

- Spherical (uniformly distributed data points within a hypersphere with radius l)
- Square (uniformly distributed data points within a hypercube with sides l)
- Line (uniformly distributed data points along a line with length l)
- 2 lines (uniformly distributed data points along two lines with length l at an angle with each other)

We create these data sets in 2, 3, and 5 dimensions; so that we have some idea of the scalability of the algorithm.

The performance measures, based on the ideal number $n_{\mathbf{v},\text{ideal}}$ of supporting vertices $\mathbf{v}_{\text{ideal}}$, the found number $n_{\mathbf{v},\text{found}}$ of supporting vertices $\mathbf{v}_{\text{found}}$, and Pratt's penalty parameter α :

- convergence (mean number of iterations per SP, lower is better)
- validity ($\|\mathbf{v}_{\text{ideal}} - \mathbf{v}_{\text{found}}\|$, 0 is best)
- completeness ($\frac{n_{\mathbf{v},\text{found}}}{n_{\mathbf{v},\text{ideal}}}$, 1 is best)
- redundancy ($\frac{n_{\mathbf{v},\text{found}} - \text{unique}(n_{\mathbf{v},\text{found}})}{n_{\mathbf{v},\text{ideal}}}$, 0 is best)
- Pratt's FoM (Pratt 2001) ($\frac{1}{\max n_{\mathbf{v},\text{found}}, n_{\mathbf{v},\text{ideal}}} \sum n_{\mathbf{v},\text{found}} \frac{1}{1 + \alpha^T (\|\mathbf{v}_{\text{ideal}} - \mathbf{v}_{\text{found}}\|)^2}$, 1 is best)

The $\mathbf{v}_{\text{ideal}}$ are determined with the QHull algorithm. For the 2 lines, the data is first divided in convex sets, and the QHull algorithm is applied to both separate sets, as the QHull algorithm only finds convex hulls.

We repeat each test 10 times and average the results.

2.7.2 Results of the experiments

In this section the mean results of the mentioned experiments are displayed.

The following parameters have been chosen for the tests:

Number of datapoints $N = 250$

Length $l = 1$

Pratt's penalty parameter $\alpha = 1$

When the algorithm is used in a working environment, the number of soap particles would be tuned because the required number is dependent on the maximum edge length, the number of dimensions, and the data set; but to show the influence of these parameters, in these test it is kept constant. The following values for the tuning parameters are used:

Number of soap particles $n = 25$

Maximum edge length $e_{\text{max}} = 0.3$

$d = 2$	Convergence	Validity	Completeness	Redundancy	FoM
Sphere	3.0080	0.0076	1.0375	1.8542	0.9996
Square	3.0560	0.0145	1.6583	6.9750	0.9992
Line	3.0320	0.0200	1.9727	19.872	0.9987
2 lines	3.2640	0.0103	1.6280	3.1320	0.9995

Table 2-1: Test result for the data sets with data in 2 dimensions

$d = 3$	Convergence	Validity	Completeness	Redundancy	FoM
Sphere	3.0680	0.0487	0.7532	0.0000	0.7480
Square	3.0840	0.0254	0.9863	0.0471	0.9841
Line	3.2800	0.0096	1.1372	0.3953	0.9997
2 lines	3.2280	0.0080	0.7615	0.0462	0.7613

Table 2-2: Test result for the data sets with data in 3 dimensions

We present the results of these experiments in three tables for each number of dimensions: the results for data in $\mathbb{R}^2$ are presented in Table 2-1 and Figure 2-6, for the 3 dimensional data in Table 2-2, and for the 5 dimensional data in Table 2-3.

2.7.3 Conclusions of the experiments

From the test results we can see that convergence is generally quite quick, within 3–4 iterations a SP is at its final position. The validity of the SPs is close to zero in the tests, which is favourable. The reason the validity is not completely zero, is more due to the fact that the

$d = 5$	Convergence	Validity	Completeness	Redundancy	FoM
Sphere	3.0360	0.0566	0.1816	0.0000	0.1798
Square	3.1280	0.0460	0.3359	0.0000	0.3342
Line	3.2200	0.0215	0.4616	0.0000	0.4610
2 lines	3.1720	0.0162	0.2762	0.0000	0.2758

Table 2-3: Test result for the data sets with data in 5 dimensions

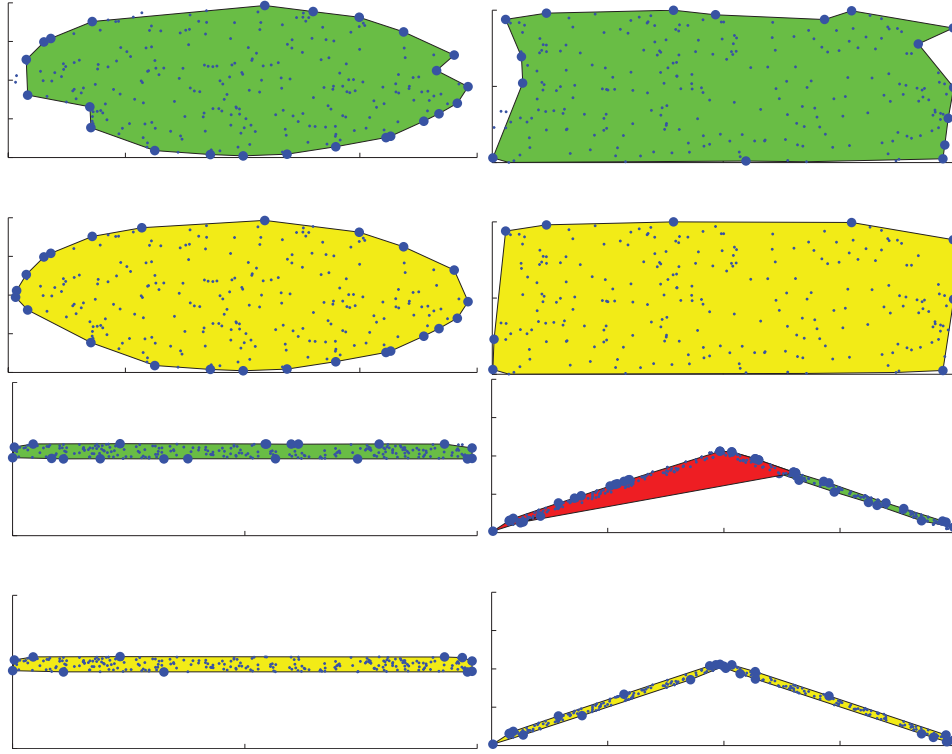


Figure 2-6: The contours found by the SP Hull finding algorithm (upper) versus the contours found by QHull (lower/yellow)

QHull algorithm does not take into account the $e_{\max}$ than due to incorrectly converged SP as mentioned in Section 2.5.4. The completeness is higher than 1 for the sphere and square due to the effects mentioned in the test set-up. If we would find a way to ensure a completeness of 1, we can remove the need to specify the number of SPs up front. Redundancy is high for the subspace and the line, because these data sets are lower dimensional and can do with less than the provided number of SP.

2.8 Conclusions and recommendations

2.8.1 Conclusions

The goal of the SP Hull Finding algorithm is to find the locally convex hull of a data set; from the experimental results we can conclude that the SP Hull Finding algorithm does find a hull, and that hull has concave regions. Therefore, generally the algorithm has achieved its goal. The experiments show that convergence is generally quick, we gives us more confidence in the convergence to which we had some reservations in Section 2.5.4. The experimental results show that validity of the position of the SPs is good: the SP Hull Finding algorithm gives solutions similar to the optimal solution of the QHull algorithm, with the added advantage that it can identify intrinsic convex substructure in data sets.

There can be made improvements in the completeness of the hull description and the redundant multiple SPs for each face: especially when there are more SPs introduced, than there are faces of the target hull. However, the performance of the algorithm can ultimately only be proven when the results are used for clustering, as we will discuss in Chapter 3 and finally for system identification, which we will discuss in Chapter 4.

2.8.2 Recommendations

From the analysis of the algorithm above, some possible improvements surfaced.

SP convergence The SPs do not always converge to the correct location, as they ignore important data points. This is due to the differences in the global and local objectives. The theoretical background of this is shown in Figure 2-5.

Initialization The SPs are initialized randomly, in the entire search space. Especially with data in higher dimensions, large parts of the search space are empty. This means that the freshly initialized SPs often have an empty neighbourhood and need to be reinitialized, which is computationally inefficient.

Redundant SPs Quite often SPs end up at the same face as a prior SP. This can be seen in the high redundancy rate in some of the tests. When a soap particle is redundant, the calculations for this soap particle are unneeded and increase the computational complexity.

Completeness Currently, we cannot guarantee the data is fully described.

In Appendix A, some recommended approaches and theory for solving these problems and thereby improving the algorithm are given.

Chapter 3

Soap Particle Clustering

3.1 Introduction

In Chapter 2 we have found a description of the locally convex hull around the data using the position and orientation of a set of artificial soap particles; in this chapter we will find the largest convex sections within this hull, which results in a partitioning of the data into convex clusters. This combined procedure is called Soap Particle clustering.

The hull finding stage (Chapter 2) already gives a partitioning when the data are separable: that is when the distance between two clusters is larger than $e_{\max}$. However, while the boundaries and the SPs describing them are known, in that case the boundaries and SPs are themselves not yet labelled. We can label them either with the extended Algorithm A.A, which initializes new SPs near the boundary of one cluster at a time, or with Single Linkage Hierarchical Clustering (Webb 2002) on the SPs. The next step then, would be to label the data points themselves: as the goal of clustering is to determine the membership degree to

the clusters for each data point. However, the method to do that from labelled SPs is not straightforward. Furthermore, a labelling based on the separability does not help us with identifying affine subsystems, therefore we need a separate stage to label the SPs and to label the data points using that cluster description, which are the subjects of Section 3.2 and Section 3.3 respectively.

3.2 Training

In this section we will describe the objective of the cluster training stage and the algorithm itself.

3.2.1 Objective

The objective of the training stage is to create a usable cluster description from the SPs. The question then remains: what is a usable cluster description? Let us focus on the different words: usable, cluster, and description.

A *cluster* is a group of data points which have a greater similarity between them, than a cluster has with the other data points. Often this similarity is related to some distance measure: for example in *c*-means to the cluster centre or in Hierarchical Clustering to one or more data points of the cluster. This inspired us to look at the margin (Section 2.3) as a similarity measure.

Furthermore, we looked at the final goal of the clusters: they should be *usable* and describe the affine subsystems present in the data. Therefore, our clusters should have an affine shape. The decision planes of our SPs have an affine shape and could describe an affine subsystem in the data. However, then we would have an excess of subsystems: one for each SP. For high usability, we would like to have as few as possible subsystems, while still describing the data well. Therefore, we should look at a way to combine the different soap particles that describe the same subsystem. One SP can only describe one side of the affine subsystems, for example

the bottom: it then has all the data above it. A second SP is then needed to describe the top, and more are needed to describe to the sides. How can we select these SPs? All the data belonging to the particular subsystem above the first SP has a positive margin to that SP: it is visible to that SP. So data belonging to the cluster, which is described by a particular SP is visible to that SP. The SP that describes the top of the subsystem, also is visible to the SP that describes the bottom. And the SPs that describe the sides is visible to both the top and the bottom SP and to each other. So then we arrive at the conclusion: SPs describing the same cluster, are visible to each other: they have a positive margin between each other (Eq. 3-1). And the data between the SPs has a positive margin to all of them (Eq. 3-4). This means that our objective is to find the largest convex subsets described by the SPs! This is illustrated in Figure 3-3.

The set of soap particles $\mathcal{S}_l$ that describe cluster $\mathcal{C}_l$ all have a positive margin m with respect to each other:

$$\mathcal{S}_l \subseteq \mathcal{S} \quad \text{such that} \quad m(s_i, s_j) \geq 0, m(s_j, s_i) \geq 0 \forall s_i, s_j \in \mathcal{S}_l \quad (3-1)$$

We should note, that with Eq. 3-1, it is required that the SP is located on the part of the face that is visible to all other SPs of the same convex subset. This is not necessarily true, as the vertices of the convex subsets are no longer necessarily vertices of the original hull. Sometimes only part of the face is used to complete the boundary of the subset, for an example see Figure 3-1. This situation is not allowed by Eq. 3-1. When we include SPs that have a margin larger than $-e_{\max}$ compared to the other SPs, we can include that particular situation.

In real system identification problems, often curves are present. As curves are concave, curves can lead to many small clusters, while larger clusters would result in a sufficient representation of the system, as illustrated in Figure 3-2. Taking a larger $e_{\max}$ parameter allows more concaveness, but also results in a coarser description of the convex regions. To this cause, we introduce a generalization parameter g_{train} , which allows more concaveness, without the coarser description of the convex regions:

$$\mathcal{S}_l \subseteq \mathcal{S} \quad \text{such that} \quad m(s_i, s_j) \geq -g_{\text{train}}, m(s_j, s_i) \geq -g_{\text{train}} \forall s_i, s_j \in \mathcal{S}_l \quad (3-2)$$

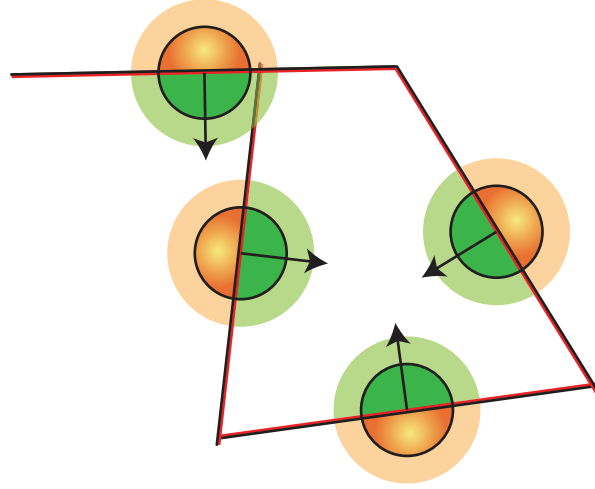
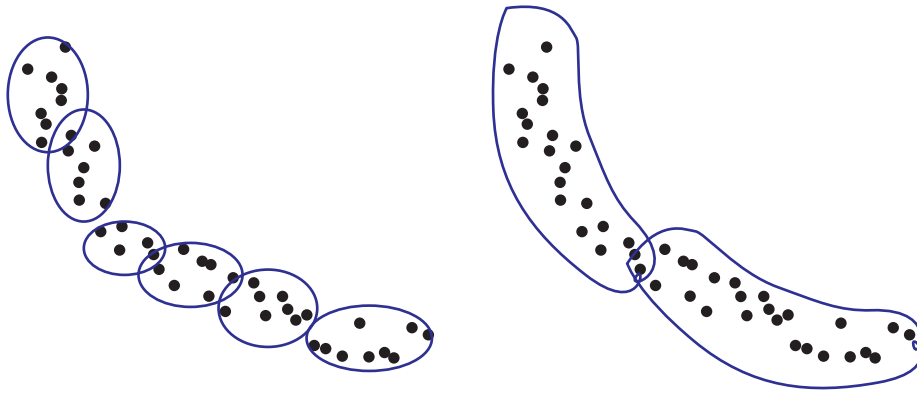


Figure 3-1: Only part of the face is used to complete the boundary of the subset



Only convex clusters

Slightly nonconvex clusters

Figure 3-2: Often slightly nonconvex clusters are preferred

The naive choice of g_{train} should be equal to e_{max} , due to the partial boundaries mentioned above.

Finally, the *description* we create, should allow us to label the data points, and provide a membership degree for each data point to each cluster. As we know the data between the SPs of a particular cluster has a positive margin to each of those SPs, so labelling of the data points should be straightforward, see Section 3.3.

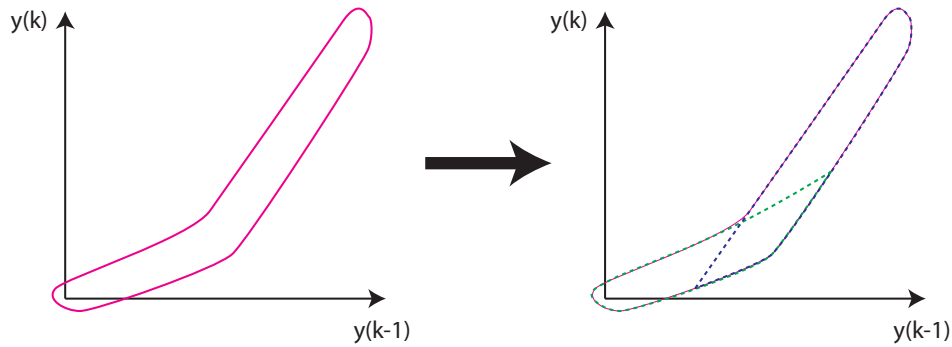


Figure 3-3: The second stage: make convex clusters based on the hull of stage 1

3.2.2 Input, output, and assumptions

A summary of all the inputs and outputs is presented in Interface 3.A.

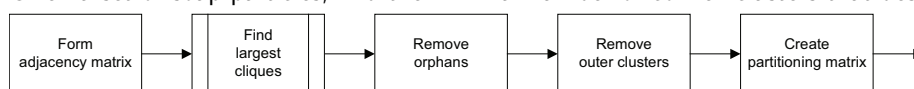
Program Interface 3.A	
Input	Set of soap particles $\mathcal{S}$
Parameters	Generalization parameter g_{train}
Output	Each SP has one or more clusters assigned to it
Assumptions	None

3.2.3 Soap Particle Cluster Training

We cast our objective into the maximum clique problem (Bomze et al. 1999). The cliques correspond with the convex regions, and therefore clusters. In Algorithm 3.A we outline the algorithm.

Algorithm 3.A (*Soap Particle Cluster Training*)

Given a set of soap particles, find the minimal number of convex clusters that describe the data



A-1 [Form adjacency matrix]

(**if** $m(s_i, s_j) \geq -g_{\text{train}}$ **and** $m(s_j, s_i) \geq -g_{\text{train}}$ **then** $A(i, j) \leftarrow 1$)

A-2 [Find largest cliques]

$(\mathcal{C}_l \stackrel{\text{clique}}{\Leftarrow} A)$

A-3 [Remove orphans]

$(\mathcal{C}_{\text{new}} = \{\mathcal{C}_l \in \mathcal{C} \mid \|\mathcal{C}_l\| > 1\})$

A-4 [Remove outer clusters]

$(\mathcal{C}_{\text{new}} = \{\mathcal{C}_l \in \mathcal{C} \mid m(s_i, s_j) < -g_{\text{train}} \forall s_i \in \mathcal{C}_l, s_j \in \mathcal{S}, s_j \notin \mathcal{C}_l\})$

A-5 [Label SPs]

(**if** $s_i \in \mathcal{C}_l$ **then** $U(l, i) = 1$) ■

Adjacency matrix formation

We represent the margins between the SPs in an adjacency matrix A . The n SPs result in an n by n matrix, in which the entry on column i and row j has a value of 1 when SP s_i and SP s_j have a margin larger than $-g_{\text{train}}$ to each other, and has a value of 0 otherwise:

$$A(i, j) = \begin{cases} 1 & m(s_i, s_j) \geq -g_{\text{train}} \text{ and } m(s_j, s_i) \geq -g_{\text{train}} \\ 0 & \text{otherwise} \end{cases} \quad (3-3)$$

Maximal cliques

Maximal cliques algorithms find the largest cliques in the adjacency matrix. A clique is a set of rows and columns (nodes) that are fully connected with each other, that means their submatrix consists of all ones. This corresponds to sets of SPs that all have a sufficient margin between each other. The maximum cliques problem¹ is NP-hard (Pardalos and Xue 1994), however, and current algorithms become infeasible when n is larger than about 15.

For our objective we actually do not need to have the maximum clique: we need the minimal number of cliques. Therefore, we use an clique finder which returns a set of maximal cliques

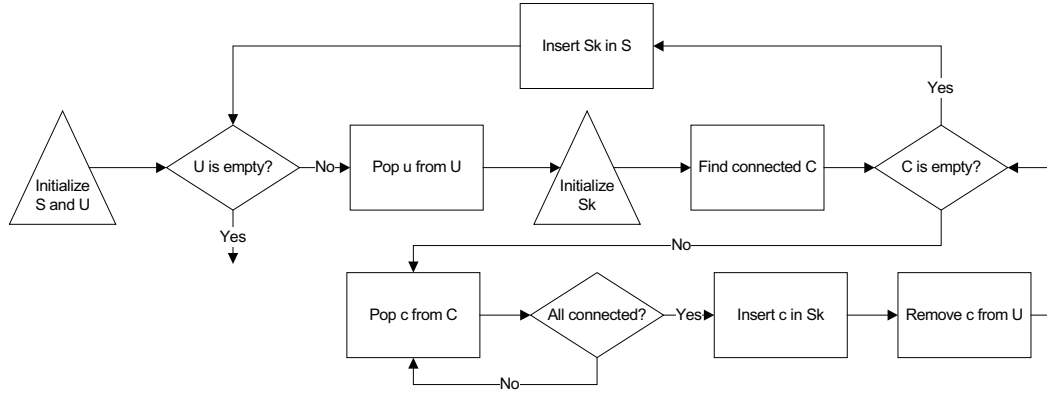
¹Note the difference between the maximum clique problem and finding a maximal clique. A maximal clique is a clique that cannot be larger, and the maximum clique is the largest of these maximal cliques.

which span the full set with polynomial complexity. This makes the clique finder feasible for at least 100 SPs (tested).

An outline of this clique finder algorithm is given in Algorithm 3.B.

Algorithm 3.B (*Clique finder*)

In an adjacency matrix find largest cliques such that each node is assigned to a clique



B-1 [Initialize S and U] The set of subgraphs S starts empty; all nodes N are unassigned U , as they do not belong to a clique yet
 $(S \leftarrow \emptyset, U \leftarrow N)$

B-2 [U is empty?] If U is empty, then stop
 (if $U = \emptyset$ then stop.)

B-3 [Pop u from U] Remove the first node from U and store in u
 $(u \leftarrow U)$

B-4 [S_k initialization] u is the first node of the current subgraph S_k
 $(S_k \leftarrow \{u\})$

B-5 [Find C] Put all nodes connected to u in the set of connected nodes C
 $(C \leftarrow \text{Connected}(u))$

B-6 [C is empty?] If C is empty, then continue, else go to **B-8**
 (if $C \neq \emptyset$ then **B-8**)

B-7 [Insert S_k in S] Insert subgraph S_k in S and go to **B-3**
 $(S_k \Rightarrow S)$

B-8 [Pop c from C] Remove the first node from C and store in c
 $(c \leftarrow C)$

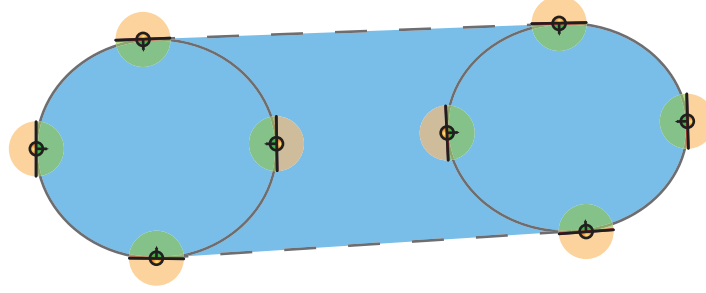


Figure 3-4: Two clusters together also form a larger redundant cluster

B-9 [All connected?] If all nodes g in the subgraph S_k are connected to c , then continue, else go to **B-8**

(if $\neg \forall g \in S_k : g \in \text{Connected}(c)$ then **B-8**)

B-10 [Insert c in S_k] Insert connected node c in subgraph S_k

($c \Rightarrow S_k$)

B-11 [Remove c from U] c is now assigned to a clique and is removed from U and go to **B-6**

($U = U \setminus c$ then **B-6**) ■

Orphans

As mentioned in Section 2.8.2, the hull description is not always complete. Outliers potentially could attract a single SP, these SPs we call orphans. Single SP do not fully describe a cluster, and are therefore removed.

Outer clusters

The SPs on the outer boundaries of two clusters all have a positive margin to each other, and are therefore found in the prior steps; an example of this situation is depicted in Figure 3-4. These clusters are unneeded and are thus removed. Any cluster that has a SP inside it, will be removed.

SP Labelling

We now have a list of SPs for each cluster, but we also need a list of clusters for each SP: the SP need to be labelled. It is trivial that all the SPs can be labelled by running through the lists for each cluster.

3.3 Labelling

3.3.1 Objective

The objective of the labelling is to determine the membership degree $\mu_{l,p}$ of all the data points z_p to all the clusters $\mathcal{C}_l$. We can determine the data points that belong to each cluster $\mathcal{C}_l$, as those data points have a positive margin m to all the soap particles $\mathcal{S}_l$, that describe the cluster:

$$\mathcal{C}_l = \left\{ z_p \mid z_p \in \mathcal{Z}, m(s_l, z_p) \geq 0, \forall s_l \in \mathcal{S}_l \right\} \quad (3-4)$$

However, this labelling is not complete. Cluster membership degrees have to fulfil certain requirements: they should be larger than 0, and should sum to 1 for fuzzy labelling. When we assign each data point fully to each cluster it belongs to and check whether these requirements are satisfied, we find in general they are not fulfilled. There can be data points which do not appear in any of the clusters, and have a membership degree of 0, which violates the first requirement. And there can be data points which appear in more than one cluster: which violates the second requirement.

Under the assumption that the data descriptor stage (Chapter 2) performed well, data points that were present in that stage should all be inside a contour found in that stage. As such, they are certain to be labelled in this stage. Only data points that were not present in the data descriptor stage could be outside all contours. When this happens, we will call these data points outliers and do not assign them to any cluster, as apparently the training set did not provide information on these data points. When a lot of data points are not assigned, this is a clear indication to the user of the algorithm, that the training set for the data descriptor stage was insufficient. It is convenient to include some of the nearby outliers, as training sets generally have a somewhat different distribution than the validation set. To this cause, we specify the generalization parameter g_{label} .

The case that a single data point ends up in more than one cluster, is common. Only when the training data set had separable convex clusters, this will not happen. One way to solve

the membership degrees sum to be larger than 1, is to assign equal membership degrees to each of the clusters: a data point z_p belonging to two clusters, then will get a membership degree $\mu_{l,p}$ of 0.5 to each of the clusters:

$$\mu_{l,p} = \frac{1}{\sum_p \bar{\mu}_{l,p}} \quad \text{where} \quad \bar{\mu}_{l,p} = \begin{cases} 1 & \text{if } z_p \in \mathcal{C}_l \\ 0 & \text{otherwise} \end{cases} \quad (3-5)$$

Another option is to assign fuzzy membership degrees: a data point can belong more to the one cluster, than to the other cluster. In fuzzy c -means, this is solved by looking at the distance to the cluster centre: the data point has a higher membership degree to the closer cluster centre. We do not have cluster centres. We do have margins: a low margin to the cluster border means that the data point is nearly outside of the cluster. A low margin therefore indicates a lower membership degree to that particular cluster:

$$\mu_{l,p} = \frac{\bar{\mu}_{l,p}}{\sum_p \bar{\mu}_{l,p}} \quad \text{where} \quad \bar{\mu}_{l,p} = \begin{cases} \min m(s_{l,i}, z_p) \forall s_{l,i} \in \mathcal{S}_l & \text{if } z_p \in \mathcal{C}_l \\ 0 & \text{otherwise} \end{cases} \quad (3-6)$$

The $\bar{\mu}$ is the not normalized membership degree, that is the membership degree when we do not require the membership degrees to sum to 1 (similar to a possibilistic membership degree).

3.3.2 Input, output, and assumptions

A summary of the inputs, output and a parameter is given below:

Program Interface 3.B

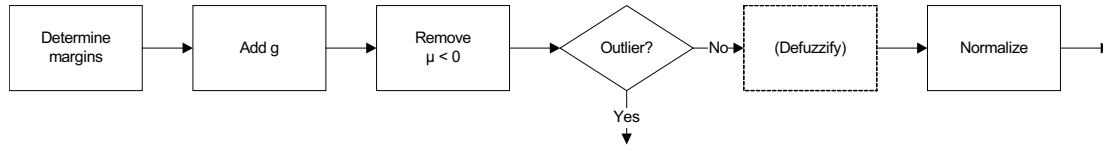
Input	Set of labelled soap particles $\mathcal{S}_l$ and a data set $\mathcal{Z}$
Parameters	• Generalization parameter g_{label}
Output	Each data point z_i is assigned to one or more clusters
Assumptions	• Data points outside clusters are outliers (and can be unlabelled)

3.3.3 Soap Particle Labelling Algorithm

The algorithm for labelling data points is presented below:

Algorithm 3.C (*Soap Particle Labelling*)

Given a set of labelled soap particles, provide the cluster membership degrees for a datapoint



C-1 [Calculate margin] For each of the c clusters, calculate the margins to the SPs and store the lowest as cluster membership degree

$$\left(\bar{\mu}_{l,p} \leftarrow \min m(s_{l,i}, z_p) \forall s_{l,i} \in \mathcal{S}_l \quad \overline{M}_p \leftarrow \left\{ \bar{\mu}_{l,p} \mid l = 1, \dots, c \right\} \right)$$

C-2 [Add generalization] Add the generalization factor to the membership degrees

$$(\bar{\mu}_{l,p} \leftarrow \bar{\mu}_{l,p} + g_{\text{label}})$$

C-3 [Remove negative membership degrees] Set the negative membership degrees to 0

$$(\text{if } \bar{\mu}_{l,p} < 0 \text{ then } \bar{\mu}_{l,p} \leftarrow 0)$$

C-4 [Outlier?] If all membership degrees are 0, then the data point is an outlier, so stop.

$$(\text{if } \sum_{l=1}^c \bar{\mu}_{l,p} = 0 \text{ then stop.})$$

C-5 [(Defuzzify)] If a crisp partitioning is required, then defuzzify

$$\left(\bar{\mu}_{l,p} \leftarrow \begin{cases} 1 & \text{if } \bar{\mu}_{l,p} \geq \mu \forall \mu \in \overline{M}_p \\ 0 & \text{otherwise} \end{cases} \right)$$

C-6 [Normalize] Ensure that the sum of the membership degrees is 1

$$\left(\mu_{l,p} \leftarrow \frac{\bar{\mu}_{l,p}}{\sum_p \bar{\mu}_{l,p}} \right) \blacksquare$$

In the following sections we will highlight some of the more involved steps in the algorithm.

Outliers

Data points that have only negative membership degrees even after adding the generalization parameter, lie far outside of all the clusters, and therefore cannot be reliably labelled. We therefore choose to not assign a membership degree to these clusters and let the user make a judgement on the approach to deal with these outliers. A possible measure would be to repeat the training stage, as apparently the clusters do not describe the data well, another measure would be to increase the generalization factor, or to assign a equal membership degree of the data point to all clusters.

Defuzzification

When it is needed to provide a crisp partitioning, defuzzification can be done by assigning the data point to the cluster to which it has the largest membership degree.

Normalization

Membership degrees should be between 0 and 1, therefore the membership degrees are normalized using Eq. 3-6. For the validation section, we will not normalize the membership degrees: this leads to data points that are partially member of one or more clusters, and that are partially outlier.

3.4 Validation

Even though this research is focussed on the exploration and development of a new clustering algorithm and not focussed on directly providing a competitive alternative to current clustering algorithms, a comparison with the alternatives gives more insight into features of the Soap Particle Clustering algorithm.

In this section we will describe some computer experiments that have been set up such that the features of the algorithm are emphasized.

3.4.1 Performance measures

There are various methods to measure the performance of clustering algorithms. According to Bezdek et al. (2005), the output of a clustering algorithm is ‘a model whose measure of mathematical similarity may capture structure in the sense a human might perceive it.’ However, while there is agreement to some degree that similarity is related to distances, the exact interpretation of similarity is not agreed upon. This can be a reason that there are so many different methods.

Some of the frequently used performance measures for clustering algorithms are:

Statistical indices the fit of clusters to an assumed distribution;

Geometric indices such as cluster density and the separation between clusters;

Davies-Bouldin index the ratio of the sum of the within-cluster scatter to the between-cluster separation;

Dunn’s index the worst-case ratio between cluster diameter and separation;

Fuzziness measures considering the compactness and separation of fuzzy clusters.

More information on these measure can be found in Bezdek, Keller, Krisnapuram, and Pal (2005). We will not discuss these measures further, because these methods are for the most

part not directly applicable to clusters for system identification, as they generally assume spherical clusters and affine subsystems are generally not spherical. However, one specific statistical index we deem applicable: the Receiver Operating Characteristic (ROC) curve (Fadili et al. 2000).

The goal of the clustering algorithm is to find and describe the convex regions in the data. As the clusters are convex by construction, we only need to validate whether the found the clusters describe the data well. The ROC-curve is a measure about how well the data is described. An example of an ROC-curve is given in Figure 3-5.

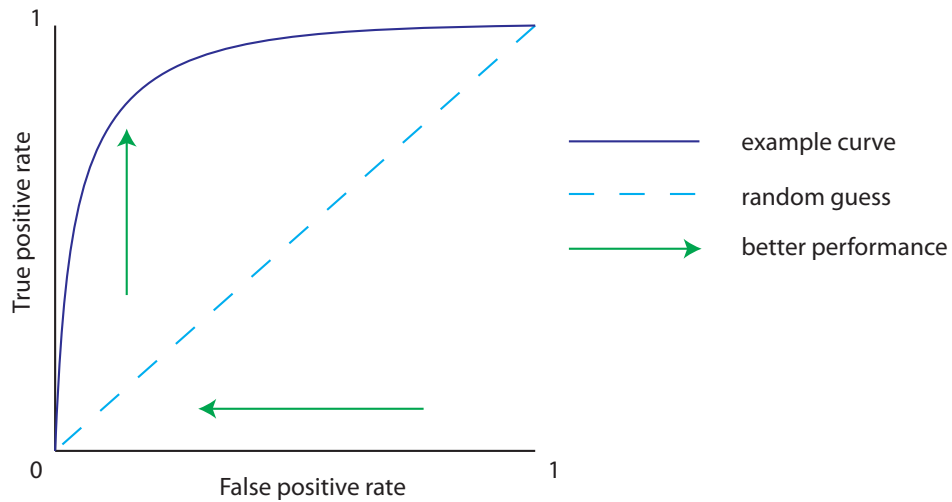


Figure 3-5: Example ROC-curve

The ROC-curve plots the True Positive Rate (TPR) versus the False Positive Rate (FPR) of a binary classifier, that is a method that labels data points as one of two classes. We define all the data that is assigned to a cluster as one of the two classes, and the outliers are the other class.

The validation uses two different data sets: a validation set with data points drawn from distribution as the training set, and a randomly distributed set as a outlier set. For both sets the membership degrees to the clusters that were created on the training set are determined. The share of data points in the validation set that are labelled as member of the clusters, is the TPR and the share of data points in the outlier set that are labelled as member of the clusters is the FPR. A random classifier will result in an equal value of the TPR and the FPR,

which in the figure is displayed as a dotted diagonal line. A good performance corresponds with a high TPR combined with a low FPR.

Conventional clustering algorithms normally do not distinguish outliers, as all data points are clustered. For fuzzy partitionings we can designate data points with a summed not-normalized fuzzy membership degree less than a certain boundary value an outlier:

$$\sum_l \bar{\mu}_{l,p} < \mu_{\text{outlier}} \quad (3-7)$$

By varying the value of the membership degree boundary value μ_{outlier} we can find multiple points on the ROC-curve.

The ROC-curve is a graph, while a performance number is more usable. Conventionally the area under the ROC-curve is used as performance number. However our method of creating an ROC-curve does not produce a full curve, therefore we cannot use this method. We pose an alternative: we use the maximum ratio between the TPR and FPR: the ROC ratio.

3.4.2 Comparison with Gustafson-Kessel clustering

The eventual goal of this thesis report is to produce a method for non-linear system identification. To this cause, we need to find convex clusters. The SP Clustering algorithm delivers these convex clusters. For comparison, another clustering algorithm that delivers convex clusters is needed. A family of clustering algorithms that is commonly used for system identification is *c*-means clustering. Just like the SP clustering algorithm, this family of clustering algorithms results in convex shaped clusters, therefore we can compare them. For system identification purposes it is common to use Gustafson-Kessel (GK) *c*-means clustering, as that algorithm allows more elongated cluster shapes, which conform more to the shape of the affine submodels. Furthermore, GK clustering –just like SP clustering– allows a fuzzy partitioning, which helps with interpolation of the subsystems in the system identification phase. Therefore, we will compare the performance of the SP clustering algorithm with GK clustering.

3.4.3 Computer experiments

We have performed experiments with different data sets. These data sets have different shapes: a banana shaped data set, a Highleyman data set –consisting of two ellipsoidal clusters that intersect, two spheres, two cubes, and two lines, see respectively Figure 3-6, Figure 3-7, Figure 3-8, Figure 3-9, and Figure 3-10. The first two data sets we only tested in $\mathbb{R}^2$, as they were only available in 2 dimensions; the other we have tested as 2, 3, and 5 dimensional data sets. The spheres and squares have both an different compactness to increase the difficulty for the algorithms.

The following parameters, we have chosen for the tests:

Number of sp $n = 100$

Maximum edge length $e_{\max} = 2$

Number of clusters For GK we take c from the SP algorithm

As the GK algorithm needs the number of clusters c beforehand, we use the number of clusters from the output of the SP algorithm.

3.4.4 Results of the experiments

We present the results of these experiments in three tables for each number of dimensions: the results for data in $\mathbb{R}^2$ are presented in Table 3-1, for the 3 dimensional data in Table 3-2, and for the 5 dimensional data in Table 3-3.

In Figure 3-6, Figure 3-7, Figure 3-8, Figure 3-9, and Figure 3-10, the resulting clustering for the 2-dimensional data sets are given. The different types of hatches correspond with the different clusters. The thickness of the hatching lines relates to the not-normalized membership degree. Also the ROC-curves are given.

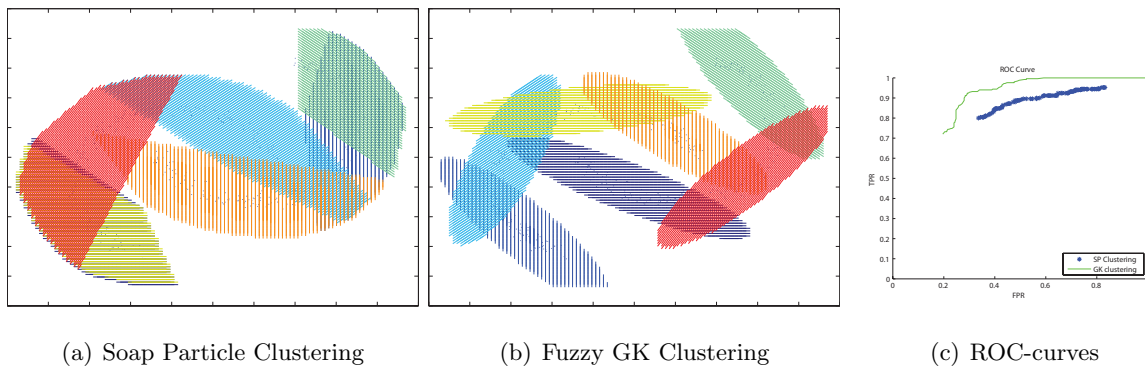
$d = 2$	SP Clustering	GK Clustering
Banana data set	1.4185	1.4864
Highleyman data set	1.4475	1.6927
Two spheres	2.4720	1.8008
Two cubes	1.6709	1.5875
Two lines	1.9638	2.4030

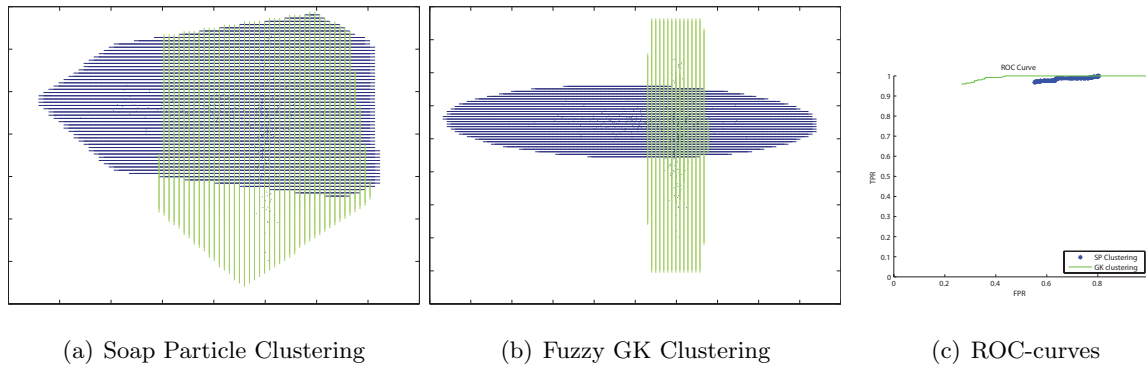
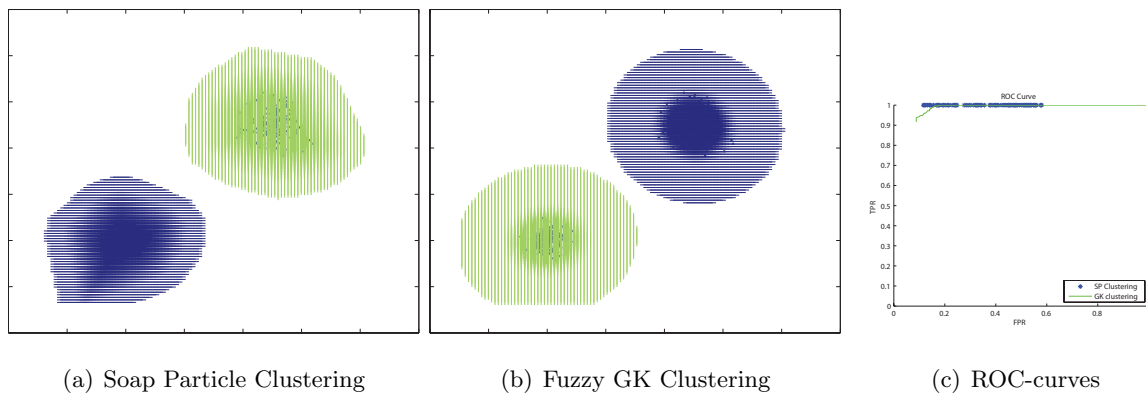
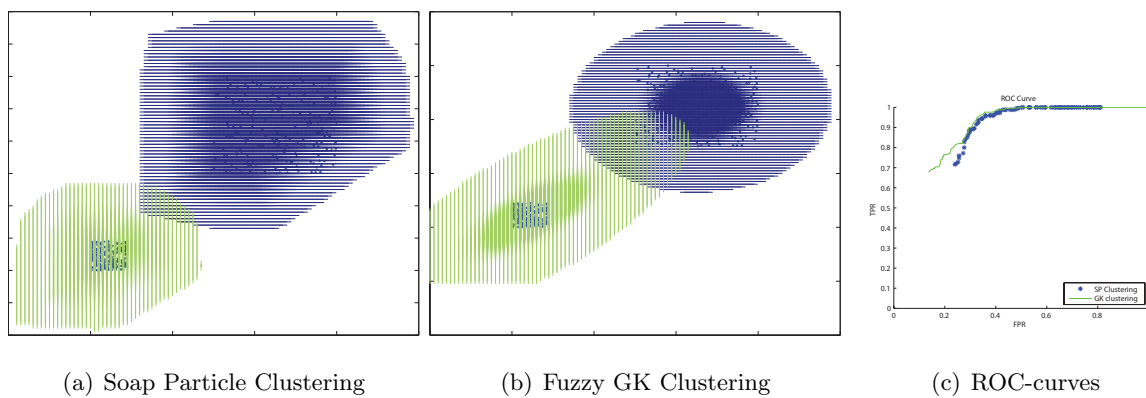
Table 3-1: ROC-ratio for the data sets with $d = 2$ (higher = better)

$d = 3$	SP Clustering	GK Clustering
Two spheres	4.8938	1.6619
Two cubes	2.7700	1.6525
Two lines	1.5371	1.2844

Table 3-2: ROC-ratio for the data sets with $d = 3$ (higher = better)

$d = 5$	SP Clustering	GK Clustering
Two spheres	19.7893	1.3811
Two cubes	2.7384	1.4249
Two lines	2.5586	1.3411

Table 3-3: ROC-ratio for the data sets with $d = 5$ (higher = better)**Figure 3-6:** The banana data set

**Figure 3-7:** The Highleyman data set**Figure 3-8:** The two spheres data set**Figure 3-9:** The two cubes data set

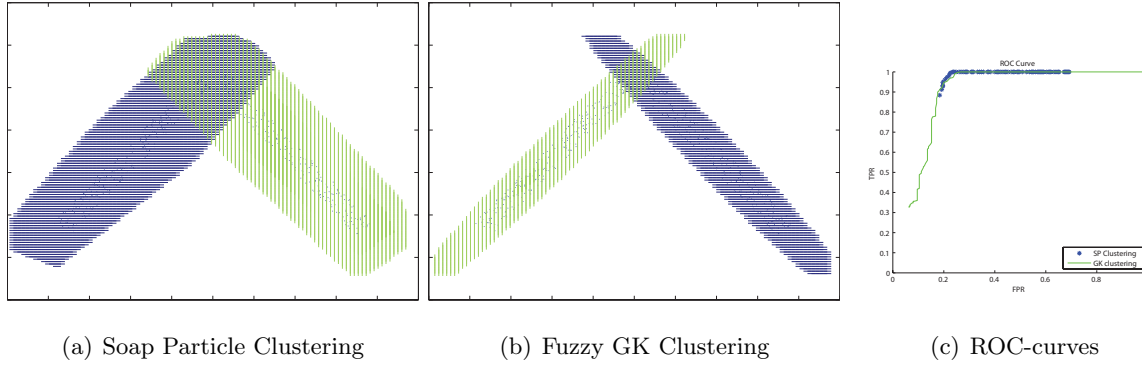


Figure 3-10: The two lines data set

3.4.5 Conclusions of the experiments

Both algorithms result in similar ROC ratios. While often the curve of the SP algorithm is lower than that of the GK algorithm, this does not directly result in a lower ROC ratio. The SPs provide a strict classification between the clusters and the outliers, as they are boundary based. This results in a high ROC ratio. However, due to the incomplete description – resulting from the SP Hull Finding – of the more difficult shapes –such as the banana data set– the ROC curve is lower for these data sets. The strict classification of the SP algorithm also results in a base level of false positives when the validation data set has a broader support than the training data set, as some data points from the validation set are classified as outlier. The GK algorithm has less problems with this, as it makes certain assumptions on the cluster shape, which are correct for these artificially generated data sets.

Furthermore, the GK algorithm was slower in our tests, mainly because the updates on one cluster centre influences the optimality of the other cluster centres, while the updates of the soap particle positions are uncoupled. This effect is best observed with a high number of cluster centres.

Due to the lack of completeness of the Hull Finding stage, the clustering in this stage was occasionally incomplete, which resulted in a high number of outliers in the training set. When this occurred, we reran the experiment with a different random seed. This can be a large drawback, but note that the *c*-means algorithm in practice also needs to be reran multiple

times, because of local minima, and to tune the number of cluster centres. In our experiments, the SP algorithm was less stable than the GK algorithm and needed more reinitializations.

3.5 Conclusions and recommendations

3.5.1 Conclusions

The proof of principle of clustering using the locally convex hull –found by the previous stage as described in Chapter 2– has been shown. The cluster descriptiveness is comparable to the Gustafson-Kessel clustering algorithm, and the empirical computational complexity is lower.

3.5.2 Recommendation

Before the SP clustering algorithm can be used in industry, some improvements need to be addressed: apart from the problem with completeness from the Hull Finding stage (addressed in Appendix A), which was the main drawback shown in the experiments, in this stage the following three improvements can be made:

Performance of maximal cliques It is likely, that the algorithm used to find maximal cliques can be made faster, as the current implementation has not yet been optimized. For example, one could look at ordering the rows and columns beforehand according to their number of connected nodes. This way, it is plausible the largest cliques are found first, which results in fewer clusters. Also it should be possible to integrate the step which removes the outer clusters with this step.

Focussed generalization The generalization currently is applied in all directions and over all SPs equally. While only the situation of Figure 3-1 really requires this generalization. When we can address this situation without relaxing the constraints globally, we can obtain a more tight cluster description. This could be the subject of future research.

Outlier handling Outliers are currently marked as outliers, and are not labelled automatically. There exist some manual methods to fix these unlabelled points, but it would be more convenient if there would be an automatic method, therefore we recommend that more research on this topic is performed.

Chapter 4

Soap Particle Non-linear System Identification

4.1 Introduction

In Chapter 2 we described an algorithm to obtain the locally convex hull of a data set. In Chapter 3 we obtained convex clusters based on that hull. In this chapter we apply those techniques to the field of system identification: the third and final stage of the SP Non-linear System Identification algorithm.

System identification has two distinct stages: first we build a mathematical model, and afterwards we can use that model for simulation or prediction. In Section 4.2 we outline the process of building the model and in Section 4.3 we describe some applications of the model. In Section 4.4 we compare the model to a model obtained by more traditional fuzzy techniques.

4.2 Model building

In this section we first state the objective of the model building stage of non-linear system identification, then the programming interface, and finally the algorithm itself.

4.2.1 Objective

A model in the field of system identification describes a relation between system outputs y and system inputs u . System identification is a branch of regression analysis that has a focus on dynamical models. In dynamical models the current outputs are not only dependent on the current inputs, but also on past inputs and outputs or time derivatives. We will take a generalized approach by building a model that gives a mapping between one dependent variable $\mathbf{y}$ and a set of independent variables $\mathbf{x}$ from measurements on an unknown system. In this approach it does not matter whether the set of independent variables consists of present inputs $u(k)$, past inputs $u(k-1)$, present outputs $y(k)$, past outputs $y(k-1)$, or time derivatives $\frac{du}{dt}$. Furthermore, the dependent variable $\mathbf{y}$ is one model output $y(k+1)$: this results in a Multiple-Input Single-Output (MISO) model. Multiple outputs can be handled with multiple models: one model for each output.

In the validation stage we will focus on discrete time models, as measurements are generally discrete time. The model input vector then consists of lagged input and output measurements, for example for a second-order model with two inputs at time instant k , the input vector is:

$$\mathbf{x}(k) = [y(k) \quad y(k-1) \quad u_1(k) \quad u_1(k-1) \quad u_2(k) \quad u_2(k-1)] \quad (4-1)$$

The model then becomes:

$$y(k+1) = \Phi(\mathbf{x}(k)) \quad (4-2)$$

Unmodelled effects are assumed to be white noise on the output, which is included with the error term ϵ_t :

$$y(k+1) = \Phi(\mathbf{x}(k)) + \epsilon_t \quad (4-3)$$

We assume that the system is time-invariant, that means that the mapping Φ is constant for any time instant k . This way we can transform the modelling problem into a regression problem, with regressor matrix X and regressand vector $\mathbf{y}$:

$$\begin{aligned} X &= \begin{bmatrix} y(2) & y(1) & u_1(2) & u_1(1) & u_2(2) & u_2(1) \\ y(3) & y(2) & u_1(3) & u_1(2) & u_2(3) & u_2(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ y(N-1) & y(N-2) & u_1(N-1) & u_1(N-2) & u_2(N-1) & u_2(N-2) \end{bmatrix} \\ \mathbf{y} &= \begin{bmatrix} y(3) \\ y(4) \\ \dots \\ y(N) \end{bmatrix} \end{aligned} \quad (4-4)$$

In general, the mapping Φ can be non-linear. Many system identification algorithms are focussed on the special case that the mapping is linear, which results in a Linear Time-Invariant (LTI) model. We consider the more general non-linear case. The aforementioned model structure (Eq. 4-3) then is a Non-linear AutoRegressive with eXogenous input (NARX) model structure.

For the construction of non-linear models a main distinction can be made between global and local methods. The global methods describe the relation between inputs and outputs with a single –possibly compound– non-linear function, for example neural networks. We use a local method. The local methods decompose the operating range into multiple local operating regimes. For each regime a local affine model Φ_l is constructed and the global model output is obtained with an interpolation mechanism, such as fuzzy if-then rules. The regimes correspond with the convex clusters $\mathcal{C}_l$ we have found the previous stage: the SP Clustering. The full Takagi-Sugeno-Kang fuzzy model (Sugeno and Kang 1988) is stated as follows:

$$y(k+1) = \Phi(\mathbf{x}(k)) = \begin{cases} \text{if } \mathbf{x}(k) \text{ is } \mathcal{C}_1 \text{ then } \Phi_1(\mathbf{x}(k)) \\ \text{if } \mathbf{x}(k) \text{ is } \mathcal{C}_2 \text{ then } \Phi_2(\mathbf{x}(k)) \\ \dots \\ \text{if } \mathbf{x}(k) \text{ is } \mathcal{C}_l \text{ then } \Phi_l(\mathbf{x}(k)) \end{cases} \quad (4-5)$$

Each local model Φ_l is an affine ARX model:

$$y(k+1) = \mathbf{c} + \Phi_l(\mathbf{x}(k)) + \epsilon_t \quad (4-6)$$

in which $\mathbf{c}$ is a constant. The constant is included, so that the local models do not have to intersect the origin.

The objective of this stage is consequently to find these affine local models, which is illustrated with Figure 4-1.

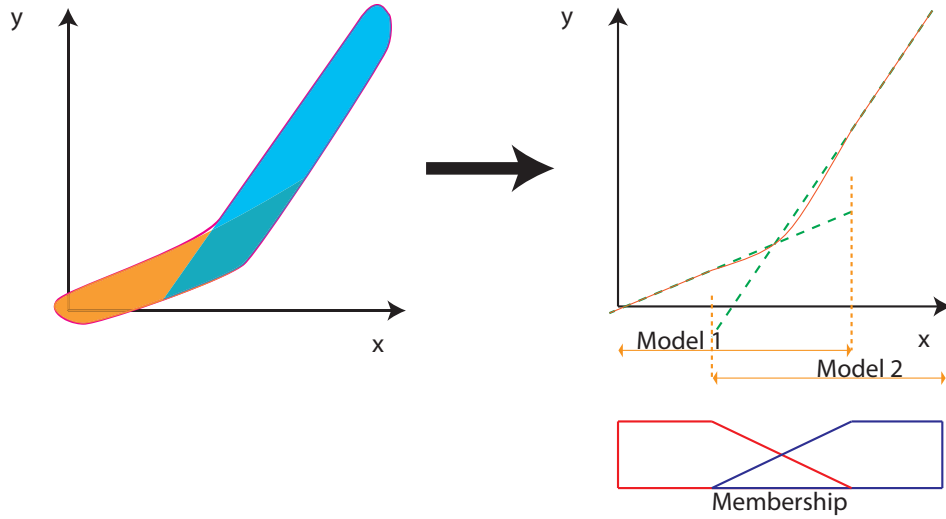


Figure 4-1: The third stage: building a model

4.2.2 Input, output and assumptions

The inputs to the algorithm are the clusters in the form of the partition matrix U with the membership degrees $\mu_{l,p}$ of each data point p to cluster l , and the regression matrix X and regressand vector $\mathbf{y}$. U we can create with Algorithm 3.C as described in Chapter 3 and X and $\mathbf{y}$ we form according to Eq. 4-4.

The assumptions are that the system can be modelled with a discrete time NARX MISO model where each local model can be modelled with an ARX model.

A summary of the inputs, the outputs, and the assumptions is given below:

Program Interface 4.A

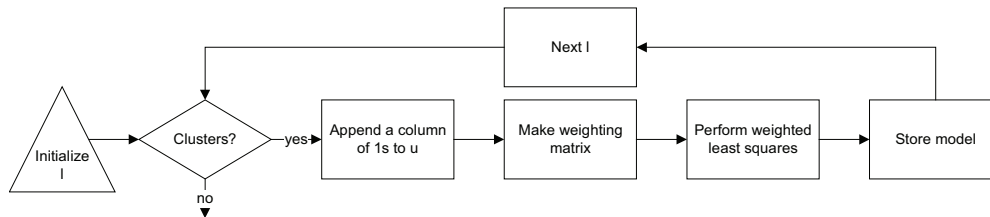
Input	A regression matrix X with membership degrees $\mu_{l,p}$ and regression vector y
Parameters	• No parameters
Output	An affine system Φ_l for each cluster
Assumptions	• Discrete time • Noise model: ARX • MISO of order o

4.2.3 SP Modelling Algorithm

The algorithm is outlined in Algorithm 4.A. Some notable steps we discuss in more detail below.

Algorithm 4.A (*Soap Particle Modelling*)

Find for each cluster an affine local submodel Φ_l



A-1 [Initialize l] Start with the first cluster

$(l \leftarrow 1)$

A-2 [Clusters?] If there are no more clusters stop

(if $l > c$ then stop.)

A-3 [Append a column of ones to X]

$$\left(\begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix} \Rightarrow X_l \right)$$

A-4 [Make weighting matrix]

$$\left(w \leftarrow \begin{bmatrix} \mu_{l,1} & & 0 \\ & \dots & \\ 0 & & \mu_{l,N} \end{bmatrix} \right)$$

A-5 [Perform weighted least squares]

$$(\Phi_l \leftarrow (X^T w X)^{-1} X^T w \mathbf{y})$$

A-6 [Store model]

$$(\Phi_l \Rightarrow \Phi)$$

A-7 [Next l] Add 1 to l and go to **A-2**

$$(l \leftarrow l + 1 \text{ then } \mathbf{A-2}) \blacksquare$$

Append a column of ones

A linear ARX model can be cast into a least squares problem, however our affine ARX models have a constant term that also needs to be estimated. We remove this term by including it into the regression matrix as a constant row.

Perform weighted least squares

Least squares is since long used as a solution to linear regression problems. It can be used for overdetermined systems, that means that the regression matrix should have more rows than columns. Consequently, for an o -th order MISO system with n_{in} inputs, we require a sufficient number of measurements N :

$$N > 1 + o(1 + n_{\text{in}}) \quad (4-7)$$

The least squares solution of

$$X\Phi = \mathbf{y} + \epsilon_t \tag{4-8}$$

is the Φ that minimizes the sum of squared errors: $\epsilon_t^T \epsilon_t$.

Not all measurements should contribute equal to the model, as their membership degree differs. We therefore create a weighting matrix, which levels their contributions according to their membership degree. The weighted least squares solution is the Φ that minimizes the weighted sum of squared errors: $\epsilon_t^T w \epsilon_t$.

4.3 Simulation and prediction

In the following section we state the objective of the simulation and prediction in non-linear system identification, the programming interface, and the algorithm itself.

4.3.1 Objective

Usually, models created of dynamical systems are used for simulating or predicting. With simulation the model only uses given inputs. Any needed past outputs are generated by past outputs of the model itself. With prediction, also the past outputs are given. For 1-step ahead prediction, all needed outputs are given. For multi-step ahead predictions, first some simulation steps are performed to obtain all the needed outputs. For example, two-step ahead prediction, first does a prediction step to calculate one model output, and then a simulation step using both the given past outputs and the calculated output.

Both simulation and prediction are regression problems: estimate output $\mathbf{y}$ from given regressor matrix X . If we can determine the membership degrees of X to all the clusters, we can apply the fuzzy if-then rules (Eq. 4-5) to create $\mathbf{y}_{\text{est}}$. However, X has a lower dimension than $\mathcal{Z}$, the data set the clusters are based on: $\mathbf{y}$ is missing from the data. This creates a Catch-22 situation: we need $\mathbf{y}$ to determine the membership degrees, and we need the membership degrees to estimate $\mathbf{y}$. These are the objectives of this section:

- Determine the membership degrees of X
- Find an estimate to $\mathbf{y}$

4.3.2 Input, output and assumptions

The input to the algorithm is the model with the submodels and labelled soap particles, and the regressor X . The output of the algorithm is an estimate for $\mathbf{y}$ called $\mathbf{y}_{\text{est}}$. We use the same assumptions as the previous section, as we use the model generated in Section 4.2. We use

the SP labelling algorithm of Section 3.3, therefore we also use the generalization parameter g_{label} . A summary is given in Interface 4.B.

Program Interface 4.B

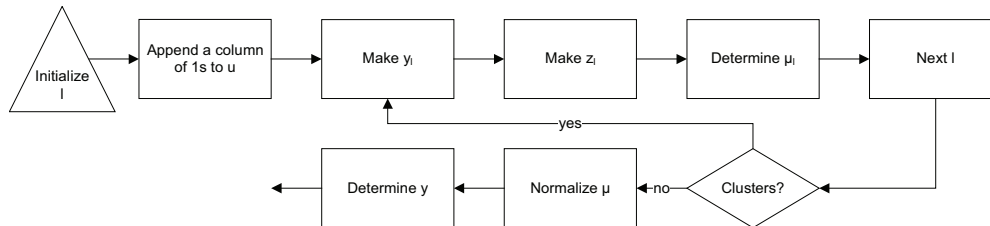
Input	Set of labelled soap particles $\mathcal{S}_l$ and associated submodels Φ_l and current and past inputs $\mathbf{x}$
Parameters	• Generalization parameter g_{label}
Output	An estimate future output $\mathbf{y}_{\text{est}}$
Assumptions	• Discrete time • Noise model: ARX • MISO

4.3.3 SP Prediction Algorithm

To solve the dilemma, we first calculate a $\mathbf{y}_{\text{est}}$ for each local model. This $\mathbf{y}_{\text{est}}$ we use to make a complete data point, which we can label with Algorithm 3.C. We for each $\mathbf{y}_{\text{est}}$ we store the not normalized membership degree to the corresponding cluster. Then we ensure that the set of these membership degrees sums to one. This way we can create the final $\mathbf{y}_{\text{est}}$ according to Algorithm 4.B. The procedure is illustrated by Figure 4-2.

Algorithm 4.B (*Soap Particle Prediction*)

Predict a future output $\mathbf{y}$ from current and past inputs $\mathbf{x}$



B-1 [Initialize l]

$$(l \leftarrow 1)$$

B-2 [Append a column of 1s to $\mathbf{x}$]

$$\left(\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \Rightarrow \mathbf{x} \right)$$

B-3 [Make $\mathbf{y}_l$]

$$(\mathbf{y}_l \leftarrow \Phi_l(\mathbf{x}))$$

B-4 [Make $\mathbf{z}_l$]

$$(\mathbf{z}_l \leftarrow [\mathbf{x} \quad \mathbf{y}_l])$$

B-5 [Determine membership to $\mathcal{C}_l$]

$$\left(\bar{\mu}_l \leftarrow \begin{cases} \min m(s_{l,i}, \mathbf{z}_l) + g_{\text{label}} \forall s_{l,i} \in \mathcal{S}_l & \text{if } \mathbf{z}_l \in \mathcal{C}_l \\ 0 & \text{otherwise} \end{cases} \right)$$

B-6 [Next l] Add 1 to l

$$(l \leftarrow l + 1)$$

B-7 [Clusters?]

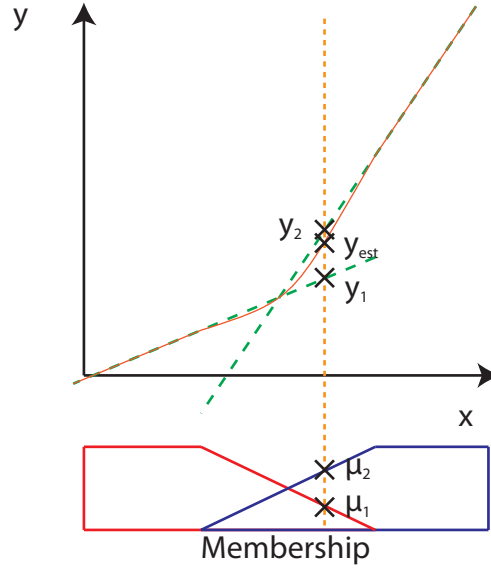
$$(\text{if } l \leq c \text{ then } \mathbf{B-3})$$

B-8 [Normalize memberships]

$$\left(\mu_l \leftarrow \frac{\bar{\mu}_l}{\sum_{m=1}^c \bar{\mu}_m} \mid l = 1, \dots, c \right)$$

B-9 [Determine $\mathbf{y}$]

$$(\mathbf{y} = \sum_{l=1}^c \mu_l \mathbf{y}_l) \quad \blacksquare$$

Figure 4-2: Prediction of y

4.4 Validation

To measure the performance of the algorithm, we perform some computer experiments. The focus of this thesis is to develop a new algorithm, therefore only a restricted number experiments are performed. We apply the algorithm to two artificial regression problems and two dynamical systems, for which we will perform a simulation.

4.4.1 Performance measures

A common measure of regression performance is the coefficient of determination R^2 . It is 1 minus the sum of squared errors divided by the total sum of squares, or alternatively: the squared correlation coefficient between the observed and the modelled output. The R^2 usually ranges between 0 and 1. A R^2 of 1 means that the model perfectly predicts the observed output, a R^2 of 0 means that the model performs equal to an estimate that always gives the mean value of the outputs. When the R^2 is lower than 0, the model actually performs worse.

When we multiply the R^2 with 100, we obtain the percentage of variance accounted for VAF by the model. We will use this measure, as it is more intuitive.

4.4.2 Comparison

The SP System Identification algorithm provides a non-linear system description based on multiple local affine submodels with a fuzzy interpolation. Another technique that delivers the same type of models, is the use of Gustafson-Kessel fuzzy c -means clustering with local linear consequents. Therefore, we compare the SP method with the GK method. To show the added advantage of the non-linear models we also include the results of a linear ARX model.

4.4.3 Computer experiments

For the computer experiments we used 5 different systems: 3 artificial regression problems and 2 dynamical systems. The 3 artificial systems are a linear system, an absolute value system Eq. 4-9 and a cosine Eq. 4-10.

$$y(k) = |u(k)| \quad (4-9)$$

$$y(k) = 1 - \cos u(k) \quad (4-10)$$

The absolute system resembles natural effects like the relation between the lift versus horizontal speed, or an iron cart moving past a magnet. The sharp change of system behaviour is similar to different effects such as saturation –for example due to physical limits– and dead-zones –for example due to friction–. The cosine is based on the relation between angle and height of a pendulum. We have added uniform random noise to the artificial measurements.

The two dynamical systems are a Simulink engine model and a non-linear pressure valve. Both are example systems from Babuška (2001).

The engine has 3 measurements: the throttle setting, the engine speed and the engine pressure. We use multiple relations between these measurements. First order systems between the throttle and speed, and between the throttle and speed together and the pressure; and a second order system between the throttle and pressure.

The pressure valve system has 2 measurements: the valve position and the output pressure.

The measurement sets are split in half: one half is used for the system identification, the other half is used for validation.

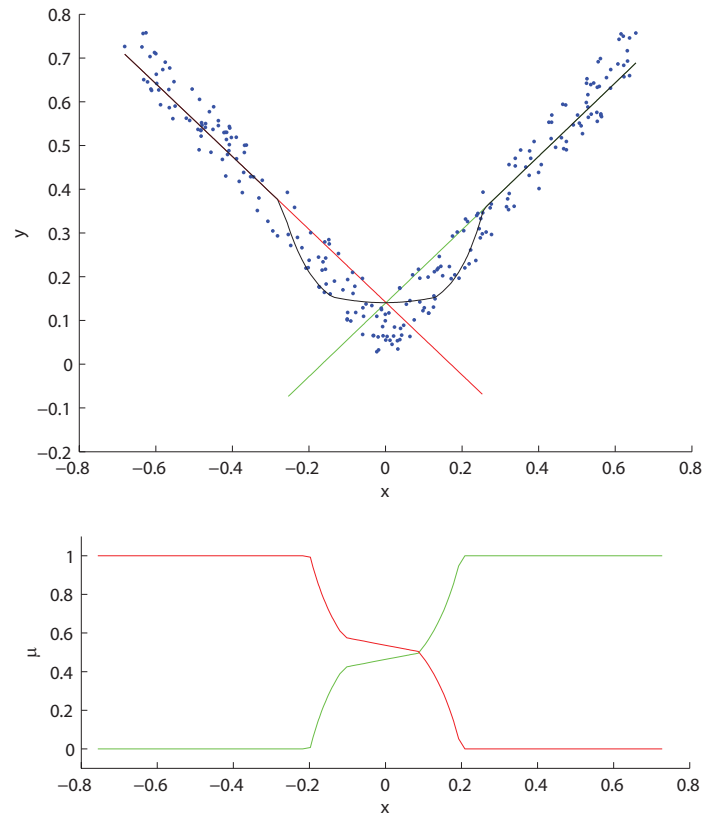
For the SP System Identification, all the measurements are first scaled between 0 and 1. These scaling factors are used again to transform the output back. A constant value for the SP radius can then be chosen, which results in a fully automated process. For the GK algorithm we tune the number of clusters c . In Table 4-1 the other settings for the experiments can be found.

r	0.1	SP radius
N	250	number of data points for the artificial data sets
n	100	number of SPs

Table 4-1: Setting for the experiments

4.4.4 Results of the experiments

First we will show the regression plane for the first artificial system in Figure 4-3. In Table 4-2 the VAF values are presented for the regression, in Table 4-3 the VAF values are presented for the simulation, and in Figure 4-4 the modelled outputs and the measured output are presented of the two input system as example.

**Figure 4-3:** SP Regression of the absolute value function

	SP Identification	c	Gustafson-Kessel	c	Linear
Linear	99.85	1	99.85	1	99.78
Absolute value	99.38	3	96.54	2	-0.21
Cosine	95.88	4	95.03	4	-0.87

Table 4-2: VAF results for regression

	SP Identification	c	Gustafson-Kessel	c	Linear ARX
Engine Speed	99.43	7	99.82	3	98.60
2 Inputs Engine Pressure	94.42	22	91.52	3	-3E43
2nd order Engine Pressure	83.11	14	89.84	4	-3E24
Valve	83.98	10	95.61	3	-41.76

Table 4-3: VAF results for simulation

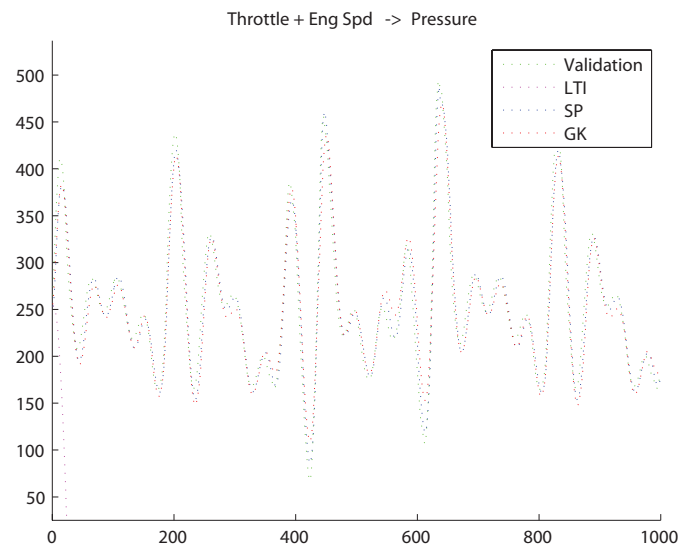


Figure 4-4: Simulation of the engine pressure based on throttle and engine speed

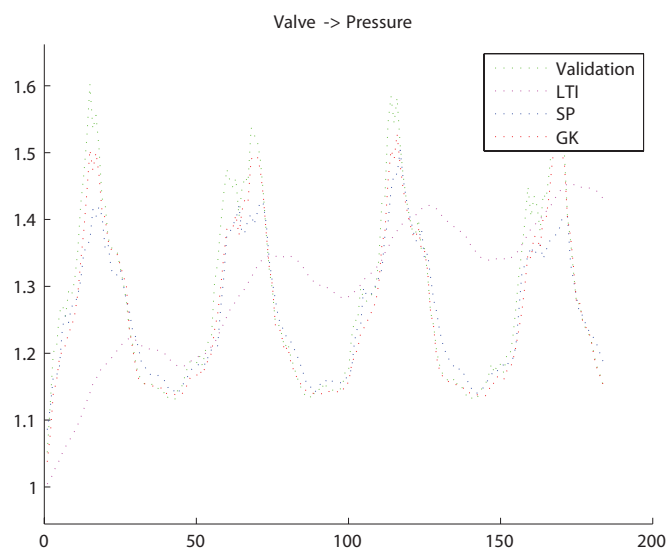


Figure 4-5: Simulation of the pressure based on the valve position

4.4.5 Conclusions of the experiments

In Figure 4-3 we observe that the results are visually comparable with the objective we have set in Section 4.2 and Section 4.3. We can identify a clear bulge or dip in the estimation; this bulge is the region of interpolation between the two affine local submodels. An interpolation only can place a point between the two submodels and the submodels continue to linearly decent in the centre region, therefore the bulge occurs. The clear and fast interpolation has the advantage that it is clear in which spaces each submodel is in effect, this in contrary to the GK algorithm in which the membership degree to any cluster will never be fully 0.

From Table 4-2 and Table 4-3 we can deduce that the SP algorithm performs comparable to the GK algorithm, with generally a slightly less VAF. The linear model generally performs worse, except for the linear models. It is nice to note that the SP algorithm finds a single cluster for the linear system: it generates a linear system.

With these results we have shown the proof of principle for the SP algorithm: the SP system identification performs comparable with the GK system and better than the linear ARX.

4.5 Conclusions and recommendations

4.5.1 Conclusions

In this chapter we have completed the steps needed to build a non-linear model. From the clusters of Chapter 3 we have created a model, and this model we can use to perform simulations and predictions.

With the results in this chapter we have shown the proof of principle for the SP algorithm. However to consistently obtain the same level of performance as the GK method, the SP method still needs to mature. It remains an advantage that there exist good defaults for the different tuning parameters in the SP algorithm, the SP radius in particular can be equal for very different systems, when the measurements are scaled; while in the GK method a crucial parameter –the number of clusters c – is generally different for different systems, and therefore needs to be manually tuned.

4.5.2 Recommendations

The implementation of the recommendations for the other stages will likely improve the performance of the algorithm in this stage. Also the simulation performance should be tested in future research. Other methods than weighted least squares can also be evaluated to obtain the local models, as Johansen and Babuška (2003) have described for the fuzzy c -means methods.

Conclusions and Recommendations

5.1 Conclusions

In this thesis, we have outlined the SP System Identification algorithm in three stages: Soap Particle Hull Finding, Soap Particle Clustering, and finally Soap Particle System Identification.

These chapters have resulted in a proof-of-principle of the methodology, by comparing the resulting Takagi-Sugeno-Kang fuzzy model with a linear ARX model and a fuzzy model obtained by Gustafson-Kessel fuzzy *c*-means clustering. On the tested data sets, the SP model outperformed the linear ARX completely. The performance was comparable with GK-fuzzy modelling: on many runs, the Variance Accounted For was even better for the SP-algorithm.

The number of clusters/subsystems does not need to be supplied beforehand with the SP algorithm. Only the delays of the inputs and outputs, the order of the system need to be specified, in accordance with most other algorithms. All other parameters have good defaults.

A particular interesting thing to note, is that the SP algorithm automatically generates a linear ARX system when the measurement series indicate a linear system.

The theoretical cluster shape in the SP clustering algorithm is any convex shape, which is more in accordance with the general shape of linear subsystems, than the spherical shape in the *c*-means algorithms.

Also, the SP algorithm is a local algorithm in every step which involves the data set, allowing larger data sets with lower computational complexity. This yields faster computations in the experiments performed, compared with the GK algorithm.

The SP has a notion of outliers and does not cluster outliers. This has three main advantages: outliers do not result in bad partitioning of the data; outliers are not included in the regression; and a high number of outliers indicate a problem in the hull finding stage, which can be subsequently solved by rerunning the SP Hull Finding algorithm with a different initialization, or by increasing the number of soap particles.

Finally, even though it was not the primary goal of the thesis project, the project has resulted in a usable Matlab tool for non-linear black-box system identification.

5.2 Recommendations

The following areas are recommended focus areas for the further development of the SP algorithms:

Hull finding One of the preconditions on the performance of the Hull Finding algorithm is that the user specifies a sufficient number of soap particles. The current default value of 100 is sufficient in most situations and results in a good performance in the subsequent stages. However an adaptive number of soap particles, which ensures a complete description, should result in a lower computational complexity and fewer required reinitializations.

Clustering In the clustering stage, we have specified a generalization parameter with a default value equal to the radius of the neighbourhood of the soap particles. Although, there is some theoretical foundation for this generalization parameter, and computer experiments show the beneficial effect of the parameter; more insight into the effects of this parameter can be beneficial.

System identification In the system identification stage currently the different local subsystems are estimated separately with a linear least squares method. There exist globally optimizing methods to determine these consequents, see for example Johansen and Babuška (2003), in which an optimum is sought between the local and global fit. Therefore, it would be advantageous to evaluate these methods when they are implemented on a partitioning made by the SP algorithm.

This project has opened an exciting new approach to clustering in general and clustering for system identification in particular and we would encourage any work that follows a similar three stage approach, but for example using an alternative to the artificial soap particles for the hull finding, such as an adapted QHull.

Bibliography

- Babuška, R. (2001). Fuzzy identification toolbox for matlab.
- Babuška, R. and M. Oosterom (2001). *Granular computing: an emerging paradigm*, Chapter Fuzzy clustering for multiple-model approaches in system identification and control, pp. 306–323. Heidelberg, Germany, Germany: Physica-Verlag GmbH.
- Barber, C. B., D. P. Dobkin, and H. Huhdanpaa (1996). The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.* 22(4), 469–483.
- Bezdek, J. C., J. Keller, R. Krisnapuram, and N. R. Pal (2005). *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. The handbooks of fuzzy sets series. New York, USA: Springer.
- Bomze, I. M., M. Budinich, P. M. Pardalos, and M. Pelillo (1999). The maximum clique problem. In D.-Z. Du and P. M. Pardalos (Eds.), *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers.
- Eckhoff, J. (1993). *Helly, Radon, and Carathéodory Type Theorems*, Chapter 2.1, pp. 389–448. Amsterdam, Netherlands: North-Holland.
- Fadili, M., S. Ruan, D. Bloyet, and B. Mazoyer (2000). A multistep unsupervised fuzzy clustering analysis of fmri time series. *Human Brain Mapping* 10(4), 160–178.
- Gill, P. E., W. Murray, and M. H. Wright (1981). *Practical Optimization*. Academic Press.

- Johansen, T. A. and R. Babuška (2003, December). Multiobjective identification of takagi–sugeno fuzzy models. *IEEE Transactions on Fuzzy Systems* 11(6), 847–860.
- Pardalos, P. M. and J. Xue (1994, April). The maximum clique problem. *Journal of Global Optimization* 4(3), 301–328.
- Pratt, W. K. (2001). *Digital Image Processing* (3 ed.). John Wiley & Sons, Inc.
- Preparata, F. R. and M. I. Shamos (1985). *Computational Geometry – An Introduction*. Springer-Verlag.
- Sjöberg, J., Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky (1995, December). Nonlinear black-box modeling in system identification: a unified overview. *Automatica* 31(12), 1691–1724.
- Sugeno, M. and G. T. Kang (1988). Structure identification of fuzzy model. *Fuzzy Sets and Systems* 28(1), 15 – 33.
- Thus, S., J. van Ast, and R. Babuška (2008). Swarm-based clustering – literature survey.
- Webb, A. (2002). *Statistical Pattern Recognition* (2 ed.). Chichester, UK: John Wiley & Sons, Ltd.

Appendix A

Recommended Approaches to Improvements

A.1 Convergence

The convergence of SPs to faces that are not part of $\mathcal{H}$ can probably best handled by verifying that there are no data points that are ignored while they could be important for the shape of the hull. The situation of Figure 2-5 for example, can be identified by calculating a new neighbourhood around both vertices, and checking the margins of the previously ignored data points. Then it is clear that two points have a negative margin, and the requirement that the hull should enclose the data is not satisfied.

Another solution would be to use a larger radius for the neighbourhood. The movement rules will ensure an SP will only converge when it is located between the vertices: in 2 dimensions, one data point on the left of the SP and one on the right. This means that in the worst-case

scenario, an SP will identify a face with one data point at the centre of the SP and one data point at the maximum radius: therefore, the smallest distance between two vertices of a face, which might not be found by an SP is equal to the radius of the neighbourhood of the SP. Using this approach, a conclusion could be that the radius r should be equal to the maximum edge length $e_{\max}$. However, the valid faces like in Figure A-1 will be incorrectly marked as invalid, and can therefore never be found. Also incorrect faces having edges larger than $e_{\max}$ will be found. Consequently, choosing $r = e_{\max}$ is not a practical approach.

A better approach might be to require that the origin of the neighbourhood is on a data point, and then use a radius of $e_{\max}$. SPs are then anchored to one data point. This solution has not been followed, as it deviates from natural soap particles and has some tricky implications for high dimensional data, but an algorithm with this approach could be very favourable.

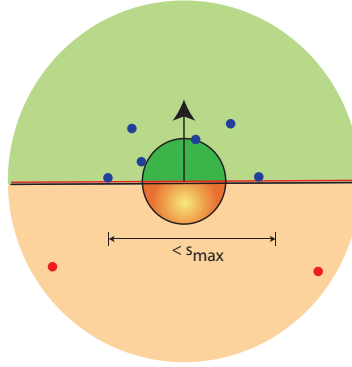


Figure A-1: SP incorrectly revokes valid face when $r = e_{\max}$

A.2 Initialization, Redundancy, and Completeness

When a soap particle is in the neighbourhood of a data point, it will usually converge within a few iterations. In data sets with a large percentage of empty space, which is common in for example high dimensional data, many reinitializations can be needed before a data point will be present in the neighbourhood of the soap particle. Many reinitializations are undesirable as they increase the time complexity of the algorithm.

There are two ways to remove the need for these reinitializations:

- Initialize in the neighbourhood of a data point
- Initialize in the neighbourhood of a converged soap particle

A comparison of these two routines with the original random initialization is made in Table A-1.

Type	Random	Near data pnt	Near sp
Infeasible	Many	Never	Some
Consecutive iterations	Many	Many	Few
Clusters	Multiple	Multiple	Single
Label	Unknown	Unknown	Known
New face	Unknown	Unknown	Primarily
Calculations	Parallel	Parallel	Serial

Table A-1: Comparison between the different initialization routines

As the optimal hull is continuous, a soap particle, which is initialized near an already converged soap particle, will be often near convergence. Furthermore, we can prove this soap particle will converge to the same cluster: because other clusters are at least $e_{\max}$ away, and the diameter of the neighbourhood of the soap particle is $e_{\max}$, the soap particle will not see the other clusters, and therefore will not move to them. This can be an advantage, as we can immediately label the new soap particle appropriately: it gets the same label as the parent soap particle.

We would like to have each face of the optimal hull described by a single soap particle. Most importantly, we would not like to have a face which is not described at all, because then we would lose important information on the cluster shape. If we have a method to determine whether all neighbouring faces of a particular face are already described, and we can determine the direction of that face, we can initialize a soap particle near that face. This soap particle then should quickly converge to the face and the face is described. and in the end ensure that all faces are approximated, using the minimal number of soap particles. While this method is still under development, the following methods are under investigation:

- Verify whether all data points within the neighbourhood of the parent particle are fully clustered.
- Each data point that is a supporting vertex for the soap particle ($m_i = 0$) should be a supporting vertex for at least d soap particles

Initializing near a converged soap particle does enforce a serial evaluation of the algorithm, as the parent soap particle should already be converged.

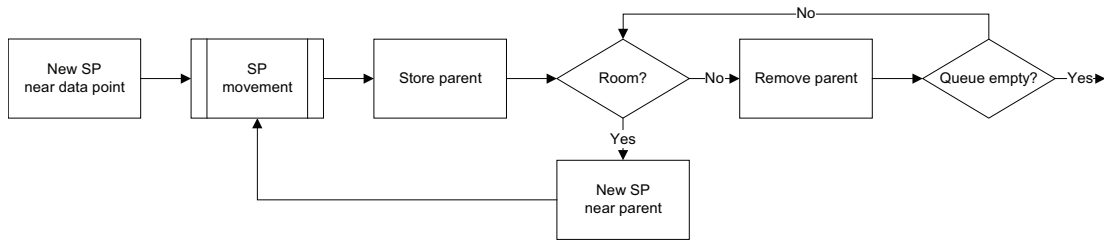
Because we cannot ensure that all regions in neighbourhood of the parent soap particle are feasible, for example, the rear side becomes quickly infeasible, we can better initialize both near a soap particle, and in the neighbourhood of a data point, to ensure feasibility.

We cannot use the initialization near a soap particle for the first soap particle. We can also not use this routine to find a new cluster. For these two situations we will only use the initialization near a data point. We further improve this initialization near a data point, by ensuring it will find thus far unclustered data, by temporarily ignoring all clustered data points during the initialization on a data point.

The change in the way of initialization requires changes in the whole algorithm, as can be seen in Algorithm A.A.

Algorithm A.A (*Neighbourhood filling strategy*)

Given a data set find a connected part of the objective hull with guarantees on the completeness



A-1 [New SP near data point] Initialize an SP near a data point

A-2 [SP movement] Move the SP until it has converged

A-3 [Store parent] Store the SP in the parent queue

A-4 [Room?] If there is room for a new SP near the top parent SP of the queue, then go to step **A-5**, else go to step **A-6**

- A-5** [New SP near parent SP] Initialize an SP near the parent SP (keep the parent SP in the queue) and go to step **A-2**
- A-6** [Remove parent] Remove the parent SP from the queue
- A-7** [Queue empty?] If the queue is empty, then stop, else go to step **A-4** ■

A.3 Noise

Outliers are not a problem for the current algorithm, but noise on the data can lead to a jagged surface of the data. When the noise becomes so large there is no longer a distinction between region with data and regions without: the current objective is no longer applicable.

A solution can be to redefine the local rules. An adaptation would be to not use the smallest margin in the translation and rotation rules, but instead use the second, or n-th smallest margin. This would allow for some extra noise. Another solution would be to use the maximum gradient of the density of the data, which would be even less affected by noise.

A.4 Low dimension

With data in higher dimensions, Euclidean distances between points generally become larger, which means the hypersurface also becomes larger. This means more soap particles are required to describe the data, when the Euclidean distance measure is used. However, any norm can be used a distance measure: for example the infinity norm will generally be less influenced by the number of dimensions.