Universidad de Granada

Bachelor's Thesis

Fuzzy systems for Big Data computing & Ordinary differential equations not solved for the derivative

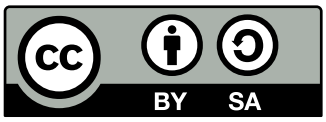
Antonio Coín Castro

Course of study: Doble Grado en Ingeniería Informática y Matemáticas.

Supervisors: Margarita Arias López,
José Manuel Benítez Sánchez,
Miguel Lastra Leidinger.

Faculties: E.T.S de Ingenierías Informática y de Telecomunicación,
Facultad de Ciencias.

September 7, 2020



The source code for this document is available at https://www.github.com/antcc/tfg.

Summary

In the first part of this document we present a study of the interaction between fuzzy systems and Big Data, both from a theoretical and an applied point of view. We begin by studying the fundamentals of fuzzy set theory and fuzzy logic, stopping to analyze in depth the concepts of fuzzy *if-then* rules and fuzzy reasoning. Next, we delve into the question of what Big Data is and we explore its fundamental characteristics, reviewing the state of the art in the infrastructures and algorithms in this field and examining the industry standard frameworks: the MapReduce paradigm and the Apache Spark platform. Once this theoretical study is complete, we undertake an implementation task to design and program a suite of scalable fuzzy learning algorithms, explaining how they could be useful in solving problems that imply large amounts of data. Finally, we perform a comparative study of the algorithms developed, executing them in a distributed environment and analyzing the results obtained.

In the second part of this work we carry out a study of first order implicit differential equations, that is, ordinary differential equations that are not solved for the derivative and cannot be directly put in explicit form. We begin with a review of the basics concepts in the theory of explicit differential equations, recalling their geometrical interpretation and the main theorems for establishing the existence and uniqueness of solutions. Next we introduce implicit differential equations both from an algebraic and a geometric point of view, framed within the general theory of differentiable mappings. We explore general conditions that allow us to treat these equations locally as explicit equations, while also analyzing the singular solutions that may arise in their resolution. Lastly, we present several concrete examples of classical equations that have been studied in the mathematical community, relating them to the theory developed in this work and providing general methods for solving them.

KEYWORDS: fuzzy systems, computational scalability, Big Data, implicit equations, differentiable mappings, singularities.

Resumen extendido

Este trabajo se compone de dos partes claramente diferenciadas, cada una enfocada principalmente a una disciplina distinta y con un tema de estudio diferente. A continuación resumimos con cierto detalle el contenido de cada una de estas partes.

En la Parte I se recoge el contenido más aplicado y relacionado con la ingeniería informática, si bien no por ello carece de contenido matemático. El objetivo principal de esta parte del trabajo es estudiar la interacción entre dos campos importantes en la informática y sus aplicaciones, como son los sistemas difusos y el Big Data.

Por un lado, los sistemas difusos son modelos de aprendizaje basados en las herramientas de la lógica difusa y la teoría de conjuntos difusos, que introduce un enfoque novedoso en la manera de modelar muchos problemas. Específicamente, en este contexto las entidades fundamentales son los *conjuntos difusos*, una generalización de los conjuntos usuales mediante la cual se permite que los elementos tengan un cierto grado de pertenencia al conjunto, en lugar de simplemente pertenecer o no pertenecer a él. Utilizando estos conjuntos como base se construye una lógica multivaluada que permite tratar y razonar con situaciones que presentan una ambigüedad inherente y son difíciles de analizar mediante la lógica bivalente a la que estamos acostumbrados, como puede ser el procesamiento del lenguaje humano.

Por otro lado, de un tiempo a esta parte hemos podido observar cómo la cantidad de datos que manejamos ha ido aumentando a un ritmo frenético. La mayoría de ordenadores personales que se comercializan en la actualidad ofrecen una capacidad de almacenamiento en el rango de los *terabytes*, y para aplicaciones específicas de tratamiento y análisis de datos estas cifras crecen considerablemente hasta los *petabytes* (10^3 *terabytes*) o los *exabytes* (10^6 *terabytes*). Es por eso que se ha vuelto necesario disponer de una infraestructura y unos algoritmos que permitan soportar y tratar adecuadamente grandes cantidades de datos.

Es aquí donde entran en juego los sistemas difusos o sistemas basados en reglas difusas del tipo *si-entonces*, que pueden ser entendidos como sistemas basados en el conocimiento preparados para realizar inferencia, extraer conclusiones sobre unos datos y resolver multitud de problemas que presenten ambigüedad o imprecisión en su planteamiento o resolución, entre los que destacan los problemas de control automático. El hecho de disponer de una gran cantidad de datos para confeccionar modelos de aprendizaje de datos en estos sistemas hace que la calidad de la respuesta sea superior, permitiendo en muchos casos resolver problemas de manera aproximada pero suficientemente buena en un tiempo reducido. Además, la interpretabilidad que proporcionan las reglas difusas hace que el resultado final tenga más valor, pues se puede explorar el procedimiento mediante el cual se llega a unas conclusiones concretas.

En este trabajo realizamos primeramente un estudio desde el punto de vista teórico de los conjuntos difusos, la lógica difusa y los sistemas difusos. También estudiamos las características fundamentales en torno al concepto de Big Data y la infraestructura, filosofía de los algoritmos y *frameworks* que han ido surgiendo en respuesta al incremento en la importancia de este campo. En particular, discutimos el paradigma de programación MapReduce y su implementación en la plataforma Apache Spark. Una vez hecho esto,

planteamos un problema de diseño e implementación de algoritmos difusos escalables, que pongan de manifiesto la afinidad comentada anteriormente entre los sistemas difusos y el campo de Big Data. Cabe destacar que no perseguimos realizar una mera paralelización de algoritmos existentes, sino que pretendemos estudiar y proponer diseños alternativos que permitan una mayor escalabilidad. En concreto, realizamos una adaptación escalable del algoritmo de Fuzzy C-Means, del algoritmo de *subtractive clustering* de Chiu y del algoritmo de construcción de reglas difusas de Wang y Mendel.

Finalmente, realizamos un estudio comparativo de los algoritmos desarrollados, ejecutándolos en un entorno distribuido con suficiente capacidad como para manejar de forma efectiva conjuntos formados por varios millones de puntos. Estudiamos experimentalmente estos algoritmos junto con algunos otros modelos escalables no difusos ya implementados en Spark, y analizamos los resultados obtenidos. De este análisis extraemos la conclusión de que efectivamente se ha conseguido diseñar algoritmos escalables en el contexto de los sistemas difusos, realizando una aportación original a la investigación actual en este ámbito.

* * *

En la Parte II encontramos el contenido puramente matemático del trabajo. En ella se acomete el estudio de ecuaciones diferenciales de primer orden en forma implícita, es decir, aquellas para las que la derivada no se puede despejar. En un curso básico de ecuaciones diferenciales ordinarias usualmente se trata siempre el caso explícito, esto es, ecuaciones de la forma $y' = f(x, y)$, y no se suelen analizar con detalle las ecuaciones implícitas de la forma $F(x, y, y') = 0$. En este trabajo proponemos profundizar en el estudio de estas últimas ecuaciones desde el punto de vista de la teoría general de aplicaciones diferenciables, analizando los comportamientos que presentan este tipo de ecuaciones y aportando condiciones generales para su tratamiento. En concreto, nos interesan especialmente cuestiones relacionadas con la existencia y unicidad de soluciones, pues en este caso no podemos aplicar los teoremas usuales en este aspecto.

En primer lugar comenzamos con una exposición de la teoría básica de ecuaciones diferenciales ordinarias, en un esfuerzo por hacer que el trabajo sea más autocontenido y fijar la notación que se usará a lo largo del mismo. Se definen con rigor los conceptos de ecuaciones diferenciales y soluciones de las mismas. También nos detenemos a analizar con detalle la interpretación geométrica que tienen las ecuaciones diferenciales en forma explícita, y cómo producen en el plano un campo de direcciones que marca la trayectoria de las soluciones o curvas integrales. Tras ello, planteamos la cuestión de existencia y unicidad de soluciones, proporcionando ejemplos en los que no se verifica ninguna de estas, y enunciando los resultados clásicos que establecen condiciones para su tratamiento, como son el teorema de Peano o el teorema de Picard-Lindelöf. En ambos casos se refiere al lector a fuentes donde puede consultar la demostración, tanto en su forma original como en términos más modernos.

Tras esta primera toma de contacto, pasamos a definir las ecuaciones diferenciales en forma implícita, entendiéndolas como ecuaciones de la forma $F(x, y, p) = 0$, donde F es una función de $\mathbb{R}^3$ en $\mathbb{R}$ y $p = dy/dx$. Comenzamos analizando cuáles son las condiciones que nos permiten transformarlas localmente en ecuaciones explícitas, para poder aplicar toda la teoría desarrollada anteriormente. Estas condiciones resultan ser precisamente aquellas que permiten aplicar el *teorema de la función implícita* para expresar p como función de x e y (en concreto, que la derivada parcial de F con respecto a su tercera variable no se anule). Tras ello, estudiamos un primer enfoque algebraico en el que se discute cuándo podemos asegurar que por un punto del plano pasan un número finito de soluciones. En este caso no podemos aspirar a tener una única solución en cada punto, pues en general la ecuación $F = 0$ representa la superposición de varios campos de direcciones, cada uno de ellos correspondiente a una familia de soluciones distinta.

Siguiendo con el estudio de las ecuaciones implícitas, cambiamos el enfoque y nos planteamos el problema desde un punto de vista geométrico, entendiendo la ecuación $F = 0$ como una superficie $M = F^{-1}(0)$ vista en un espacio de tres dimensiones. En este apartado hacemos uso de varios conceptos y resultados de geometría diferencial, y en particular, de geometría diferencial de superficies. Estudiamos cómo surge en la superficie M un campo de direcciones, que al proyectarse en el plano produce localmente un campo de direcciones para cada una de las ecuaciones representadas en la expresión $F = 0$. Además, este planteamiento geométrico nos permite aproximarnos de forma efectiva al estudio de puntos singulares y soluciones singulares que surgen en este tipo de ecuaciones, definiendo los conceptos de *curva criminante* y *curva discriminante*. En este sentido presentamos un resultado interesante que asegura que en torno a puntos singulares que cumplan una cierta condición de regularidad, podemos expresar mediante un adecuado cambio de variable la ecuación $F = 0$ en la denominada *forma normal de una ecuación implícita*, mucho más sencilla de resolver.

Finalmente concluimos la exposición examinando una serie de ecuaciones clásicas en el ámbito de las ecuaciones en forma implícita, como son las ecuaciones de Clairaut y de Lagrange. En cada caso estudiamos cómo se aplica la teoría desarrollada en el trabajo para el análisis de dicha ecuación, proporcionando también métodos efectivos de resolución y visualizando ejemplos concretos de algunas curvas integrales y sus peculiaridades.

PALABRAS CLAVE: sistemas difusos, computación escalable, Big Data, ecuaciones implícitas, aplicaciones diferenciables, singularidades.

Contents

Part I

Fuzzy systems for Big Data computing

Computer Science

1 Introduction

Nowadays the words *neural network* and *deep learning* sound familiar not only to experts in the field, but also to the general public, as they have become the visible face of artificial intelligence and machine learning. But they are not novel approaches, since the first perceptron model proposed by Frank Rosenblatt dates all the way back to 1958 [Ros58], and then some deep learning algorithms were proposed in the late 60s and early 70s (see [IIL+67] and [Iva71]). However, they did not get much traction because the hardware available at the time could not cope with the requirements that these models presented, so they were relegated only to the theoretical realm. It was not until the decade of the 90s and the beginning of this century that a renewed interest arose in complex algorithms that could *learn* from the data without explicit human intervention.

In this regard, a good alternative were models based on fuzzy theory, which was being concurrently developed as a tool that provided an effective and mathematically tractable way of dealing with situations that presented an inherent ambiguity, such as the processing of human language and thoughts. These fuzzy systems were initially implemented as a way of modelling control systems, in which expert human knowledge was usually involved. The idea of making them learn from the data in a semi-automatic way was borrowed from the artificial neural models described above.

Coupled with the increase in computing power came the increase in the ability to generate more and more data, so much that a term was coined to represent the scenarios in which a large volume of data was involved: it was called Big Data. It turns out that the use of fuzzy systems can improve the performance in the results obtained by Big Data learning algorithms. There are at least two reasons that support using fuzzy techniques in a Big Data context:

1. They serve to model uncertainty not only in the data, but in the whole gathering and analyzing pipeline. Furthermore, they permit a higher level of interpretation than other learning models, and many problems that are unavoidably turning into Big Data problems can surely benefit from some degree of interpretability in their answers.

2. There are situations in which obtaining the exact answer to a problem can be very expensive or time consuming, and it could be the case that a sufficiently good solution that reduces both economic and temporal costs is preferred. This circumstance is aggravated when we have massive amounts of data, or when the data volume never stops increasing.

For the reasons stated above, we set out to first gain a deep understanding of fuzzy systems and Big Data from a theoretical perspective, and then to design some algorithms that bring to light the peculiarities of the interaction between these two concepts, while hopefully making a useful contribution to the current research in the matter.

The current lines of research in this area focus on re-designing existing fuzzy learning algorithms with scalability in mind, which is precisely what we will do in this work. A good review of the state of the art in fuzzy systems for Big Data can be consulted in [WXP17] and [FCDH16].

1.1 Objectives

We summarize below the main objectives of this work:

1. To review the theory of fuzzy sets, fuzzy logic and fuzzy systems, and of fuzzy *if-then* rules in particular. Also, to analyze the distinction between a Mamdani and a TSK fuzzy system, and to explore the philosophy behind the learning algorithms related to them.
2. To understand what Big Data is (and what it is not), to explore the principal characteristics that define it and to familiarize ourselves with the various methods and tools that come with this new paradigm.
3. To design, implement and test a suite of scalable fuzzy system learning algorithms to use in a Big Data environment, employing the industry standard tools for these tasks.

1.2 Structure

In Chapter 2 we tackle the first of the tasks proposed above. We begin with an introduction to fuzzy set theory, reviewing the basic definitions and analyzing with detail the concept of membership functions. Next we introduce fuzzy logic as a muti-valued logic that finds applications in many fields, such as natural language processing. In this part we stop to study in depth the concepts of fuzzy rules and fuzzy inference, which is arguably the core of the fuzzy theory presented here. Finally, we define what a fuzzy system is and analyze some of the different types that can arise.

In Chapter 3 we continue our theoretical study and explore the main characteristics of Big Data and the idiosyncrasies of the algorithms that are used in this context. We also embark in a brief discussion about the ethics and moral implications involved in the gathering of massive amounts of data. Finally, we describe the principal architectures built specifically for Big Data, including Apache Spark and the MapReduce framework.

In Chapter 4 we review the state of the art in fuzzy learning algorithms and analyze the three main types: algorithms based on space partitions, neuro-fuzzy algorithms and genetic algorithms. Next we present a description of the algorithms designed and implemented in Spark as part of the practical portion of this project. In each case we give a high-level account of the general behaviour of the algorithm, we analyze their time complexity and scalability issues and then we detail how we have proceeded in implementing them in a scalable manner, providing code snippets to illustrate the implementation.

Finally, in Chapter 5 we perform a comparative study of the algorithms we have developed, also measuring them against some well-known learning algorithms that are already implemented in Spark. This study is done by executing the algorithms on a few data sets with a large quantity of instances, using a dedicated cluster server with enough capacity. We present the execution results in different formats and later discuss how the algorithms have performed.

2 Fuzzy logic and fuzzy systems

We begin this work by presenting the fundamentals of *fuzzy theory*. This theory was developed by computer scientist and mathematician Lofti A. Zadeh around 1965 (see [Zad65]). He felt the need to introduce a concept that would enable the treatment of systems in an imprecise manner, due to lack of information or simply because of some inherent ambiguity. At the same time, he wanted this notion to be mathematically coherent and tractable, and to have a strong theoretical base from which to expand to practical applications.

With these ideas in mind he established the concepts of *fuzzy sets* and *fuzzy reasoning*, providing a robust foundation for developing various methods to deal with uncertainty and impreciseness. These methods are widely used nowadays in the construction of control systems and the modelling of situations which involve a certain degree of vagueness, uncertainty or ambivalence.

The main references for this chapter are [CP00] and [JSM97], from which we extract some basic definitions and a handful of results to illustrate the extent of this theory.

2.1 Fuzzy set theory

2.1.1 Basic concepts

Let X be a space of objects, known as the *universe of discourse*. Where necessary in our discussion, we may assume that $X = \mathbb{R}$. We begin with the most basic definition, which sets the ground rules for all the theory that will be developed.

Definition 2.1 (Fuzzy set). A fuzzy set A in X is the set of ordered pairs

$$A = \{(x, \mu_A(x)) \mid x \in X\},$$

where $\mu_A(x)$ is the *membership function* (MF) for the fuzzy set A. It maps each element of X to a membership grade in $[0, 1]$.

Thus, a fuzzy set is uniquely specified by its membership function, which measures the degree to which every element of the universe belongs to this particular set. Because of this, an abuse of notation will sometimes be allowed when using only a membership

function to denote the underlying fuzzy set. An alternative way of denoting a fuzzy set A is as follows:

$$A = \sum_{x_i \in X} \mu_A(x_i)/x_i, \quad \text{if } X \text{ is discrete,}$$
$$A = \int_X \mu_A(x)/x, \quad \text{if } X \text{ is a continuous space.}$$

Remark. If $\mu_A(x)$ is restricted to $\{0, 1\}$, then A becomes a set in the classical sense (which will be referred to as a *crisp set*). We note that the fundamental distinction between a crisp set and a fuzzy set is that the latter introduces a measure of uncertainty to model the notion of an element belonging to a certain group, which can be useful in real-life scenarios.

We now present a couple of concepts that help summarize the characteristics of our newly defined fuzzy sets.

Definition 2.2 (Support). The support of a fuzzy set A is the set of all points $x \in X$ such that $\mu_A(x) > 0$.

Definition 2.3 (Core). The core of a fuzzy set A is the set of all points $x \in X$ such that $\mu_A(x) = 1$.

Definition 2.4 (Normality). A fuzzy set A is *normal* if its core is non-empty, that is, if there exists some $x \in X$ such that $\mu_A(x) = 1$.

Definition 2.5 (Crossover points). A crossover point of a fuzzy set is a point $x \in X$ for which $\mu_A(x) = 0.5$:

$$\text{crossover}(A) = \{x \mid \mu_A(x) = 0.5\}.$$

Definition 2.6 (Fuzzy singleton). A fuzzy set whose support is a single point in X with $\mu_A(x) = 1$ is called a fuzzy singleton.

Definition 2.7 (α-cut, strong α-cut). The α-cut or *α-level set* of a fuzzy set A is defined for $\alpha \in [0, 1]$ as follows:

$$A_\alpha = \{x \mid \mu_A(x) \geq \alpha\}.$$

Similarly, the strong α-cut, or *strong α-level set* A'_α is defined by replacing '$\geq$' with '$>$'.

Remark. With the notation introduced for level sets, it is clear that $\text{support}(A) = A'_0$ and $\text{core}(A) = A_1$.

The concept of level sets plays an important role in the theory of fuzzy sets, as they provide a way of characterizing these entities using only crisp sets. This behaviour is exemplified in the following theorem.

Theorem 2.8 (Resolution principle). *Every MF can be decomposed into a combination of its α-level sets' MFs:*

$$\mu_A(x) = \sup_\alpha \min\left\{\alpha, \mu_{A_\alpha}(x)\right\}.$$

Remember that A_α is a crisp set and thus μ_{A_α} is understood as its characteristic function, taking values in $\{0, 1\}$.

Proof. Let $x \vee y$ denote the operation $\max\{x, y\}$ and $x \wedge y$ denote $\min\{x, y\}$. Then it follows that for every $x \in X$,

$$\begin{aligned}
\sup_\alpha \left\{\alpha \wedge \mu_{A_\alpha}(x)\right\} &= \sup_{\alpha\in[0,\mu_A(x)]} \left\{\alpha \wedge \mu_{A_\alpha}(x)\right\} \vee \sup_{\alpha\in[\mu_A(x),1]} \left\{\alpha \wedge \mu_{A_\alpha}(x)\right\} \\
&= \sup_{\alpha\in[0,\mu_A(x)]} \{\alpha \wedge 1\} \vee \sup_{\alpha\in[\mu_A(x),1]} \{\alpha \wedge 0\} \\
&= \sup_{\alpha\in[0,\mu_A(x)]} \{\alpha\} \vee 0 \\
&= \mu_A(x).
\end{aligned}$$

■

Lastly, we finish this section with yet another set of definitions, concerned mainly with the shape that fuzzy sets can adopt. These concepts will be useful later on when describing specific MFs.

Definition 2.9 (Convexity). A fuzzy set A is *convex* if for any $x_1, x_2 \in X$ and any $\lambda \in [0, 1]$, it holds that

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq \min\{\mu_A(x_1), \mu_A(x_2)\}.$$

Alternatively, A is convex if all its α-level sets are convex (i.e. composed of a single line segment only).

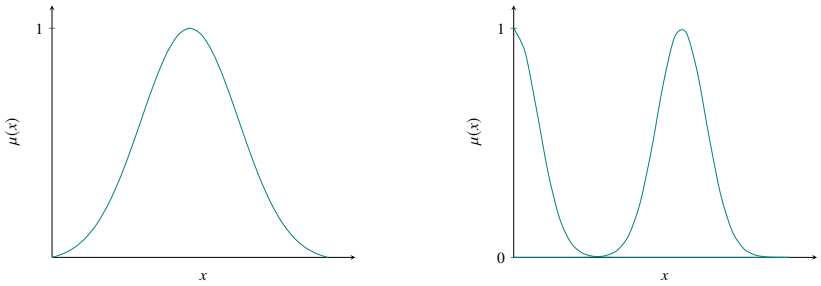
Definition 2.10 (Fuzzy number). A fuzzy number is a fuzzy set in $\mathbb{R}$ that is both normal and convex.

Definition 2.11 (Symmetry). A fuzzy set A is said to be *symmetric* around a point $c \in X$ if it satisfies

$$\mu_A(c + x) = \mu_A(c - x), \quad \text{for all } x \in X.$$

Definition 2.12 (Open left, open right, closed). A fuzzy set A is said to be *open left* if $\lim_{x\to-\infty} \mu_A(x) = 1$ and $\lim_{x\to\infty} \mu_A(x) = 0$; *open right* if $\lim_{x\to-\infty} \mu_A(x) = 0$ and $\lim_{x\to\infty} \mu_A(x) = 1$; and *closed* if $\lim_{x\to\pm\infty} \mu_A(x) = 0$.

In most applications it makes sense to work with convex membership functions, since they are easy to visualize and interpret. An example visualization to help us differentiate between convex and non-convex fuzzy sets can be seen in Figure 2.1.



(a) Convex fuzzy set **(b)** Non-convex fuzzy set

Figure 2.1: Representation of membership functions of convex and non-convex fuzzy sets.

2.1.2 Set-theoretic operations

Most set-theoretic operations in the classical theory translate to fuzzy set theory in a natural way. We present a list of some of the most basic of these operations and their reinterpreted definitions in our setting, while introducing some of the notation that will be used in what follows.

Definition 2.13 (Subset). If A and B are fuzzy sets, we say that A is a subset of B, and we write $A \subseteq B$, if $\mu_A(x) \leq \mu_B(x)$ for all $x \in X$.

Definition 2.14 (Union). The union of two fuzzy sets A and B is a fuzzy set $C = A \cup B$, whose MF is $\mu_C(x) = \max\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \vee \mu_B(x)$. Alternatively, we can say that C is the "smallest" fuzzy set that contains both A and B.

Definition 2.15 (Intersection). The intersection of two fuzzy sets A and B is a fuzzy set $C = A \cap B$, whose MF is $\mu_C(x) = \min\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \wedge \mu_B(x)$. As in the case of the union, C can be viewed as the "largest" fuzzy set contained in both A and B.

Definition 2.16 (Complement). The complement of a fuzzy set A, denoted by $\bar{A}$, is another fuzzy set defined by $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$.

Definition 2.17 (Cartesian product and co-product). Let A and B be fuzzy sets in X and Y, respectively. The *Cartesian product* $A \times B$ is a fuzzy set in $X \times Y$ given by

$$\mu_{A\times B}(x, y) = \min\{\mu_A(x), \mu_B(y)\} = \mu_A(x) \wedge \mu_B(y).$$

Similarly, the *Cartesian co-product* $A + B$ is a fuzzy set in $X \times Y$ which has the membership function

$$\mu_{A+B}(x, y) = \max\{\mu_A(x), \mu_B(y)\} = \mu_A(x) \vee \mu_B(y).$$

As it turns out, there are various ways of defining these basic operations in a fuzzy context, so we will refer to the operators used above as the *classical* or *standard fuzzy operators*. Other operators are discussed briefly in Subsection 2.1.5. The choice of a specific fuzzy operator will depend on the specific problem we try to model, since different operators will most likely yield different outcomes and conclusions.

As with any new mathematical structure, one of the first steps after studying its basic properties is to study which operations preserve this structure, and how to generate new entities from existing ones. Regardless of the concrete operators employed, the operations just described provide us with the ability of generating new fuzzy sets from existing ones, while maintaining a reasonable interpretation of what they represent. Some of them will even preserve interesting properties of fuzzy sets, as seen in next result, whose proof can be derived by merely writing down the definitions of the concepts involved.

Proposition 2.18. *An arbitrary intersection of convex fuzzy sets is a convex fuzzy set.*

Continuing along the same lines, we arrive at an important theorem regarding the creation of new fuzzy sets, stated by Zadeh himself. In short, it says that we can generate a fuzzy set that represents any structure, concept or relation that can be described with a (classical) mapping. A detailed introduction to this result and some insight as to its usefulness is given by E. Kerre in [Ker11].

Theorem 2.19 (Extension principle). *Let f be a mapping from an n-dimensional product space $X_1 \times \cdots \times X_n$ to a one-dimensional universe Y, and suppose $A_1, \ldots, A_n$ are n fuzzy sets in $X_1, \ldots, X_n$, respectively. Then f induces a fuzzy set B in Y defined by*

$$\mu_B(y) = \begin{cases} \sup\limits_{(x_1,\ldots,x_n)=f^{-1}(y)} \min\limits_i \{\mu_{A_i}(x_i)\}, & \text{if } f^{-1}(y) \neq \emptyset, \\ 0, & \text{if } f^{-1}(y) = \emptyset. \end{cases}$$

Corollary 2.20. *Let $f : X \to Y$ be a one-to-one mapping between one-dimensional spaces, and let μ_A represent a fuzzy set in X. Then, a fuzzy set $\tilde{f}(A)$ is induced in Y, given by*

$$\tilde{f}(A) = \int_X \mu_A(x)/f(x).$$

Example 2.21. Let $+ : \mathbb{R}^2 \to \mathbb{R}$ represent the usual addition operation. Using the extension principle we can define the sum of two fuzzy sets A and B in $\mathbb{R}$ as a new fuzzy set $A + B$ with membership function given by

$$\mu_{A+B}(z) = \sup \{\mu_A(x) \wedge \mu_B(y) \mid z = x + y\}, \quad \forall z \in \mathbb{R}.$$

To conclude this brief exposition of operations between fuzzy sets, we now consider the general setting in which we have a multidimensional universe (i.e, a product space). In this case, it can be useful to think of fuzzy sets as a *relation* among tuples of elements, each of which has a membership grade to the implied fuzzy set.

Definition 2.22 (Fuzzy relation). Let X and Y be two universes of discourse. Then the fuzzy set

$$\mathcal{R} = \{((x, y),\ \mu_{\mathcal{R}}(x, y)) \mid (x, y) \in X \times Y\}$$

represents a *binary fuzzy relation* in $X \times Y$. A generalization to n-ary relations is straightforward.

If X and Y are both discrete, we can express a fuzzy relation as a relation matrix $\mathcal{R} = (r_{ij})$, where r_{ij} is the membership grade between the ith element of X and the jth element of Y. As with fuzzy sets, we can operate with relations to produce new fuzzy sets, again using the classical operators previously described.

Definition 2.23 (Max-min composition). Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two fuzzy relations defined on $X \times Y$ and $Y \times Z$, respectively. The max-min composition of $\mathcal{R}_1$ and $\mathcal{R}_2$ is a fuzzy set $\mathcal{R}_1 \circ \mathcal{R}_2$ in $X \times Z$ defined by

$$\mu_{\mathcal{R}_1 \circ \mathcal{R}_2}(x, z) = \max_y \{\mu_{\mathcal{R}_1}(x, y) \wedge \mu_{\mathcal{R}_2}(y, z)\}.$$

Remark. If $\mathcal{R}_1$ and $\mathcal{R}_2$ are relation matrices, the calculation of $\mathcal{R}_1 \circ \mathcal{R}_2$ is almost the same as matrix multiplication, except that $\times$ and $+$ are replaced by $\wedge$ and $\vee$, respectively.

Proposition 2.24. *Let $\mathcal{R}, \mathcal{S}$ and $\mathcal{T}$ be binary relations on $X \times Y$, $Y \times Z$ and $Z \times W$, respectively, and let $\circ$ denote the max-min composition. Then, the following properties hold:*

(*i*) *(Associativity)* $\mathcal{R} \circ (\mathcal{S} \circ \mathcal{T}) = (\mathcal{R} \circ \mathcal{S}) \circ \mathcal{T}$.
(*ii*) *(Distributivity over union)* $\mathcal{R} \circ (\mathcal{S} \cup \mathcal{T}) = (\mathcal{R} \circ \mathcal{S}) \cup (\mathcal{R} \circ \mathcal{T})$.
(*iii*) *(Weak distributivity over intersection)* $\mathcal{R} \circ (\mathcal{S} \cap \mathcal{T}) \subset (\mathcal{R} \circ \mathcal{S}) \cap (\mathcal{R} \circ \mathcal{T})$.
(*iv*) *(Monotonicity)* $\mathcal{S} \subset \mathcal{T} \implies \mathcal{R} \circ \mathcal{S} \subset \mathcal{R} \circ \mathcal{T}$.

Proof. All four properties are easy to verify and follow directly from the well-known properties of the operators $\wedge$ and $\vee$. ∎

As one could expect, there are many possible definitions for a composition operator. For instance, another type of composition used in the literature (see [Mar04; LF01]) is the *max-product composition*.

Definition 2.25 (Max-product composition). Assuming the same notation as above, the max-product composition is defined as follows:

$$\mu_{\mathcal{R}_1 \circ \mathcal{R}_2}(x, z) = \max_y \{\mu_{\mathcal{R}_1}(x, y)\mu_{\mathcal{R}_2}(y, z)\}.$$

2.1.3 Parametrized one-dimensional MFs

In this section, we describe the classes of parametrized functions commonly used to define one-dimensional fuzzy sets.

Definition 2.26 (Triangular MF). Three parameters determine the x-coordinates of the three corners of a triangle:

$$\text{triangle}(x; a, b, c) = \max\left\{\min\left\{\frac{x-a}{b-a}, \frac{c-x}{c-b}\right\}, 0\right\}.$$

Definition 2.27 (Trapezoidal MF). Four parameters determine the x-coordinates of the four corners of a trapezoid:

$$\text{trapezoid}(x; a, b, c, d) = \max\left\{\min\left\{\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right\}, 0\right\}.$$

Definition 2.28 (Gaussian MF). It is determined by its center (c) and its width (σ) as follows:

$$\text{gaussian}(x; c, \sigma) = \exp\left(-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2\right).$$

Definition 2.29 (Generalized bell MF). It is specified by three parameters:

$$\text{bell}(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}},$$

where b is usually positive.

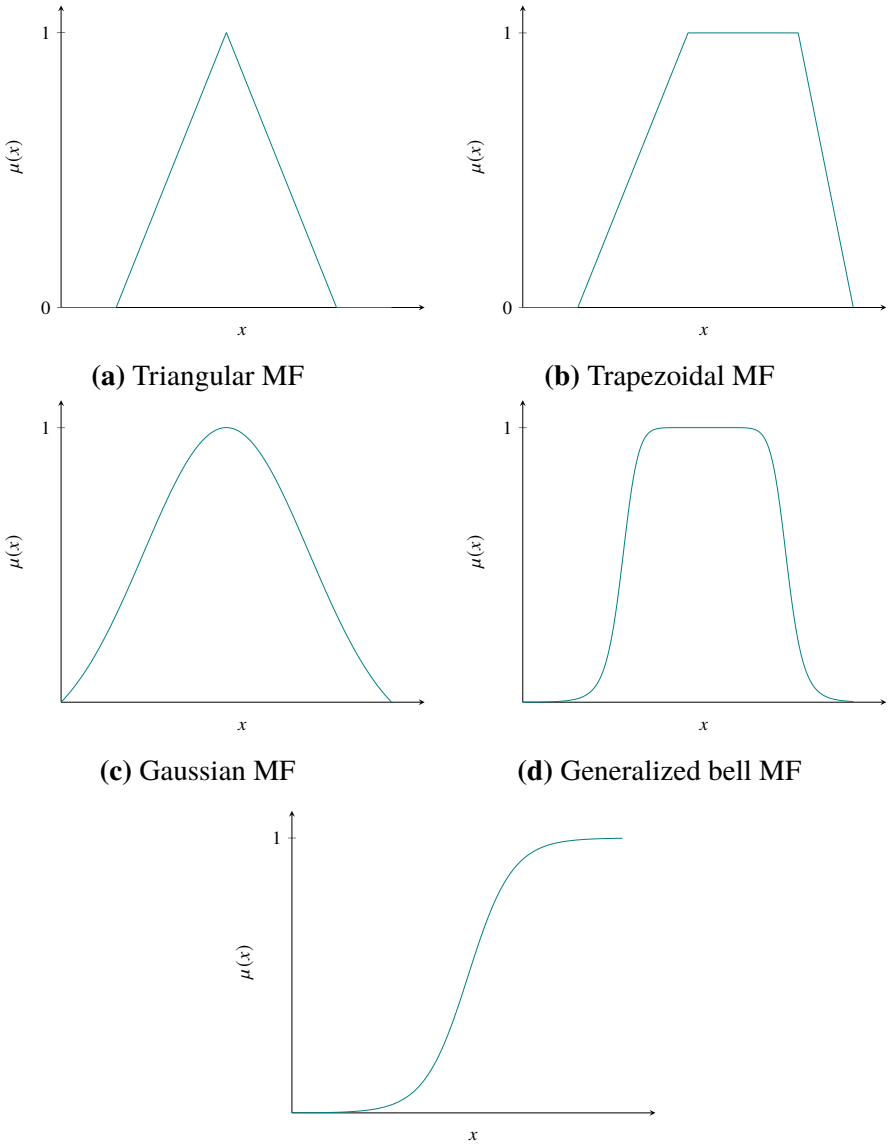
Definition 2.30 (Sigmoidal MF). A sigmoidal MF is defined by

$$\text{sig}(x; a, c) = \frac{1}{1 + e^{-a(x-c)}},$$

where a controls the slope at the crossover point $x = c$.

Remark. Depending on the sign of the parameter a, a sigmoidal MF is inherently open right or left, and thus asymmetric. Closed and asymmetric MFs can be synthesized using either the absolute difference or the product of two sigmoidal functions.

Triangular and trapezoidal membership functions are used because of their simplicity, both computational and interpretative. Nonetheless, they are composed of line segments and thus are not differentiable. This is where smooth MFs such as the Gaussian MF come into play, proving their usefulness in certain scenarios such as neuro-fuzzy systems. Some of these membership functions are shown in Figure 2.2.



(a) Triangular MF

(b) Trapezoidal MF

(c) Gaussian MF

(d) Generalized bell MF

(e) Sigmoidal MF

Figure 2.2: Examples of different membership functions.

2.1.4 Two-dimensional MFs

While it is possible to define membership functions in two or more dimensions, it is often desirable to begin in one dimension and build from there, effectively *extending* the one-dimensional MFs defined above (or any other).

Definition 2.31 (Cylindrical extension). If A is a fuzzy set in X and Y is another universe, then the cylindrical extension of A in $X \times Y$ is defined by

$$c(A) = \int_{X \times Y} \mu_A(x)/(x, y).$$

We can also perform the inverse operation, that is, decompose a two-dimensional MF in two one-dimensional MFs.

Definition 2.32 (Projections). Let R be a two-dimensional fuzzy set on $X \times Y$. Then the projections of R onto X and Y are defined as

$$R_X = \int_X \max_y \mu_R(x, y)/x, \quad R_Y = \int_Y \max_x \mu_R(x, y)/y.$$

Definition 2.33 (Composite MF). A two-dimensional MF is said to be *composite* if it can be decomposed as an analytic expression of two MFs of one dimension; otherwise it is *non-composite*.

The Gaussian MF is arguably the most notable example of a composite MF, since it is a well known result that every n-dimensional Gaussian function is the product of n one-dimensional Gaussian functions, scaled by some constant factor.

2.1.5 Fuzzy complement, T-norm and T-conorm

Lastly, we explore the generalization of the classical fuzzy operators that have been described throughout this section.

Definition 2.34 (Complement operator). A fuzzy complement operator is a continuous function $N : [0, 1] \to [0, 1]$ which meets the following axiomatic requirements:

(i) *(Boundary)* $N(0) = 1$ and $N(1) = 0$.
(ii) *(Monotonicity)* $N(a) \geq N(b)$ if $a \leq b$.
(iii) *(Involution)* $N(N(a)) = a$.

Remark. Sometimes an operator verifying only (i) and (ii) is considered a fuzzy complement operator.

Example 2.35. One can routinely verify that the following two operators are fuzzy complement operators.

(i) *(Sugeno's complement* [Sug93]) $N_s(a) = \dfrac{1 - a}{1 + sa}$, where $s > -1$.
(ii) *(Yager's complement* [Yag79]) $N_w(a) = (1 - a^w)^{1/w}$, where $w > 0$.

Definition 2.36 (T-norm operator). The intersection of two fuzzy sets A and B is specified in general by a function $T : [0,1] \times [0,1] \to [0,1]$, referred to as a T-norm operator, which aggregates two membership grades,

$$\mu_{A\cap B}(x) = T(\mu_A(x), \mu_B(x)),$$

and verifies several basic properties, namely:

(*i*) *(Boundary)* $T(0,0) = 0$ and $T(a,1) = a$.
(*ii*) *(Monotonicity)* $T(a,b) \leq T(c,d)$ if $a \leq c$ and $b \leq d$.
(*iii*) *(Commutativity)* $T(a,b) = T(b,a)$.
(*iv*) *(Associativity)* $T(a,T(b,c)) = T(T(a,b),c)$.

For the sake of completeness, here are four of the most frequently used T-norm operators:

(*i*) *Minimum:* $T_{min}(a,b) = a \wedge b$.
(*ii*) *Algebraic product:* $T_{ap}(a,b) = ab$.
(*iii*) *Bounded product:* $T_{bp}(a,b) = 0 \vee (a+b-1)$.
(*iv*) *Drastic product:* $T_{dp}(a,b) = \begin{cases} a, & \text{if } b = 1, \\ b, & \text{if } a = 1, \\ 0, & \text{otherwise.} \end{cases}$

Definition 2.37 (T-conorm operator). Similarly, the union of two fuzzy sets A and B is specified in general by a function $S : [0,1] \times [0,1] \to [0,1]$, known as a T-conorm operator, which works in the same way as a T-norm, except that the boundary conditions are now $S(1,1) = 1$ and $S(0,a) = a$.

Corresponding to the four T-norm operators previously defined, we have the following T-conorm operators:

(*i*) *Maximum:* $S_{max}(a,b) = a \vee b$.
(*ii*) *Algebraic sum:* $S_{as}(a,b) = a + b - ab$.
(*iii*) *Bounded sum:* $S_{bs}(a,b) = 1 \wedge (a+b)$.
(*iv*) *Drastic sum:* $S_{ds}(a,b) = \begin{cases} a, & \text{if } b = 0, \\ b, & \text{if } a = 0, \\ 1, & \text{otherwise.} \end{cases}$

Theorem 2.38 (Generalized De Morgan's Law). *T-norms and T-conorms are duals which support the generalization of De Morgan's law:*

$$T(a,b) = N(S(N(a),N(b))),$$
$$S(a,b) = N(T(N(a),N(b))).$$

In fact, the eight operators showed above are dual in the sense of the generalized De Morgan's law.

2.2 Fuzzy logic

In this section we will study the methods and principles of *fuzzy reasoning*, and present fuzzy logic as a multi-valued logic that generalizes the classical two-valued logic. If we think of the predicates *true* and *false* as being codified as 0 and 1, respectively, we can understand how to derive a *fuzzy* logic theory in which every predicate has a *truth grade* quantified by an appropriate membership function.

The whole new range of possible truth values gives something of a gray area in which propositions can fall. This is especially interesting in situations which involve a high degree of uncertainty, and thus its structure is difficult to capture with crisp values (if possible at all). The first example that comes to mind is what is known as *natural language processing*, or the modelling of human speech to be ultimately processed by a computer. It is precisely this example that inspires the definitions and concepts that will be presented next, and it can be viewed as the starting point of the subsequent theory.

2.2.1 Linguistic variables

Definition 2.39 (Linguistic Variable). A linguistic variable is a quintuple $(x, T(x), X, G, M)$, in which:

- (i) The symbol x represents the name of the variable.
- (ii) $T(x)$ is the *term set* of x, that is, the set of its *linguistic values*.
- (iii) X is, as usual, the universe of discourse.
- (iv) G is a *syntactic rule* which generates the terms in $T(x)$.
- (v) M is a semantic rule which associates with each linguistic value A its *meaning* $M(A)$, which in turn is nothing more than a fuzzy set in X.

Example 2.40. To help clarify the preceding definition, consider the concept of *age* interpreted as a linguistic variable. Then its term set $T(age)$ could be something like the following:

$$T(age) = \left\{ \begin{array}{l} \textit{young, not young, very young, not very young, } \ldots, \\ \textit{middle aged, not middle aged, } \ldots, \\ \textit{old, not old, very old, more or less old, not very old, } \ldots, \\ \textit{not very young and not very old,} \ldots \end{array} \right\},$$

where each term in $T(age)$ is characterized by a fuzzy set in a universe $X = [0, 100]$.

The syntactic rule refers to the way that linguistic values are generated. We can see that the term set of the previous example consists of several *primary terms* (*young, middle aged, old*) modified by the *negation* (*not*) and/or *adverbs* (*very, more or less, quite,* and so forth), and linked by *connectives* such as *and, or, either* and *neither*.

We shall come up with a way of treating all these modifiers as operators that change the meaning of their operands in a specified, context-independent fashion.

Definition 2.41 (Concentration and dilation). Let A be a linguistic value characterized by a fuzzy set with membership function μ_A. Then A^k is interpreted as a modified version of the original linguistic value expressed as

$$A^k = \int_X (\mu_A(x))^k / x.$$

In particular, the operation of *concentration* is defined as $\mathrm{CON}(A) = A^2$, while that of *dilation* is expressed by $\mathrm{DIL}(A) = A^{0.5}$.

Conventionally, we take $\mathrm{CON}(A)$ and $\mathrm{DIL}(A)$ to be the result of applying the modifiers *very* and *more or less*, respectively, to the linguistic term A. Following the definitions in the previous chapter, we can interpret the negation operator *not* and the connectives *and*, *or* as follows:

$$\begin{aligned}
\mathrm{NOT}(A) = \neg A &= \int_X 1 - \mu_A(x)/x, \\
A \,\mathrm{AND}\, B = A \cap B &= \int_X \mu_A(x) \wedge \mu_B(x)/x, \\
A \,\mathrm{OR}\, B = A \cup B &= \int_X \mu_A(x) \vee \mu_B(x)/x.
\end{aligned}$$

Definition 2.42 (Contrast intensification). The operation of contrast intensification on a linguistic value A is defined by

$$\mathrm{INT}(A) = \begin{cases} 2A^2, & \text{if } 0 \le \mu_A(x) \le 0.5, \\ \neg 2(\neg A)^2, & \text{if } 0.5 \le \mu_A(x) \le 1. \end{cases}$$

This operator has the effect of reducing the fuzziness of a linguistic value. In the extreme case of repeated applications, the fuzzy set becomes a crisp set with boundaries at the crossover points.

Lastly, when we define MFs of linguistic values in a term set, it is intuitively reasonable to have these MFs roughly satisfy the requirement of orthogonality, which is described next.

Definition 2.43 (Orthogonality). A term set $T = \{t_1, \ldots, t_n\}$ of a linguistic variable x on the universe X is *orthogonal* if it fulfills the following property:

$$\sum_{i=1}^{n} \mu_{t_i}(x) = 1, \quad \forall x \in X,$$

where each t_i is a convex and normal fuzzy set defined on X.

2.2.2 Fuzzy if-then rules

The core idea behind fuzzy reasoning is that the classical *if-then* rules of inference used in many systems can be reinterpreted under a fuzzy perspective, taking advantage of all the properties and easiness of interpretation that come with it.

Definition 2.44 (Fuzzy if-then rules). If A and B are linguistic values, a fuzzy if-then rule associated with them assumes the form

$$\text{if } x \text{ is } A \text{ then } y \text{ is } B,$$

where "x is A" is the *antecedent* part and "y is B" is the *consequent* part. Sometimes we will abbreviate such a rule as $A \to B$.

Some examples of fuzzy rules include *if the patient's temperature is high, give them a moderate dose of medicine*, or *tall people will get a big discount*. These rules are most useful, as we have already pointed out, when the concepts we are trying to model are difficult to describe in exact terms: is there a universally accepted height from which a person is considered to be tall? Is a body temperature of 38° indicative of a problem but 37.9° is not?

Since a fuzzy if-then rule models a relation between two variables x and y, it makes sense to interpret $A \to B$ as a fuzzy relation $\mathcal{R}$ given by a membership function $\mu_{\mathcal{R}} = f(\mu_A(x), \mu_B(y))$, for an appropriate choice of an implication function f (usually a T-norm operator). Now we derive the basic operation of fuzzy reasoning, which consists of an *approximate modus ponens*. Starting from the fuzzy rule *if x is A then y is B*, if we have the premise x is A', where A' means an approximation of A, we can conclude that y is B', with B' being an approximation of B. In other words, if x is *A to a certain degree*, then y is *B to a certain (possibly different) degree*. We give a precise meaning and structure to this process in the following definition.

Definition 2.45 (Fuzzy reasoning). Let X, Y be two universes of discourse and let A, A' and B be fuzzy sets in X, X and Y, respectively. If $A \to B$ is a fuzzy implication, then a fuzzy set B' is induced on Y, defined by

$$B' = A' \circ (A \to B),$$

where $\circ$ is the max-min composition (see Definition 2.23), or equivalently,

$$\mu_{B'}(y) = \vee_x(\mu_{A'}(x) \ \wedge \mu_{A\to B}(x, y)).$$

It goes without saying that we can have as many fuzzy rules as we want, and our ultimate goal will be to aggregate all of them together and produce a coherent reasoning system. Even though we have only considered the simplest case with one antecedent and one consequent, we can treat rules with multiple antecedents using the cartesian product operator, and rules with multiple consequent as the union of many rules with just one consequent. For example, we can have the following line of reasoning:

$$\begin{gathered} x \text{ is } A' \text{ and } t \text{ is } B' \\ \text{if } x \text{ is } A_1 \text{ and } t \text{ is } B_1 \text{ then } z \text{ is } C_1 \\ \text{if } x \text{ is } A_2 \text{ and } t \text{ is } B_2 \text{ then } z \text{ is } C_2 \\ \hline z \text{ is } C' \end{gathered}$$

In this case, we interpret the first rule as a relation $\mathcal{R}_1 = A_1 \times B_1 \to C_1$ and the second one as $\mathcal{R}_2 = A_2 \times B_2 \to C_2$. Then, it can be checked that the induced fuzzy set C' is given by

$$C' = (A' \times B') \circ (\mathcal{R}_1 \circ \mathcal{R}_2) = C'_1 \cup C'_2,$$

where C'_1 and C'_2 are the fuzzy sets inferred from the first and second rule, respectively.

2.3 Fuzzy inference systems

Now that we have gone over all the key elements in the theory of fuzzy reasoning, we are prepared to understand what a *fuzzy inference system* (FIS) is. In summary, it is a system that maps an input space to an output space, using fuzzy techniques along the way as the main reasoning mechanism. Generally speaking it is made of four clearly differentiated parts:

1. A *fuzzification module*, which converts crisp inputs into fuzzy sets, with some predefined membership functions.
2. A *rule base* that contains a set of fuzzy if-then rules.
3. An *inference engine* that takes some input and employs the rules in the rule base to infer the corresponding output, using the fuzzy reasoning scheme described in the previous section. The membership functions in the output space are also predefined.
4. A *defuzzifying method* used to convert the output back to a crisp value, if necessary.

A positive quality of a FIS is that its rule base can be easily modified with the addition of new rules or the adjustment of existing ones. The rules in the rule base can be defined by an expert based on their knowledge of the specific problem we are tackling, but they can also be *learned from data*. This fact opens the door to the treatment of fuzzy systems with the techniques and learning algorithms that have been emerging over the course of the last decades in this field. This is precisely what we pursue: to build a fuzzy system that can learn and benefit from enormous amounts of data. This process will be discussed in depth in Chapter 4.

One of the most important things to take into account is that the choice of membership function is subjective, but not arbitrary. It heavily depends on the problem we are trying to solve, and selecting the appropriate MF for a given task usually gives much better results than choosing one either at random or one that gave good results on some entirely different problem.

Even though we have described every step of the inferring process in this and the previous sections, we summarize them below for completeness.

1. In the first place, inputs to the system are *fuzzified* and the membership grade of each input to the predefined fuzzy sets are calculated.
2. Then, the *firing strength* of each rule is computed, which usually consists on performing the fuzzy AND operation between the antecedents.
3. Now the firing strength of each rule is combined with the consequent part to form an overall degree of satisfaction of the rule, known as the *qualified consequent*. This is done as we described when we talked about implication operators, which in turn are commonly implemented with T-norm operators (e.g. the $\wedge$ operator).
4. We aggregate all qualified consequents and choose the one that best represents the input value, for which the OR operator is the standard choice.
5. Finally, we employ the aggregated output MF to obtain a crisp value as the output, i.e. a defuzzified value.

Because of the use of fuzzy rules in the reasoning process, these systems are often called Fuzzy Rule-Based Systems (FRBS).

2.3.1 Fuzzy system types

To bring this exposition about fuzzy theory to an end, we present the two most common types of fuzzy systems used in practical applications. These two systems differ mainly in how they treat the output values.

Definition 2.46 (Mamdani fuzzy inference system). A Mamdani fuzzy inference system (see [MA75]) is a FIS that takes an input $X = (X_1, \ldots, X_n)$ and produces an output $Y = D(Y_1, \ldots, Y_m)$, using rules of the form

$$\begin{gathered}\text{if } X_1 \text{ is } A_1 \text{ and } X_2 \text{ is } A_2 \text{ and } \dots \text{ and } X_n \text{ is } A_n \\ \text{then } Y_1 \text{ is } B_1 \text{ and } Y_2 \text{ is } B_2 \text{ and } \dots \text{ and } Y_m \text{ is } B_m,\end{gathered}$$

together with a defuzzification function $D : [0,1]^m \to \mathbb{R}^m$ to compute the final crisp values.

The most common defuzzification method is the *centroid of area* method, or COA for short. It extracts a representative value from a fuzzy set, which can be thought of as the center of gravity of the area under its membership function. For a one-dimensional fuzzy set A in a universe Z, it assumes the form

$$z = \frac{\int_Z z\mu_A(z)\,dz}{\int_Z \mu_A(z)\,dz}.$$

Other defuzzification methods are studied in the literature. A non-exhaustive classification of such methods can be found in [LK99].

Example 2.47. A typical single-input single-output Mamdani fuzzy model could be expressed as follows:

$$\begin{cases} \text{If } X \text{ is small then } Y \text{ is small.} \\ \text{If } X \text{ is medium then } Y \text{ is large.} \\ \text{If } X \text{ is large then } Y \text{ is very large.} \end{cases}$$

The other type of FIS that we will study is one in which the output is not a fuzzy set, but a crisp function of the input values. It is aimed at generating fuzzy rules from a given input-output data set, and is appropriate for solving regression problems.

Definition 2.48 (TSK fuzzy inference system). A TSK fuzzy inference system (proposed by Takagi, Sugeno and Kang [TS85; SK88]) is a FIS that takes an input $X = (X_1, \dots, X_n)$ and produces an output $Y = (Y_1, \dots, Y_m)$, using rules of the form

$$\begin{gathered}\text{if } X_1 \text{ is } A_1 \text{ and } X_2 \text{ is } A_2 \text{ and } \dots \text{ and } X_n \text{ is } A_n \\ \text{then } Y_1 \text{ is } f_1(X_1, \dots, X_n) \text{ and } Y_2 \text{ is } f_2(X_1, \dots, X_n) \text{ and } \dots \text{ and } Y_m \text{ is } f_m(X_1, \dots, X_n),\end{gathered}$$

where each f_i is a crisp function of the input values (usually a polynomial).

One thing to take into account is that the overall output is obtained via a *weighted average* of the output of each rule (the weights being their firing strength), since now there are no fuzzy sets in the consequent part. This also eliminates the need for a defuzzification module, and thus improves the speed of the system.

Example 2.49. A typical single-input TSK fuzzy model could be expressed as follows:

$$\begin{cases} \text{If } X \text{ is small then } Y = 0.1X + 6. \\ \text{If } X \text{ is medium then } Y = -0.5X - 3. \\ \text{If } X \text{ is large then } Y = X + 2. \end{cases}$$

3 Big Data

Since the beginning of time, we humans have always had the need to store information of some kind for a variety of applications: to keep track of the grain stock, to make a census of the population of a certain area, or simply to make a note for latter use. But it was not until the start of the so-called *era of information* that the storage of data in a digital format became a reality and a part of everyday life. From then on, there has been an ever-increasing need for storage capacity, growing from kilobytes to megabytes to gigabytes and beyond in a mere few decades. But in recent years, the quantity of data that we collectively manage has grown so large that it is even hard to comprehend its scale. To put things into perspective, the total estimate of data ever created, captured or replicated amounts to 18 zettabytes (more than 10^{13} gigabytes) [RRG18]. That is indeed *a lot* of data.

Given the enormous quantity of information that we are constantly generating, it should not come as a surprise that the conventional ways of storing, manipulating and analyzing this information fail to accomplish their goal: they have become outdated. This is why the search for new approaches to this problem has recently gained traction in industry as well as in academia; we require new architectures and methods for dealing with such a big volume of data. It is precisely for this reason that the term *Big Data* was coined. It refers to the manipulation and the extracting of information from data sets so large that they exceed the capabilities of most modern operating systems and hardware, and cannot be handled with the usual data-processing methods within a reasonable elapsed time.

3.1 Fundamental characteristics of Big Data

First of all we need to clarify what we understand by Big Data. While the precise definition varies somewhat in the literature (see [KM16] for example), there are a series of characteristics that any Big Data environment (problem, data set, etc.) should present in order for it to be considered as such. Coupled with these characteristics there are certain considerations to take into account, either as a direct result or as a side effect of working with large amounts of data. We explore all of these elements over the next subsections.

It should be noted that we will not consider a Big Data question the act of merely storing information with no other purpose than to preserve it. There should always be a particular problem (in the broad sense of the term) that is being solved by means of the data.

3.1.1 The five V's

The concept of the various *V's of Big Data* has become a popular way of summarizing the principal characteristics that are desirable in a Big Data context. While the exact number and definition of these traits varies from author to author, they all share the same underlying principle: to capture the essence of what makes Big Data *big*. We list below the ones that are deemed the most important, in no particular order.

1. *Volume*. It refers to the quantity of generated and stored data. It must be sufficient to draw meaningful conclusions and gain insight into the problem at hand. A necessary condition for establishing that some data has enough volume is that it exceeds the standard capacities of modern computers, which as of the year 2020 is in the terabytes.
2. *Velocity*. It relates to the speed or rate at which the data is generated or received. In some cases it includes real-time data processing, also known as *streaming data*. In general, it is expected that the information be produced continually (for example by means of a sensor of some kind).
3. *Variety*. The type and nature of the data also plays an important role. It includes both structured and unstructured data, drawn from various sources such as text, image, audio, etc.
4. *Veracity*. We can think of it as a way of measure how trustworthy the data is. The quality of the data can be affected by multiple factors such as inconsistencies, incompleteness or ambiguity, among others.
5. *Value*. It describes the added value that the collected data may have in the intended process, and the potential utility that can be extracted from the data. This value could change with time as the data is stored for a longer period, or even if the volume or velocity are modified.

3.1.2 Big Data algorithms

Another relevant aspect apart from the above characteristics is *how the data is being used*. In this work we focus on the analysis of said data to draw meaningful conclusions that could help solve a real-life problem. For this task the most common approach is to use the data to learn a specific pattern and try to generalize this behaviour to previously unseen data. This is essentially what is called a learning algorithm in the field of machine learning, in particular in *supervised learning*. These algorithms are designed to solve a *learning problem*, which in turn is almost always categorized as a *classification problem* or a *regression problem*. In general, a learning problem can be formally stated as follows:

> *Let X be an input space and Y an output space, and suppose there is some kind of unknown relation between them, modelled by a function $f : X \to Y$. Further suppose that there is an underlying probability distribution $\mathcal{P}$ on*

> *$X \times Y$. The problem consists on estimating f using a set of n samples $\mathcal{D} = \{(x_i, f(x_i)) : i = 1, \ldots, n\}$ drawn independently and identically distributed from $X \times Y$ according to $\mathcal{P}$.*

If the output space is discrete we say it is a classification problem (the elements of Y are regarded as labels), and if it consists on a continuous range of values, we say it is a regression problem. One common example is analyzing medical data to extract information about a certain disease, or trying to predict if a patient will develop a condition based on some previously collected data from sensors, tests, etc.

Although the problem of learning from data was not born along with Big Data, it certainly has to adapt to this new setting. Often the well-known algorithms to solve tasks involving a controlled amount of data cannot be directly implemented to solve Big Data problems, and have to be redesigned with an essential feature in mind: *scalability*. This property refers to the ability of algorithms to perform the task they were designed to do in a reasonable amount of time as the volume of data increases, if we equally increase the computation power of the machine where it is running. For example, an algorithm whose time complexity grows quadratically with the number of data points is not scalable, while one with a linear time complexity may very well be.

3.1.3 Big Data ethics

Since more and more data is being generated every day, there is a growing concern that this data could be used for malicious or unethical purposes. Of particular importance is the usage of *personal data*, that is, data that could be used to identify a person or to extract information and patterns about their behaviour unbeknownst to them. In this regard there are a few principles that need to be taken into account when dealing with Big Data:

1. *Ownership*. This involves the rights over recorded data and the ability to exercise control over and limit the sharing of it. A recently passed European Union regulation called General Data Protection Regulation, or GDPR [EU16], indicates that individuals own their personal data, and have a say in how it is being treated. This regulation clearly states that *«the processing of personal data should be designed to serve mankind»*.
2. *Transaction transparency*. It refers to the right of any individual to have complete knowledge of why their data is being collected, how it is going to be used and for how long it will be stored, as well as how to amend any part of the collected data.
3. *Consent*. Before any use of personal data there must be an informed, explicit and unambiguous consent from the subject of the information.

4. *Privacy*. The idea that a reasonable effort should be made to preserve the privacy of the subjects of the data being collected is an important one. Anonymity should be guaranteed and under no circumstances should the data be used to or facilitate the task of identifying a particular person without their explicit consent and knowledge.
5. *Openness*. In an ideal setting, all data should be freely available and open to the general public. This is especially important when it comes to data gathered by governments, both for transparency and accountability reasons.

3.2 Big Data architectures

Having reviewed the main properties and characteristics of Big Data, we conclude this introduction presenting the principal architectures and methods for the treatment of this data. There are roughly three areas in which the process can be divided: storage, processing and analysis.

We have already covered the analysis step, which consists on applying suitable algorithms to try to extract meaningful conclusions about the data, remembering that these algorithms must be developed in a scalable way. Not only should they be designed for scalability, but also adapted to be run on parallel architectures that allow the scalability to be implemented. In other words, in a Big Data environment it is typical to work with distributed machines, or *clusters* of computers, so that more and more computing power can be easily added to the mix if needed.

Motivated by this approach arises the need for a method of storing data in multiple places in an efficient and safe way. Although there are multiple proposals to tackle this issue, one of the most prominent architectures is the *Hadoop Distributed File System*[1], or HDFS [CKRS10]. It provides an interface to store and access data in a distributed file system, and it is designed to be fault-tolerant and reliable.

As for the processing of the data, the reference framework is the MapReduce programming model, proposed by Google in 2004 [DG04]. It consists on a *mapping* stage, in which transformations are performed on the data, and a *reduce* stage that serves as a summarizing or aggregating operation. These tasks are handled in a parallel fashion, and all communications and data transfers are managed by the framework.

As a final comment, we should point out that a trend that has emerged to facilitate dealing with these architectures is *cloud computing*. Companies are offering their hardware resources for the public to rent and use as a way of working with Big Data, eliminating the need of physically assembling and maintaining a cluster of machines.

[1] The name Hadoop comes from a toy elephant that belongs to the son of Doug Cutting, one of the main developers of the Apache Hadoop project.

3.2.1 Apache Spark

Apache Spark[2] is an open-source, distributed processing system and cluster-computing framework used to handle Big Data workloads in a fail-safe manner. It was released under its current name in 2014, and since then it has quickly become the *de facto* standard for processing and analyzing large quantities of data. It has a large community of users who steadily make contributions and help improve and maintain the software. It also has a library specifically focused on distributed machine learning algorithms, called *MLlib*, and some APIs specific to various programming languages, such as Python or Scala.

One of the main reasons for its popularity and acceptance is that under the hood Spark implements an improved version of the MapReduce architecture in a way that is transparent to the end user. This is achieved by utilizing a data structure known as *Resilient Distributed Dataset*, or RDD for short, which is an immutable distributed collection of objects. These datasets are stored in memory in a distributed fashion across many machines and can be manipulated via a series of transformations and actions to produce the desired result. As their name implies, there are a number of safeguarding mechanisms that prevent loss of data if a problem occurs in any of the datasets containers; they are designed to be fault-tolerant. One of the preventive measures taken is to keep track of the *lineage* of each RDD, that is, the sequence of operations that were performed on it before it reached its current state, so that it can be reconstructed if needed.

With regard to the processing of the data, there is a *master node* that controls all operations and is responsible for handing out pending tasks to the *worker nodes* distributed across the cluster of machines. Sparks builds a directed acyclic graph (DAG) as a scheduling layer, which determines which tasks are executed on which nodes and in what sequence. In terms of the map-and-reduce model, the process is as follows:

1. *Map function*. The master node takes the input from the distributed file system or from memory, divides it into several sub-problems or *partitions*, and transfers them to the worker nodes. Next, each worker node processes the data and performs transformations on it, transferring the result back to the master node. In every stage, data is stored and handled as a set of ⟨key, value⟩ pairs.
2. *Reduce function*. The master node collects the answers of the worker nodes and combines them in a predefined way, acting on values that share the same key.

It is worth pointing out that Spark implements a *lazy evaluation* model consisting on *transformations* and *actions*. The so-called transformations are applied to an RDD and produce a new RDD. These transformation lazily stack together, without modifying any RDD in place, until an action is invoked on it. Then and only then the results are collected

[2]Full documentation available at https://spark.apache.org/.

back to the main node, all the queued transformations are applied and the final action is performed. In this way, transformations are related to the mapping phase, while actions are linked to the reduce phase.

This platform requires a cluster manager and a distributed storage system to function properly. Popular choices regarding these components are Hadoop YARN and HDFS, respectively. We present below a review of the other principal components of the system and its workflow (see Figure 3.1), in Spark terminology:

- *Driver*. It is a separate process to execute user applications. It is responsible for scheduling jobs and negotiating with the cluster manager via its *master node*.
- *Node or Worker*. Any machine in the cluster that can run application code.
- *Executor*. It is a process launched on a node, that runs tasks and keeps data in memory or disk storage.
- *Task*. The minimal unit of work, carried out by one executor on a single thread.
- *Job*. A parallel computation that consists of multiple tasks that get spawned in response to an action.

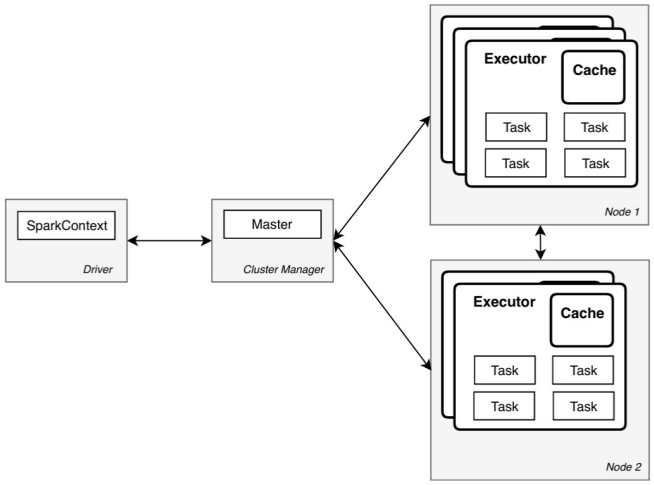


Figure 3.1: Apache Spark components and schematic workflow. Taken from [Gon19].

4 Design and implementation of fuzzy systems for Big Data

In this chapter we will concentrate on the design and implementation of several fuzzy systems for working with Big Data. While approaching these systems from a theoretical perspective as we have been doing throughout the text, we will shift our focus somewhat and also describe a number of implementations in an actual programming language. This part can be regarded as a more practical aspect of our work, a small software development project undertaken with the intent of fusing the ideas previously presented.

The language of choice is the Scala programming language, and the platform to develop our Big Data programs will be no other than Apache Spark. Although we will generally provide a high-level description of the algorithms, we will also insert occasional pieces of code that help clarify what we are trying to achieve, and even sometimes discuss technical aspects of the implementation or particularities of the language.

The interested reader can refer to [Sue12] or [OSV08] for a detailed introduction to Scala. Basic and intermediate notions of the interaction between Scala and Spark are also covered in [KKWZ15].

4.1 Learning algorithms for fuzzy systems

There are basically two different strategies to build FRBS, depending on the information available at the moment of its construction. The first way is to rely on human experts, inputting the knowledge manually into the system. This is one of the main tasks tackled in the field of *knowledge engineering*, where knowledge engineers devise the best methods to extract and properly represent the knowledge from experts in the matter. The other way is to try to learn the rules and behaviour of the system from the data, by using several learning methods. We will concentrate on the latter approach.

Generally speaking, learning algorithms for FRBSs can be decomposed in two stages: a structure identification step and a parameter tuning phase. In the first one, the structure and size of the rule base is estimated, while in the second one the parameters corresponding to the various membership functions involved are optimized. These steps can be performed sequentially or in a simultaneous way.

While these algorithms are tailored to work with fuzzy systems, the restrictions and particularities of the general learning algorithms still apply. That is, choosing a specific algorithm is only one step of the full learning process: data preprocessing, dimensionality reduction, parameter selection, etc. A number of software libraries have been developed to work with fuzzy systems, among which we would like to highlight the `frbs` R package [RBHB15]. It gathers a multitude of classification and regression algorithms under a common API that focuses on easiness of use.

Before we specify the algorithms implemented, we present an overview of the three main types of learning algorithms for FRBSs.

4.1.1 Space partition algorithms

These algorithms are designed so that a partition on the input and output spaces is established, and rules are derived from the grade in which each data point represents each of these partitions. In other words, rules are created *ad-hoc* directly from the data.

The parameters of the membership functions are dependent on the specific technique used to partition the spaces, and the expressions used to measure the affinity of each point to each partition. A notable special case is the use of *clustering algorithms* to make the partitions. Generally speaking, the purpose of clustering is to group together data points that share certain features, with the hope of detecting similarities and obtaining a more concise representation of a system's behaviour, which we can later use to refine it. In this case, each centroid acts as a fuzzy rule, with a membership function that measures the grade in which each points belongs to each centroid. This is known as *fuzzy clustering*, and differs from classical clustering algorithms in that each point generally belongs to every cluster to a greater or lesser extent, not just to one of them.

4.1.2 Neural network algorithms

As their name indicates, these algorithms are based on the concept of artificial neural networks. These structures have been adapted from the field of machine learning and more precisely of deep learning, where they are widely used and serve as the basis of multiple state of the art algorithms (see for example [LBH15]).

The FRBS is usually generated beforehand, and it is then adapted to fit a neural network structure. This network is used as a way of tuning the parameters of the membership function using the backpropagation algorithm, and thus retaining all the advantages of this technique. A typical neuro-fuzzy system has a layer that represents the membership functions of the linguistic values associated with the input variables, an antecedent layer that represents the antecedent parts of every rule, a consequent layer to express the consequent of the rules, and a defuzzification layer to provide a final response. Commonly there are weights involved in the process that calibrate the importance of each rule, and they can

also be adjusted when training the network. A good example of this type of algorithm is the HyFIS algorithm [KK99], which uses a 5-layer neural network structure and a hybrid learning scheme.

4.1.3 Genetic algorithms

We arrive at the last type of FRBS considered here, which is based on genetic algorithms. Ample research has been conducted on this topic, and the reader can refer to [Cor11] for a historical review or to [CHHM01] for a whole book on the subject.

The philosophy behind these algorithms is similar to that of the neuro-fuzzy systems: an initial structure is established, and then a genetic method is adapted to tune a specified set of parameters. But genetic algorithms can also be used to generate the whole structure from the start, as shown in [CHV01]. As an alternative to neuro-fuzzy systems, these algorithms present all the advantages (as well as the disadvantages) of the standard genetic methods. We mention a couple of them that are studied in the literature.

1. The first one is a basic genetic fuzzy system based on Thrift's method [Thr91] of integer coding. In this method, which follows an elitist scheme, each possible rule represents a chromosome, and a new population is obtained by applying the usual crossover and mutation operators (although mutations are adapted to this particular setting). The fitness of an individual is calculated as the mean squared error between predictions and actual values, as output by the system.
2. Cordón and Herrera proposed the D-MOGUL method [CH97] to automatically generate a complete knowledge base; it uses a genetic algorithm to determine the structure of the fuzzy rules and the parameters of the membership functions simultaneously. It consists on three steps:
    - An iterative *rule generation* process that explores the search space for rules that satisfy certain desirable conditions. A partition of the input space consisting on triangular membership functions is considered.
    - A *genetic simplification phase*, that eliminates redundant rules from the previous step by exploring the subsets of rules that best cooperate among themselves.
    - A *genetic tuning phase* in which membership parameters are adjusted to try to maximize the accuracy of the final model.

4.2 Fuzzy learning algorithms implemented

Before proceeding with the implementation tasks, and even before starting the theoretical study that we have already presented in the previous chapters, we made an estimation of the total cost of this project and the temporal planning involved, which can be consulted in Appendix A.

We begin with an implementation of a basic state partition algorithm, arguably the simplest one of its family, but one that will provide a starting point from which to build upon. We continue our implementation task with two algorithms that belong to the family of fuzzy clustering methods. As we will see shortly they are closely related, and the fact that neither of them was available in the MLlib library at the time of writing this thesis was enough motivation to implement both of them.

We tried to follow MLlib's code style and conventions as much as possible, though at some points there might be some deviations from said conventions, and a few adjustments would probably be needed in order to integrate these algorithms in the library. Nevertheless, the philosophy of building a model, fitting it to some data and then using it to make predictions is still there.

4.2.1 Wang & Mendel algorithm

The first algorithm developed is the Wang & Mendel algorithm for learning fuzzy rules from examples [WM92], which as explained before is a basic space partition algorithm for fuzzy systems. In general terms, it divides the input and output spaces to make a grid, and uses these divisions to infer the rules of the system checking how well the data fits in each portion of the grid. Supposing that we have a collection of n input-output pairs of the form $(x_1^{(1)}, x_2^{(2)}; y^{(1)}), \ldots, (x_1^{(n)}, x_2^{(n)}; y^{(n)})$, we summarize below the principal steps of this algorithm. The extension to the multi-input multi-output case is straightforward.

In *Step 1*, we assume that the domain intervals of the variables are $[x_1^-, x_1^+]$, $[x_2^-, x_2^+]$ and $[y^-, y^+]$, which means that with high probability the data on each dimension will lie on those intervals. We then divide each interval in $2N + 1$ regions (N can be different for each variable), and assign a membership function to each region. In our implementation we have chosen triangular membership functions, but other choices such as trapezoidal and Gaussian are possible as well. Each region is may be given a name that represents our understanding of the intervals (for example, we could name them $S1, \ldots, SN, C, B1, \ldots, BN$, where '$S$' stands for small, '$C$' for center and '$B$' for big). A sample partition of the input and output spaces can be seen in Figure 4.1.

In *Step 2*, we proceed to the generation of fuzzy rules based on the data points. For the ith data point, we compute the membership of $x_1^{(i)}, x_2^{(i)}$ and $y^{(i)}$ to each of the regions in their respective dimension, and in each case we select the region with the highest membership. Then we form a rule in which the antecedents are the regions found in the input space, and the consequent is the region selected in the output space. For example, if the regions found were $S1$, CE and $S1$, then the rule would look like

$$\text{if } x_1 \text{ is } S1 \text{ and } x_2 \text{ is } CE \text{ then } y \text{ is } S1. \tag{4.1}$$

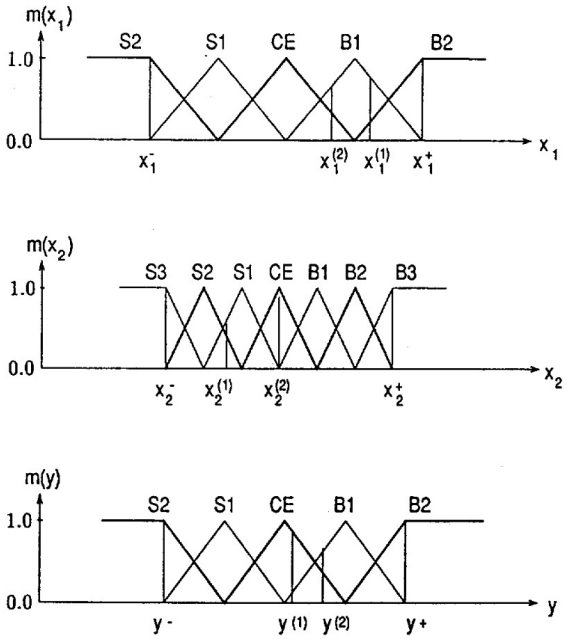


Figure 4.1: Sample division of input output spaces in fuzzy regions. Taken from [WM92].

In *Step 3* we assign a degree of activation to each rule, which is simply the product of the values of the membership functions of the data point in question to each of the selected regions in each dimension. For example, if we take the rule shown in (4.1) its degree would be $D = m_{S1}(x_1)m_{CE}(x_2)m_{S1}(y)$.

In *Step 4* we create a combined rule base, aggregating all the rules previously found. To prevent duplicates and conflicts, we only select among the rules with the same antecedent those which have the highest degree.

Finally, *Step 5* is regarded as part of the prediction phase. For a given input (x_1, x_2) we combine the antecedents of the ith fuzzy rule to determine the degree to which it is satisfied by (x_1, x_2), and we denote it by

$$m^i_{O^i} = m_{I^i_1}(x_1)m_{I^i_2}(x_2),$$

where O^i is the output region of the ith rule and I^i_1, I^i_2 represent its input regions. Then we use the centroid defuzzification formula to obtain an output

$$y = \frac{\sum_{i=1}^{K} m^i_{O^i}\bar{y}^i}{\sum_{i=1}^{K} m^i_{O^i}}, \tag{4.2}$$

where K is the total number of rules and $\bar{y}^i$ denotes the center of the fuzzy region O^i, which is defined as the smallest absolute value for which the membership function is unity. We see that this step is the only one that would need to be significantly modified to adapt the algorithm to multiple outputs. We would only need to change O^i to O^i_j in (4.2), where O^i_j is the output region of rule i for the jth component (note that $m^i_{O^i_j}$ is the same for all j), and also replace $\bar{y}^i$ with $\bar{y}^i_j$ (the center of region O^i_j).

In this case the scalability is exploited when building the rule base (steps 2 and 3), since each point can be processed in parallel, and then all the rules are aggregated as they undergo the conflict elimination process. The general scheme for this design is described in Listing 4.1. We can aim for a truly scalable implementation, since it can be checked that the time efficiency of this algorithm is $O(ndN)$, where n is the number of data points of dimension d, and N is the maximum number of partitions across all dimensions.

Listing 4.1 Rule base distributed calculation in Wang & Mendel algorithm.

```
val ruleBase = data.mapPartitions { iterator =>
    for {
        (x, y) <- iterator
    } yield {
        // Compute regions with maximum membership in each dimension
        val (ri, degreeInput) = maxRegion(x, regionsInput, limitsInput)
        val (ro, degreeOutput) = maxRegion(y, regionsOutput, limitsOutput)

        // Compute rule degree
        var degree = degreeInput * degreeOutput

        (ri, (ro, degree))
    }
}.reduceByKey { case (r1, r2) =>
    if (r1._2 > r2._2) r1 else r2
}.map { case (ri, (ro, _)) => (ri, ro) }
```

An extension to this system for solving a classification problem is also immediate: the output variable would simply be the label of the input point, and no division of the output space would be needed. The degree of each rule could be computed using only the antecedents, and the defuzzification strategy would need to be changed to one that took into account that the output space was comprised of labels. This extension is known as Chi's algorithm [CYP96, p. 141], and a detailed account of the adaptation process can be found in [ÁM18].

4.2.2 Subtractive clustering

The second algorithm that we tackle is the subtractive clustering method for cluster estimation, proposed by Stephen Chiu in 1994 [Chi94]. The aim of this algorithm is to provide an estimate number of clusters and an initial guess for the centroids that represent them, obtaining this information from the numerical data available. It can serve as a preliminary step in other purely clustering-based algorithms, in which estimating the number of centroids or their initial value can be a difficult task, and usually a poor choice in this department leads to equally poor performance of the algorithm.

This method builds upon Yager and Filev's iterative mountain method [YF94], which basically proposed making a grid in the data space and assigning a value of *potential* to each point on the grid based on their distance to the actual data points, such that a grid point with many nearby points would have a high potential value. In the first iteration the grid point with the highest potential is chosen as the first cluster center, and then the idea is to reduce the potential of each grid point proportionally to its distance to the chosen cluster center. Once all the potentials are updated, a second cluster center is chosen in the same way in the next iteration, and the process is continued until the highest potential value falls below a threshold. This method is simple and effective, but its main drawback is that the computation of the grid potential grows exponentially with the dimension of the problem (a grid is needed in each dimension). This is a very restricting issue, especially in a Big Data context where we can expect to have very high dimensionality. The novel approach introduced by Chiu is to consider the data points themselves as potential cluster centers instead of making a grid. In this way the number of "grid points" to be evaluated is simply equal to the number of input data points, and this is independent of the dimension of the problem.

Consider a collection of n input data points $\{x_1, \ldots, x_n\}$ in an M-dimensional space. Without loss of generality we suppose that each data point is bounded by a hypercube, and define a measure of potential at each point by

$$P_i = \sum_{j=1}^{n} e^{-\alpha \|x_i - x_j\|^2}, \tag{4.3}$$

where

$$\alpha = \frac{4}{r_a^2}, \quad r_a > 0.$$

We note that the potential of each point is a function inversely proportional to its distance to every other data point. The value r_a is a key parameter in this computation, and it measures the effective neighbourhood radius in which data points have a significant effect in the potential of a specific point. We denote by x_k^* the kth cluster center chosen, and by P_k^* its

potential. Following the algorithm outlined above, after the kth cluster center is selected, we revise the potential of every point according to the formula

$$P_i \leftarrow P_i - P_k^* e^{-\beta \|x_i - x_k^*\|^2},$$

where

$$\beta = \frac{4}{r_b^2},$$

and $r_b > 0$ is the radius of the neighbourhood in which the cluster center significantly influences the change in potential of the remaining points. Again, the potential is decreased by a quantity that depends on the distance of the point in question to the newly found cluster center. This process is iterated until a stopping condition is reached, depending on two thresholding parameters $\underline{\varepsilon}$ and $\bar{\varepsilon}$, which we describe in Algorithm 4.1.

Algorithm 4.1 Stopping condition for Chiu's algorithm.

if $P_k^* > \bar{\varepsilon} P_1^*$ **then**
  Accept x_k^* as a cluster center and continue
else if $P_k^* < \underline{\varepsilon} P_1^*$ **then**
  Reject x_k^* and end the clustering process
else
  $d_{\min} \leftarrow$ shortest distance between x_k^* and all previously found cluster centers
  if $\frac{d_{\min}}{r_a} + \frac{P_k^*}{P_1^*} \geq 1$ **then**
    Accept x_k^* as a cluster center and continue
  else
    $P_k^* \leftarrow 0$
    Reject x_k^* and select the next data point with the highest potential as the new x_k^* to re-test
  end if
end if

Apart from the upper and lower thresholds, we have a gray zone in which we check if the data point provides a good trade-off between having a sufficient potential and being sufficiently far from other cluster centers. The usual choice of parameters without additional information of the problem is $r_b = 1.5 r_a$, $\underline{\varepsilon} = 0.15$ and $\bar{\varepsilon} = 0.5$. However, we will experiment with these parameters and choose the values that we feel are best suited for the problem at hand.

In order to implement this algorithm in Spark, we need to carefully replicate the iterative algorithm in an appropriate manner, taking into account that the data structure holding the potential of each data point will be a RDD indexed by the point itself, represented as a tuple of coordinates, and the reduction of potential will be an instance of a map-reduce

operation. The key step of the algorithm is the initialization step, in which we need to pair every point in the data set with *all* the other points to get the pairwise distance matrix. To implement this initialization, we first produce every possible pair of data points, and then we invoke a map-reduce operation to compute the actual potential.

Another thing to take into account, and arguably the most important aspect of our contribution to this algorithm, is the approach we take in implementing the parallelism. The objective was not to merely adapt the algorithm to a parallel setting, but to design modifications that would increase the scalability. In the end we derived *three distinct versions* of this algorithm for use with Big Data, each one more polished than the last. We provide below a brief summary of each alternative and some insight as to how we moved from one to the next.

In the first place we implemented a *global version* of the algorithm. We started with the most naive strategy, which was almost a literal translation of the sequential algorithm, but distributing the operation involving computations of potentials among the nodes in the cluster. We used the transformation `cartesian` provided by the Spark API to compute every pair of points for the initialization. The time efficiency of this operation is inevitably $O(n^2)$, so we did not really hold any hope that this implementation would be scalable (as of course the experimentation would later confirm). However, we insisted on programming this version to have a baseline case to which we could compare the rest of the implementations. If for some reason another version performed anywhere near this one efficiency-wise, we would know that something in the design was wrong. The concrete implementation of the initialization phase is shown in Listing 4.2.

Listing 4.2 Global version of the initialization phase of Chiu's algorithm in Spark.

```
val pairs = data.cartesian(data)
val potential = pairs.map { case (x, y) =>
    (x, exp(-alpha * Vectors.sqdist(x, y)))
}.reduceByKey ( _ + _ )
```

In the second place, we implemented a *local version* of the algorithm. The next thing we thought about was how we could tackle the bottle neck in the initialization phase, because once that was done, the rest of the algorithm has a linear time efficiency and could potentially be refined with scalability in mind. The goal was clear: to reduce the number of interactions between points when computing the initial potential. We thought about a method in which each point would only be paired with some "nearby" points in order to calculate the potential. This notion of closeness was implemented not as a function of the distance in the M-dimensional space (which is precisely the thing we are trying to avoid to compute) but as a function of how the points were internally split up in partitions by Spark. Thus, we devised a technique in which the algorithm is only applied locally: it is executed on each partition only with the points that belong to it, and in the end all the responses are

aggregated so that the final result is the concatenation of all the cluster centers found on each partition. In this way the overhead of initializing the potential with a huge quantity of data is eliminated, and the effectiveness of the algorithm is hopefully retained because the (random) partitions preserve some measure of representativity of the whole data set. In this case we needed to change the implementation of the initialization phase, which is reflected on Listing 4.3.

Listing 4.3 Local version of the initialization phase of Chiu's algorithm in Spark.

```
val pairs = data.flatMap ( x => data.map ( y => (x, y) ) )
val potential = pairs.groupBy ( _._1 ).map { case (x, xs) =>
  val pot = xs.map ( y => exp(-alpha * Vectors.sqdist(x, y._2)) ).sum
  (x, pot)
}
```

However, with this implementation it might be the case that in complex and high-dimensional spaces some nodes lose a significant amount of information when looking only at a subset of the data, and this could result in cluster centers that are not sufficiently spaced. This was observed to be the case when trying this implementation on some data sets, so this version was not satisfactory either. Nevertheless, for a data set consisting on about 5 million data points in a 8-dimensional space, this algorithm only took a couple of minutes to finish, while the global version took *more than a week*.

Finally, we developed what we called an *intermediate version*. We wanted to try to resolve the limitation pointed out in the local version, and somehow include some bits of global information into the local scope of the algorithm. The approach taken involved a somewhat hybrid version based on the two previous ones.

First we apply the algorithm locally, but next instead of concatenating the results of all partitions, we form groups of a predefined size and combine the centers of the selected partitions only. Then we apply the global version of the algorithm on each group, taking the cluster centers as the data points, and their current potential as the initial potential. In this way, after executing this second step the centers obtained would be more spaced (because the algorithm internally discards points which are close to the selected cluster centers), and finally the results obtained from applying the algorithm on each group are concatenated. However, since the points come front different partitions, their potential is not directly comparable, so we need to normalize it first. For this we propose to divide the potential of each point by the number of points of its original partition. In this version there are a number of parameters involved, and we will need to find a good balance between the number of cluster centers found and the number of partitions allowed, since more partitions means more parallelism and thus less computation time, but an excessive number of clusters can diminish the quality of the result.

Regardless of the version chosen to implement this algorithm, we get a set of cluster centers as a result, which will be denoted by $\{x_1^*, \ldots, x_c^*\}$. Now we want to build a FRBS based on these cluster centers, thinking of each of them as a fuzzy rule. We implemented two variants of this construction, depending on whether we want to solve a classification problem or a regression problem. In both instances we get a Mamdani FIS.

In the first case, we suppose that the points used to train the algorithm are labeled with their real class. First we perform the unsupervised clustering analysis (without looking at the labels), and then we assign to each cluster the label of its center. Then, for the prediction phase we simply assign a previously unseen point to the cluster for which it has the greater grade of membership. We were considering each point as a fuzzy rule, so given an input x and taking (4.3) into account, we define the degree to which rule i is fulfilled as

$$\mu_i = e^{-\alpha \|x - x_i^*\|}.$$

The label of this point will obviously be the label of the selected cluster. In the intermediate version described earlier we let that the value r_a for the membership function above be chosen by the user, since it could have been different in the applications of the local and global versions.

In the second case, we split the data points x into input and output space, denoting the corresponding component vectors y and z, respectively. We proceed as in the previous case, and measure the degree to which each rule (represented by each cluster center) is fulfilled with the membership function

$$\mu_i = e^{-\alpha \|y - y_i^*\|}.$$

In this case a defuzzification is needed to predict the output for an input point $x = (y, z)$, for which we employ the COA method,

$$z = \frac{\sum_{i=1}^{c} \mu_i z_i^*}{\sum_{i=1}^{c} \mu_i}.$$

4.2.3 Fuzzy C-Means

To complete the study of fuzzy clustering algorithms we explore the implementation of one of the most well-known algorithms in this regard: the Fuzzy C-Means (FCM) algorithm. This method is the fuzzy analog of the standard K-Means clustering algorithm, and their operation is essentially the same, save for the fact that the fuzzy version allows every point to belong to any number of clusters to some extent.

To perform the clustering of the data we need to provide beforehand the number of clusters and their initial cluster centers. Suppose we have a collection of d-dimensional input data $\{x_1, \ldots, x_n\}$ and we want to divide it into c groups. Each data point will then have a membership function that measures the degree to which it belongs to every cluster center, that is, each x_i has an associated set

$$\{u_{ij} \in [0,1] : j = 1, \ldots, c\},$$

which needs to satisfy the conditions

$$u_{ij} \geq 0, \quad \text{for all } i, j, \quad \text{and} \quad \sum_{j=1}^{c} u_{ij} = 1, \quad i = 1, \ldots, n. \tag{4.4}$$

If we denote the membership matrix by $U = (u_{ij})$ and the set of cluster centers by $V = \{v_1, \ldots, v_c\}$, the strategy of the algorithm is to minimize the function

$$J_m(U, V) = \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^m \|x_i - v_j\|^2, \tag{4.5}$$

where $m > 1$ is the *fuzzification parameter*, which controls how fuzzy the final clustering will be (the value $m = 1$ would imply a crisp clustering; typically $m = 2$ is chosen). We can apply the method of Lagrange multipliers to the constrained optimization problem of minimizing (4.5) subject to the conditions (4.4), obtaining the iterative update rules given by

$$u_{ij} = \frac{1}{\displaystyle\sum_{k=1}^{c} \left(\frac{\|x_i - v_j\|}{\|x_i - v_k\|} \right)^{\frac{2}{m-1}}} \tag{4.6}$$

and

$$v_j = \frac{\displaystyle\sum_{i=1}^{n} u_{ij}^m x_i}{\displaystyle\sum_{i=1}^{n} u_{ij}^m}. \tag{4.7}$$

As for the stopping condition, we take a double approach. We will end the iterative process when the cluster centers do not change within a fixed tolerance level from one iteration to the next, or when we reach a maximum number of iterations L. Having derived these expressions, the literal FCM algorithm is fairly straightforward to implement and can be seen schematically in Algorithm 4.2.

Not taking into account the initialization of the cluster centers, this algorithm runs in time $O(nc^2d)$. While there are a number of proposals to improve the literal FCM algorithm and increase its speed via probabilistic or stochastic methods (see for example [Pha01]),

Algorithm 4.2 Literal Fuzzy C-Means algorithm.

Input: $X, V, c, m, L, \varepsilon$
Output: U, V'

1: $l \leftarrow 0$
2: Compute membership matrix U according to (4.6)
3: Compute the new cluster centers V' as in (4.7)
4: $l \leftarrow l + 1$
5: **if** $\|V - V'\| < \varepsilon$ **or** $l \geq L$ **then** ▷ Here $\|\cdot\|$ is some fixed matrix norm
6: **return** U, V'
7: **else**
8: $V \leftarrow V'$
9: **go to** 2
10: **end if**

we will focus on developing a version of the algorithm that is scalable and adheres to the map-reduce model, but without altering its core. In other words, we are looking for a variant that produces the same final result as Algorithm 4.2, but that can be reliably executed with arbitrary large amounts of data.

To achieve our goal we will need to identify the steps of the algorithm that can be parallelized in a map-reduce fashion. Since the calculation of the membership degree of each data point is independent of the rest, we focus on computing this value in a distributed way. On top of this, we note that the actual membership values are not needed to recalculate the cluster centers, but only the values of the sums involved in (4.7). For this reason, we save quite a lot of space and computational time if we do not keep the entire matrix in memory between iterations. This is precisely what we will do, by means of a custom map operation as described in Algorithm 4.3.

Algorithm 4.3 Map stage of the distributed Fuzzy C-Means algorithm.

Input: x_i, V, m
Output: $\langle j, \langle u_{ij}^m x_i, u_{ij}^m \rangle \rangle$

1: **for each** v_j **in** V **do**
2: **yield** $\langle j, \langle u_{ij}^m x_i, u_{ij}^m \rangle \rangle$ ▷ The values u_{ij} are computed according to (4.6)
3: **end for**

The reduce phase will just be the pairwise addition of the values returned by the map operation, denoted by sx_j and s_j (notice that we are mixing scalar and vector addition, so a custom reduce function is needed in the actual implementation in Spark). A high-level description of the final design can be seen in Algorithm 4.4.

Algorithm 4.4 Distributed Fuzzy C-Means algorithm.

Input: $V_0, X, c, m, L, \varepsilon$
Output: V'

1: $V \leftarrow V_0$ ▷ Cluster initialization step
2: $l \leftarrow 0$
3: $I \leftarrow X.\text{mapFCM}(V, m).\text{reduceByKey}(+)$ ▷ Compute membership information
4: **for** $\langle j, \langle sx_j, s_j \rangle \rangle$ **in** I **do** ▷ Compute the new cluster centers V'
5: $\quad v'_j \leftarrow \dfrac{sx_j}{s_j}$
6: **end for**
7: $l \leftarrow l + 1$
8: **if** $\|V - V'\| < \varepsilon$ **or** $l \geq L$ **then** ▷ Here $\|\cdot\|$ is some fixed matrix norm
9: $\quad$ **return** V'
10: **else**
11: $\quad V \leftarrow V'$
12: $\quad$ **go to** 3
13: **end if**

For the prediction stage, we consider directly a classification problem, and approach it in the same way as we did in the subtractive clustering algorithm. However, in this case the cluster centers are not generally input points, and thus we do not know their labels beforehand. To solve this problem, we assign to each cluster the predominant label among the points that belong to it with a degree of at least α (that is, we perform an α-cut of the fuzzy set represented by the cluster, and then we select the predominant label). The membership function used for this is of course (4.6). In addition to this, note that it might be the case that on some clusters the maximum membership function attached at any point is far from 1 (possible, but not very likely, since if a cluster is selected it should have at least some nearby points). To account for this incongruence when performing the α-cut, we first scale every membership function by its maximum value.

It is worth pointing out some implementation details that have been added to try to increase the performance of the algorithm. In the first place, the custom map function has actually been implemented as a `mapPartitions` call in Spark. This transformation is the same as a `map`, but performs the operation in question on every element of the same partition, reducing the number of calls involved and improving the overall speed. Also, in each iteration we *broadcast* the cluster centers to every partition, so that they have a copy stored locally. In this way we reduce the number of interactions between the master node and the worker nodes, which is always a desirable improvement. In general, we have tried to use a more functional style to match the rest of the Spark API and philosophy. This

is why some operations such as the computation of the centers and the checking of the stopping conditions are written in a map-reduce syntax, even though they are not actually distributed among nodes.

We finish with a comment about the initialization phase. In the algorithm implemented we provided three options to initialize the number of clusters and the initial centers. They can be either user-specified, chosen at random or generated by Chiu's algorithm described in the previous section. There are a number of initialization algorithms proposed in the literature, many of them based on sampling the input space and performing a rough clustering to approximate the clusters. Though we will not implement any of these algorithms, we mention a specific one that is widely used in practical applications: the kmeans++ initialization algorithm[3] proposed by D. Arthur and S. Vassilvitskii in 2006 [AV06], and its parallel version called kmeans||, proposed by Bahmani *et al.* in 2012 [BMV+12]. They both consider data points as potential cluster centers, and they are based on the intuition that generally it is a good thing to have the initial cluster centers as spread out as possible. To achieve this, an initial center is chosen at random from the input data set, and in subsequent iterations more cluster centers are chosen from the remaining points with probability proportional to their distance to the nearest existing cluster center.

[3]The value of k representing the number of cluster centers still needs to be provided.

5 Comparative study

In order to test our implementations, we will conduct a study to see how well our algorithms perform in a couple of real data sets. These data sets will of course present characteristics of Big Data, mainly the data volume. Before we begin, we would like to make some considerations.

1. The main aspect of the algorithms we want to test, apart from their correctness, is their *scalability*. This is not a study designed to discern whether FRBS are better or worse than other classification or regression algorithms, though we will make comparisons with some of them for reference. That is to say, we do not intend to discuss the effectiveness of fuzzy systems, which has already been extensively established. What we do want to show is how the fuzzy algorithms we developed can function in an environment in which an arbitrary large amount of data is involved.
2. Continuing along the lines of the previous point, we do not aim to get the best possible result with our algorithm either. That is why we will not be applying fine tuning techniques such as parameter grid search or cross-validation. We will sometimes experiment with different configurations of parameters, but not in a systematic way.
3. The data sets chosen for this study have a considerable number of instances, and present a challenge in the sense that a typical personal computer would struggle to process all the data into the algorithms. However, the limitations in the infrastructure available to test our implementations and the interest in making a sizeable number of executions with different parameters result in the choice of data sets with controlled size so that the execution time stays within reasonable margins.

5.1 Hardware and infrastructure

We have deployed our programs on a dedicated server tailored for Big Data operations, called `ulises.ugr.es`. This is a cluster of 20 machines interconnected and ready to execute Spark programs. The main characteristics of this architecture are as follows:

- There are 20 nodes in total, each carrying 2 processors, 6 cores per socket and 2 threads per core.
- Each individual CPU is an Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz.
- Each node has 64GB of memory.

- The Spark installation is currently at version 2.3.2. It uses YARN as the cluster manager and HDFS as the underlying distributed file system.
- The Scala version installed on this server is 2.11.8.
- The operating system is Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-74-generic x86_64).

5.2 Data sets

We have selected five data sets in total for this experimentation. The first three of them are fit for classification problems (the data is labeled), and the last two are considered for regression purposes. We provide a very brief description of them below[4].

1. *HAR data set.* This is a relatively small data set consisting on 165632 instances of data gathered by sensors on the subject of human activity recognition. It possesses 18 numerical attributes (so we are working in $\mathbb{R}^{18}$) and 5 different class labels. For testing purposes, we make a 70%-30% train-test stratified split, resulting in the class distribution shown in Table 5.1.

Table 5.1: Class distribution for the HAR data set.

Class	No. of instances
0	50631
1	11827
2	47370
3	12414
4	43390

2. *POKER data set.* This data set is almost 10 times as big as the previous. It has 1025010 instances with 10 attributes and 10 classes, representing poker hands. The division in this case is 80%-20% for train and test, and the class distribution is imbalanced, but representative of the distribution we want to learn. The class distribution after the split can be seen in Table 5.2.
3. *SUSY data set.* This is a relatively large data set based on physics experiments, and it is a well known reference for academic studies in Big Data (see [BSW14]). It has 5 million instances, 18 numerical attributes and 2 classes. We consider the first 4.5 million points for training and the rest for testing. The class distribution is perfectly balanced in both cases.

[4] All data sets used are publicly available at https://archive.ics.uci.edu/ml/datasets.php.

Table 5.2: Class distribution for the POKER data set.

Class	No. of training instances	No. of testing instances
0	410960	102741
1	346478	86619
2	39062	9766
3	17308	4326
4	3182	796
5	1640	410
6	1168	292
7	189	47
8	13	4
9	7	1

4. *Query Analytics Workloads data set (QAW)*. This is a regression synthetic data set, consisting of 199843 instances, 6 numerical attributes and one numerical variable to predict, with a 80-20 train-test split.
5. *Gas Sensor Array data set (GSA)*. The last regression data set consists on a considerable higher number of instances, comparable to that of the SUSY data set. It consists of 4177504 instances with 16 numerical attributes and a numerical variable to predict. The data was gathered by a number of sensors measuring the concentration of some chemical compounds in a gas mixture under specific conditions. As usual, we consider a 80-20 train-test split.

5.3 Experimental results

We proceed now with the results of the various scenarios considered. There are a couple of aspects that remain constant across all the executions, so we will briefly address them here. In the first place, we use the transformation `StandardScaler` to standardize the data points on each dimension and make them comparable, with mean 0 and standard deviation 1. Next, we fix the parameters in a call to Spark in order to use all the resources available, setting:

- `num-executors`: 20.
- `executor-cores`: 19.
- `executor-memory`: 32g.

We also set the number of Spark partitions to 380 (1 partition per core) for every data set except SUSY and GSA, for which we consider 760 partitions (2 partitions per core). We tried to find a balance between the number of partitions and the size of the data set, since using many partitions when the number of points is not big enough can lead to some unwanted situations. Moreover, we limit the number of iterations to 100 in iterative algorithms, and set the tolerance for stopping conditions at $\varepsilon = 10^{-4}$.

For the reasons explained in Chapter 4 we will only consider the implementation of the intermediate version of Chiu's algorithm, denoted ChiuI, for which we set the number of groups to 10. In addition, when coupled with the FCM algorithm, we will always compare the results with the random initialization version using *the number of centroids found with the subtractive clustering algorithm*.

The metric used to measure the quality of the algorithms will be the *accuracy* in the case of classification problems (that is, the relative amount of testing instances correctly classified by the algorithm) and the *mean squared error* (MSE) in regression problems. In each case we will show the value of the metric on the test set and the execution time in minutes, denoted by T.

5.3.1 Classification algorithms

For comparison we have included some some well-known classification algorithms from the MLlib: a Random Forest classifier with 200 trees and Logistic Regression classifier with L2 regularization. We have also included the clustering algorithm K-Means, to which we have applied the same modification to turn it into a classification algorithm that we did with the Fuzzy C-Means algorithm. We present the results obtained on each of the first three data sets, showing the number of iterations, denoted by #, and the cluster centers (if applicable). In addition, we will also show the value of the loss function being optimized in FCM. Firstly, the results of the HAR data set are available on Tables 5.3, 5.4 and 5.5.

Table 5.3: Experimental results of comparison algorithms for the HAR data set.

Algorithm	Centroids	Acc	T	#
Random Forest	–	**97.340**	0.82	–
Logistic Regression	–	83.411	0.53	–
KMeans	150	93.001	1.08	59
	255	94.568	1.48	52
ChiuI	255	70.892	1.03	–

We observe that, as expected, the Random Forest classifier achieves a high accuracy and is the clear winner of this comparison. What is somewhat surprising is that the adapted KMeans algorithm beats the Logistic Regression model, which is specifically designed to perform well in multi-label classification problems. Both algorithms took roughly the same time to finish, and the KMeans model ended up 10 points ahead, which is quite a margin. Finally, the subtractive clustering algorithm performs worse than the rest, which was to be expected, since in general fuzzy algorithms trade a bit of accuracy for a better interpretability of the model. What is important to note here is that the fuzzy ChiuI algorithm appears to be scalable: it takes up about as much time as any of the other algorithms tried. We chose the value $r_a = 1$ after some experimentation, and left the rest of the parameters with their default values.

Directing our attention to Table 5.4 we see that a brief parameter search was carried out for the FCM algorithm, albeit in a manual way. In this case we wanted to test the random initialization variant, and we tried varying the fuzzification parameter m and the level sets cutoff α, apart from the number of centroids. The results can be summarized as follows:

- The final accuracy of the model seems to be inversely proportional to the value of α. It would appear that selecting a lower value for the cutoff has a positive impact in the performance of the algorithm, maybe because the points degree of membership is divided more or less equally among the clusters, and they do not present a high membership to any one of them.
- The value of m exhibits the same behaviour as the value of α: the lower it is, the better performance we achieve. By reducing the level of fuzziness we are able to get more accurate results, which hints at the fact that more often than not a crisp algorithm is more effective, at the expense of losing the interpretability that comes with fuzzy rules. Also, we observe that the loss function decreases as m increases, which is only natural looking at the expression being optimized.
- The execution time of FCM related to KMeans is higher, which was to be expected since this last algorithm runs in time $O(ncd)$ instead of $O(nc^2d)$. However, the difference is not unreasonable, and the FCM algorithm finishes in an acceptable amount of time; on average it takes 3 seconds per iteration with 150 centroids, and 7 seconds per iteration with 255 centroids.
- The accuracy obtained is in general lower than that of the comparison algorithms (except for the subtractive clustering algorithm), though with some combinations of parameters we reach the numbers of the Logistic Regression model. We believe that a good fine-tuning of the parameters would increase the performance even more.

Lastly, Table 5.5 shows the same experimentation as before but utilizing the subtractive clustering algorithm in the initialization step. The conclusions are essentially the same as in the random version, except for the fact that the overall performance drops a little. This can

Table 5.4: Experimental results of the FCM algorithm with random initialization for the HAR data set.

Algorithm			Centroids	Acc	Loss	T	#
FCM + Random	$\alpha = 0.2$	$m = 1.75$	150	81.350	11922.266	5.16	100
		$m = 2.00$		73.401	4106.746	5.14	
		$m = 2.25$		73.131	1331.418	5.30	
	$\alpha = 0.4$	$m = 1.75$	150	80.557	11922.266	5.27	100
		$m = 2.00$		71.847	4106.746	5.27	
		$m = 2.25$		71.795	1331.418	5.47	
	$\alpha = 0.6$	$m = 1.75$	150	80.197	11922.266	5.31	100
		$m = 2.00$		71.847	4106.746	5.29	
		$m = 2.25$		71.795	1331.418	5.44	
	$\alpha = 0.8$	$m = 1.75$	150	80.197	11922.266	5.44	100
		$m = 2.00$		71.847	4106.746	5.32	
		$m = 2.25$		71.795	1331.418	5.49	
	$\alpha = 0.2$	$m = 1.75$	255	**84.979**	7910.072	11.59	100
		$m = 2.00$		74.745	2412.663	11.44	
		$m = 2.25$		72.600	682.426	11.92	
	$\alpha = 0.4$	$m = 1.75$	255	84.979	7910.072	11.85	100
		$m = 2.00$		73.337	2412.663	11.74	
		$m = 2.25$		71.803	682.426	11.90	
	$\alpha = 0.6$	$m = 1.75$	255	84.840	7910.072	11.94	100
		$m = 2.00$		73.248	2412.663	11.81	
		$m = 2.25$		71.803	682.426	11.93	
	$\alpha = 0.8$	$m = 1.75$	255	84.840	7910.072	11.95	100
		$m = 2.00$		73.248	2412.663	11.79	
		$m = 2.25$		71.803	682.426	11.92	

be explained because it might be the case that some of the cluster centers selected are still too close together (a remnant of the problem faced with the local version of the algorithm). Nevertheless, with this version we have the advantage of not having to specify the number of clusters beforehand, and it can be very useful to have an algorithm that estimates a good range for this number when we have no other information available.

Next we analyze the results on the POKER data set. In Table 5.6 we can see how the comparison algorithms perform on this data set, observing almost the same behaviour as in the previous one. The Random Forest classifier still gets the best performance, followed

Table 5.5: Experimental results of the FCM algorithm with initialization given by Chiu's algorithm for the HAR data set.

Algorithm			Centroids	Acc	Loss	T	#
FCM + ChiuI	$\alpha = 0.2$	$m = 1.75$	255	**80.509**	8803.850	11.97	100
		$m = 2.00$		72.230	2609.958	11.95	
		$m = 2.25$		71.856	732.872	12.04	
	$\alpha = 0.4$	$m = 1.75$	255	80.396	8803.850	12.14	100
		$m = 2.00$		71.666	2609.958	11.89	
		$m = 2.25$		71.862	732.872	12.17	
	$\alpha = 0.6$	$m = 1.75$	255	79.998	8803.850	12.20	100
		$m = 2.00$		71.600	2609.958	11.95	
		$m = 2.25$		69.112	732.872	12.25	
	$\alpha = 0.8$	$m = 1.75$	255	79.515	8803.850	12.31	100
		$m = 2.00$		71.600	2609.958	11.95	
		$m = 2.25$		69.102	732.872	12.16	

closely in this case by the KMeans algorithm with 1000 centroids. The Logistic Regression model has very poor performance in this instance, and the subtractive clustering algorithm again finds itself in the middle, almost 10 points below the score that KMeans got with roughly the same number of centroids.

For the experimentation with the FCM algorithm on this data set we tested every combination of the parameters $\alpha \in \{0.2, 0.5, 0.8\}$ and $m \in \{1.75, 2.0, 2.25\}$, obtaining a curious result: in every case the accuracy was exactly the same. The ChiuI algorithm was run with parameters $r_a = 2.0$ and $r_b = 3.5$. In all instances the number of iterations before reaching convergence was relatively low, and the execution time was again admissible, averaging 47 seconds per iteration with 250 centroids and 85 seconds per iteration with 356 centroids. We show the average results on Table 5.7, in which we can appreciate how the initialization with the subtractive clustering algorithm does not introduce a significant overhead, while providing us with a reasonable number of centroids. In addition, as in the case of KMeans, the performance would most likely improve if we chose a higher number of centroids, but then the execution time would increase accordingly.

Lastly we arrive at the SUSY data set, which is the real test to see if our implementations can withstand such a large quantity of data or if, on the contrary, they take a disproportionate amount of time to finish. Looking at the comparison algorithms in Table 5.8 we see that we can expect higher execution times than in the previous data sets, and also that the general tone of the results remains the same. In this case the Logistic Regression algorithm does outperform the rest save for the Random Forest model, but the interesting thing is that it

Table 5.6: Experimental results of comparison algorithms for the POKER data set.

Algorithm	Centroids	Acc	T	#
Random Forest	–	**58.776**	1.17	–
Logistic Regression	–	34.198	0.12	–
KMeans	10	50.131	0.68	100
	25	50.237	0.78	
	50	50.811	1.07	
	100	51.850	1.67	
	250	52.743	3.46	
	500	53.679	5.91	100
	750	54.391	8.41	
	1000	55.019	10.91	
ChiuI	356	43.179	2.67	–

Table 5.7: Experimental results of the FCM algorithm for the POKER data set.

Algorithm	Centroids	Acc	T	#
FCM + Random	250	**50.117**	11.00	14
	356	50.117	20.00	14
FCM + ChiuI	356	50.117	21.00	14

does so many times faster (it takes less than a minute to finish). For the ChiuI algorithm we chose $r_a = 1.5$ and $r_b = 3.0$, getting a number of clusters just upwards of 600, while maintaining a controlled execution time.

In the case of the FCM algorithm we did not experiment with many parameters, since the execution times are higher. As we can see in Table 5.9 the algorithm needs in this case about 3 minutes per iteration with 250 centroids. This is not unreasonable if we take into account that we are dealing with almost 5 million points on each iteration and we are working in a 18-dimensional space. The accuracy is not as high as we would have liked, and falls short of the results observed in the comparison algorithms. However, this can be attributed to multiple factors, such as the relatively small number of centroids (we lowered it to decrease the execution time) and the fact that we did not perform an exhaustive grid

Table 5.8: Experimental results of comparison algorithms for the SUSY data set.

Algorithm	Centroids	Acc	T	#
Random Forest	–	**79.509**	5.46	–
Logistic Regression	–	77.931	0.47	–
KMeans	250	73.899	5.60	100
	500	74.746	11.87	
	1000	75.618	25.03	
ChiuI	605	65.217	13.18	–

search to find the optimal parameters. In addition, it may well be the case that this specific data set does not behave well when presented with these types of fuzzy algorithms. On the other hand, the value of α does not seem to influence the result.

In this case we needed to increase the number of partitions in which Spark internally divides the data set for it to work properly, but doing so we managed to get an algorithm that performs the tasks it was designed to do and can handle large amounts of data. This is precisely what scalability is all about.

Table 5.9: Experimental results of the FCM algorithm for the SUSY data set.

Algorithm			Centroids	Acc	T	#
FCM + Random	$\alpha = 0.2$	$m = 2.0$	250	**54.114**	274.15	90
	$\alpha = 0.4$		250	54.114	272.96	90
FCM + ChiuI	$\alpha = 0.2$	$m = 2.0$	256	53.244	283.22	91
	$\alpha = 0.4$		256	53.244	275.32	91

We perform a final comparison of the three clustering algorithms analyzed on the three data sets selected, looking only at the execution time. As we can see in Figure 5.1, the HAR and POKER data sets present a similar execution time even though the latter has almost 1 million data points more. As for the algorithms themselves, the KMeans algorithm is the fastest, as we already knew from a theoretical point of view. However, for the first two data sets the FCM algorithm remains in the same order of magnitude when it comes to execution time, which is a point in favour. The SUSY data set is the one that takes up the most time to finish its execution, as expected, but even then the running time is acceptable. The increase in time is also caused by the excessive number of iterations as compared to the ones observed in the POKER data set (14 versus 90), because the convergence here is

slower and the algorithm struggles to find a good solution. If the number of iterations was lower, the execution time would almost certainly be much closer to that of the POKER data set (which has almost 5 million points less).

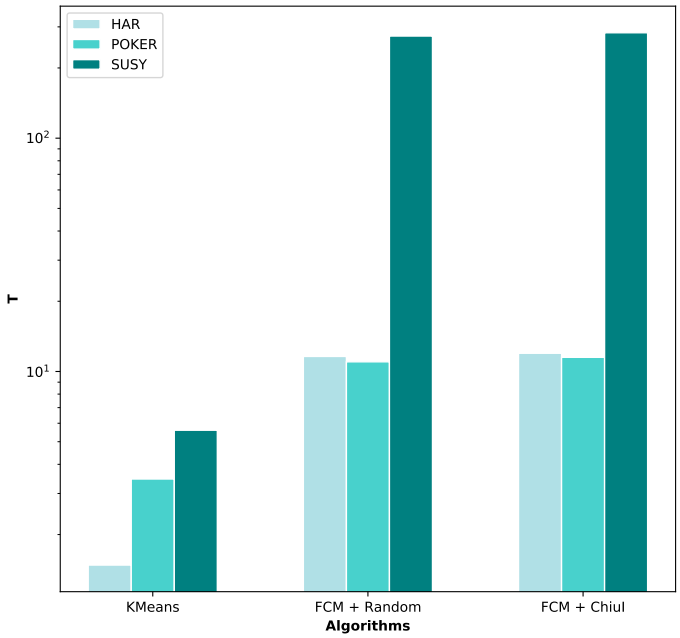


Figure 5.1: Comparison of execution time of the three clustering algorithms on the three data sets employed, using 250 centroids. The vertical axis represents the time in minutes, measured on a logarithmic scale.

5.3.2 Regression algorithms

Having reviewed all the classification algorithms implemented, we move on to the regression tasks. In this regard we will be experimenting with the adapted version of the subtractive clustering algorithm to build a FRBS, and also with the algorithm developed by Wang and Mendel for accomplishing that same goal. In both cases we build a Mamdani type fuzzy inference system. First of all we constructed a very small example to visualize how these algorithms performed on a simple function estimation, and also to ensure that they worked as intended. The result is shown on Figure 5.2.

Now we focus on the real experimentation on our two regression data sets. As we did with the classification models, we have included some well-known algorithms for comparison: a Random Forest regressor and a standard linear regressor with L2 regularization.

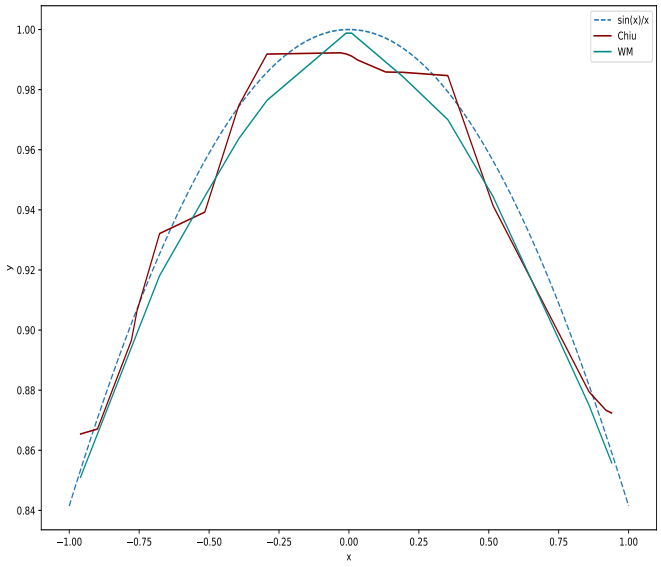


Figure 5.2: Models obtained with the Wang & Mendel and the subtractive clustering algorithms when trying to estimate the function $\sin(x)/x$ from a few points on its domain.

Apart from the execution time, which is our main concern, the metric to analyze in this case is the MSE. In the case of the Wang & Mendel algorithm we use the same number of partitions across every dimension, and in the case of the subtractive clustering algorithm we fix the stopping conditions in the same way as we before.

We begin with the smaller QAW data set, whose execution results can be seen in Table 5.10. We can see that the standard linear regression is the winner in this case, getting the best result in a negligible amount of time. On the other hand, what may seem strange is that the Random Forest regressor did not outperform almost any other algorithm, probably due to overfitting. Of the three different configurations tried for the Wang & Mendel algorithm, we observe that the best one is the one with less partitions (a total of 7), and it seems that increasing the number of partitions up to 15 results in a higher prediction error while also increasing the execution time. Lastly, Chiu's algorithm does quite well in this case, reaching the second best overall score in a reasonable amount of time. In this case the algorithm profits from a smaller neighbourhood radius, and the more cluster centers selected, the better the results.

Similarly, the results of the executions on the GSA data set are summarized in Table 5.11. The comparison algorithms behave in a similar way as in the previous case, but now the Random Forest does perform fairly well. However, in the Wang & Mendel algorithm

Table 5.10: Experimental results of every regression algorithm considered for the QAW data set.

Algorithm		Centroids	MSE	T
Random Forest		–	0.779	0.22
Ridge Regression		–	**0.098**	0.08
Wang & Mendel	$N = 3$	–	0.655	0.06
	$N = 5$	–	1.374	0.14
	$N = 7$	–	2.195	0.30
ChiuI	$r_a = 0.5$	2845	0.382	2.11
	$r_a = 0.75$	492	0.664	0.51
	$r_a = 1.0$	225	0.827	0.31

we observe an effect opposed to what we saw before, because in this case it seems that a higher number of partitions is better. This diametral change in the situation should not come as a surprise, since now we are considering 4 million points more, and it makes sense that a few more rules are needed to better explain the behaviour of the system.

The behaviour exhibited by Chiu's algorithm is also in direct opposition to the previous case, because now the results are better (and of course faster) with a lower number of clusters. This can be explained because, in a Big Data setting, having a huge quantity of points could mean that clusters might occupy more space, and fewer of them are needed to accurately describe the idiosyncrasies of the data. With regard to the execution time, the Wang & Mendel algorithm with 15 partitions finishes in less than half the time as the subtractive clustering algorithm, and also produces a better result. However, in both cases we can say that the time taken to finish is relatively low, especially when compared with the classification algorithms for the similarly sized SUSY data set.

Finally, we show a comparison of the execution time of all the algorithms in both data sets. In this case the conclusions drawn are even better, since it can be seen in Figure 5.3 that the execution times remain more or less in the same order of magnitude in each data set, but they also do not differ a lot when visualized together.

Table 5.11: Experimental results of every regression algorithm considered for the GSA data set.

Algorithm		Centroids	MSE	T
Random Forest		–	**0.301**	3.59
Ridge Regression		–	0.392	0.25
Wang & Mendel	$N = 3$	–	0.539	0.55
	$N = 5$	–	0.592	1.63
	$N = 7$	–	0.395	3.13
ChiuI	$r_a = 0.3$	438	0.627	8.89
	$r_a = 1.0$	246	0.471	8.13
	$r_a = 1.5$	225	0.453	8.72

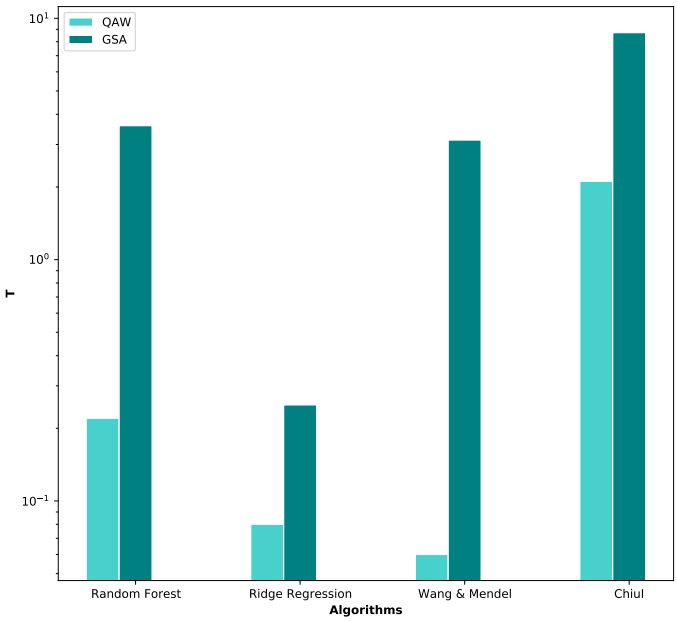


Figure 5.3: Comparison of execution time of the four regression algorithms on the two data sets employed. The vertical axis represents the time in minutes, measured on a logarithmic scale.

6 Conclusions and future work

Having concluded the exposition of the work we set out to do in the beginning to explore the interaction between fuzzy systems and Big Data, we can confidently assert that we have completed it satisfactorily. We will now bring this study to an end by summarizing the main tasks accomplished and the conclusions drawn from the various parts of this work.

In the first place, we introduced the theory of fuzzy systems from the ground up, reviewing the basics concepts of fuzzy set theory and fuzzy logic from a theoretical point of view, while never losing sight of the possible practical applications implied therein. Of special significance is the analysis of the process of fuzzy inference and fuzzy reasoning, and in particular of the fuzzy *if-then* rules, whose implications are clearly reflected on our practical work.

In the theoretical realm we also studied the concept of Big Data, its main characteristics and a plethora of aspects to take into account when dealing with this emerging field of computer engineering and data analysis. The main conclusion extracted from this disquisition is that new technologies and infrastructures are needed to keep up with the enormous quantity of data that we are constantly generating. Moreover, this shift in perspective is here to stay and will soon be the norm and not the exception, so it is of the utmost importance to develop robust algorithms that can help process and analyze such large amounts of data. In addition to this descriptive exposition, we have explored the ethical implications that come with the gathering, processing and eventual application of personal information. This is a topic we feel that, while being tangential to our study, merits a profound reflection as well. Though brief, in our discussion about Big Data ethics we concluded that the individuals should own their data and have a say in how it is utilized. They also need to be protected so that this remains so: it should always be the case that data is used *for* the subjects and not *against* them. Together with this exposition we presented a review of the structure of Spark and the MapReduce programming model, which is considered to be the state-of-the-art framework as far as distributed programming is concerned.

From the implementation task we have undertaken there are also some valuable conclusions to be extracted.

- The adaptation of the usual algorithms to a Big Data environment cannot be a mere parallelization of the sequential version. In most cases a careful planning of a scalable design is needed to succeed in this endeavour, or else the results will most likely

be disappointing. We experienced this situation in the first algorithm developed, for which we needed to implement three versions until the results were admissible time-wise. Nevertheless, in the end we developed three fuzzy learning algorithms that are scalable and can be readily used in any Big Data problem, which was the main goal of this practical portion of the work.

- Fuzzy systems are traditionally used as the backbone of control systems and other processes that require a high level of interpretability. By fusing this approach with the scalability that comes with an algorithm designed specifically to handle big amounts of data, we have reached a compromise in which we can increase the information available to learn while maintaining some notion of how this information is learned and reflected in the final model. This contrasts with some celebrated models such as *deep neural networks*, which act almost like a black box that receives an input and produces an output in some intricate way.
- The fact that we used a framework of reference such as Spark to implement the algorithms referred to in this work results in a more easily maintainable and future-proof implementation, with plenty of documentation and libraries available. A sufficient knowledge of the Scala programming language, of the Spark paradigm and of the interaction between them was needed before the implementation could begin. For this reason we dedicated a considerable amount of time to this learning effort, and as a side effect we began to develop a certain affinity for a more functional style of programming. However, some difficulties did arise while completing the implementation, and a more in-depth study into the inner workings of Spark was needed in order to overcome some of them. A brief description of these mishaps is presented in Appendix B.

Finally, we explore a few options for future work on this topic. The first and more straightforward path would be to fully adapt the implementation of the algorithms developed to integrate them in Spark's MLlib. The second idea would be to build upon our implementations and develop a complete suite of scalable fuzzy rule-based systems, providing a common interface and improving their easiness of use. Lastly, an interesting approach would be to inquire into the interpretability of these systems and analyze to what extent the fuzzy rules that stem from them lend themselves to be interpreted by humans, exploring the so-called *accuracy-interpretability trade-off*. In this regard a good starting point for research would be the works of Alcalá *et al.* [AAC+06; GAH11].

Part II

Ordinary differential equations not solved for the derivative

Mathematics

7 Introduction

The theory of differential equations came into existence after Newton and Leibniz invented calculus in the late 17th century. The concept of a relation between a function and its derivatives quickly became an essential one in the mathematical works of the time, and renowned mathematicians such as Euler, Lagrange or the Bernoulli brothers dedicated a great part of their lives to the study of these equations[5]. Since then, it has arguably become one of the main topics of research in the mathematical community, not only because it comprises the methods and techniques of many other fields (analysis, geometry, algebra...), but also because of its extensive practical applications to solve real-life problems.

Throughout modern history, many problems and phenomena in classical mechanics, physics, biology and other sciences have ended up being modelled by differential equations, either explicitly or indirectly. Among these situations, a certain type of equation seemed to be recurrent, one in which the derivative could not be explicitly isolated in one side of the equation. These equations came to be known as *equations not solved for the derivative* or *implicit differential equations*, and in the first order case they take the general form $F(x, y, y') = 0$. Thus, there was an interest, both practical and theoretical, in studying and understanding these implicit equations, and this is precisely the motivation behind the topic of this work.

We will focus on first order differential equations not solved for y'. While the theory of explicit differential equations is well-known and established in the curriculum of every undergraduate course in mathematics, implicit equations are usually not treated in depth from a theoretical point of view. This is why we will first make a brief summary of the most essential aspects of the theory of ordinary differential equations, and then introduce the theory of implicit equations, explaining how the former affects the latter.

The main references used for the writing of this document were the works of two Russian mathematicians, Vladimir Arnold [Arn12; Arn92] and Ivan Petrovsky [Pet66]. Citations of these and other complementary sources are made within the text.

7.1 Objectives

The goals of this study are as follows:

[5] A fairly complete historical review on the subject can be consulted in [AFG04].

1. To review the theory of first order differential equations at an undergraduate level, highlighting the geometrical interpretation of these equations and stating the most relevant theorems on existence and uniqueness of solutions.
2. To carry out a thorough study on equations not solved for the derivative, developing a theory rooted in the general theory of differentiable mappings, and providing general conditions for their treatment.
3. To analyze the so-called singular solutions of implicit equations, which are solutions that usually arise outside of the general family of solutions, and can be thought of as the set of critical points of a certain mapping. These singular solutions have special and interesting properties that warrant their study.
4. To present some classical equations such as the Lagrange and Clairaut equations, explain how they fit in the theory of implicit equations described before, and provide explicit techniques for their resolution.

7.2 Structure

In Chapter 8 we present the theory of first order ordinary differential equation, starting from the most basic definitions. First, the concepts of a differential equation and its solutions are rigorously defined. Next, we occupy ourselves with a geometrical analysis of differential equations, arriving at the concepts of integral curves and direction fields. Lastly, the theorems of Peano and Picard-Lindelöf are presented, along with a very schematic description of their proof and a reference where one can read them in their entirety.

In Chapter 9 the study of equations not solved for the derivative is carried out. We begin by showing a simple condition that allows us to write implicit equations locally in explicit form, namely the condition for the *implicit function theorem* to hold. Next, we present a result that establishes an algebraic condition under which there are finitely many solutions passing through a specified point on the plane. Following this result, we shift our perspective again and return to the geometrical point of view, regarding an equation $F = 0$ as a surface on a certain three-dimensional space, and its solutions as projections onto a two-dimensional plane. Lastly, we study the concepts of singular points, singular solutions and envelopes of curves, which still retain a notable geometrical component.

Finally, in Chapter 10 some concrete and famous examples of implicit equations are presented. In particular, we analyze and provide a method for solving Clairaut, Lagrange and Chrystal equations, applying the techniques studied throughout this work and presenting a visual representation of the general and singular solutions.

8 Basic concepts

In this chapter we will introduce the basic terminology needed to understand the theory that will subsequently be developed. We will present ordinary differential equations and their solutions in a rigorous manner, as well as some concepts intimately related to them, such as integral curves and initial value problems, and analyze how these equations fit into the general theory of differentiable functions. At the same time, we will set the notation that will be used throughout the exposition. The main reference for this chapter is [Pet66].

8.1 Ordinary differential equations

By an *ordinary differential equation of order n* we mean a relation of the form

$$F(x, y, y', \dots, y^{(n)}) = 0, \tag{8.1}$$

where F is a real-valued function of $n + 1$ real variables, x is an independent variable, $y = y(x)$ is an unknown function and $y', \dots, y^{(n)}$ are the first n derivatives of this function with respect to x. When the value of n is understood or simply not relevant, we will refer to (8.1) only as an ordinary differential equation (ODE), with the word *ordinary* meaning that the unknown function depends solely on one independent variable. A central concept in our study will be that of a solution of such a differential equation, which is rigorously defined below.

Definition 8.1. A *solution* of the differential equation (8.1) is a function $\varphi : I \to \mathbb{R}$, where $I = (a, b)$, $-\infty \leq a < b \leq \infty$ is some open interval of the real line, satisfying the following conditions:

(i) The function φ has derivatives up to order n.
(ii) The identity $F(x, \varphi(x), \varphi'(x), \dots, \varphi^{(n)}(x)) = 0$ holds for all $x \in I$.

The process of finding solutions of a differential equation is also known as *integrating* the equation.

From now on, unless otherwise stated we will restrict ourselves to *first order* ODEs, which take the general form

$$F(x, y, y') = 0, \tag{8.2}$$

and thus their solutions are differentiable functions of x that reduce the equation to an identity when substituted for y. Furthermore, we will initially consider differential equations in *explicit* form, that is,

$$y' = f(x, y), \tag{8.3}$$

and we will assume that the function $f(x, y)$ is defined on some domain (open and connected subset) D of the (x, y)-plane.

Remark. Sometimes we will find it more useful to use Leibniz's notation and express the above equation as

$$\frac{dy}{dx} = f(x, y).$$

8.2 Geometric interpretation of ODEs

Continuing our description of differential equations, we shall now present a natural interpretation of equations such as (8.3) and a way to visualize them in the plane. To begin with, suppose that we draw a short line segment through every point (x, y) of D with slope equal to $f(x, y)$, obtaining a set of directions in D that is called the *direction field* of equation (8.3). In other words, we transform our (known) function f in a vector field on its domain. At this point we can restate the problem of solving (8.3) from a geometric perspective:

> *Find all differentiable curves $y = \varphi(x)$ in D whose tangents have directions belonging to the direction field of* (8.3).

To see how this is equivalent to solving (8.3), one need only recall that the slope of the tangent of a (differentiable) curve at any point coincides with its derivative at that point, and then look at Definition 8.1. However, this formulation of the problem has two noticeable caveats from a geometric point of view:

1. By considering only *graphs* over the x-axis we are excluding curves that are intersected more than once by vertical lines.
2. The imposition that the slope of the direction field generated in D be given by f automatically excludes directions parallel to the y-axis, because f would need to be infinite at those points.

We are interested in looking at solutions of differential equations in a more general setting. This is why, based on the previous reformulation, we will develop a broader concept of solution with a strong geometrical meaning, one that will address both difficulties outlined above. In the first place, we allow general curves in parametric form, that is, $x = \lambda(t)$,

$y = \mu(t)$, $-\infty \leq \alpha < t < \beta \leq \infty$. This solves the first problem. To overcome the second limitation, in addition to equation (8.3) we consider the associated differential equation

$$\frac{dx}{dy} = f_1(x, y), \tag{8.3'}$$

where

$$f_1(x, y) = \frac{1}{f(x, y)}.$$

The rationale behind this decision is to allow the function f to become infinite at some points (representing a vertical slope), and shift our perspective from the x-axis to the y-axis at those points. In this way, at points where both f and f_1 are defined we can use either (8.3) or (8.3') to construct the direction field, but we use (8.3') at points where f becomes infinite[6].

We are now ready to define our new, more general concept of solution, which will necessarily be related not only to the original equation, but also to the newly defined associated equation. We will restrict ourselves to the class of *smooth regular* curves, that is, continuously differentiable curves with non-vanishing derivative everywhere. From now on, all curves will implicitly be assumed to belong to this class.

Definition 8.2. An *integral curve* of equations (8.3) and (8.3') is a smooth regular curve whose tangent at each point has a direction specified by the direction field of said equations.

Given that equations (8.3) and (8.3') are closely related, sometimes we will use the singular form to encompass both of them as a single equation. As we are looking at curves in parametric form, to obtain an explicit formula in this case we may first apply the chain rule,

$$\frac{dy}{dt} = \frac{dy}{dx}\frac{dx}{dt},$$

and then we solve for dy/dx, getting

$$\frac{dy}{dx} = \frac{dy/dt}{dx/dt} = \frac{M(x, y)}{N(x, y)}. \tag{8.4}$$

If the chosen curve does indeed satisfy the equation, then $f = M/N$ will hold. For simplicity we will seldom write down explicitly the associated equation,

$$\frac{dx}{dy} = \frac{N(x, y)}{M(x, y)} = f_1(x, y), \tag{8.4'}$$

[6] At each point of D, at least one of f or f_1 is always defined, for $f = 0$ if and only if f_1 is infinite, and $f_1 = 0$ if and only if f is infinite.

but we shall sometimes write (8.4) as

$$M\,dx - N\,dy = 0,$$

to stress the symmetry in x and y. This equation specifies a direction field at every point where both M and N are defined and at least one of them is non-zero.

Even though we are allowing general curves in parametric form, they are not (the graph of) a true solution of our equation unless we restrict them to some interval in which they can be expressed as a graph. We can think of the family of all solutions to equation (8.3) as a subset of the family of integral curves of (8.3) and (8.3'), restricted to a suitable interval if need be. An example visualization of direction fields and integral curves is shown in Figure 8.1.

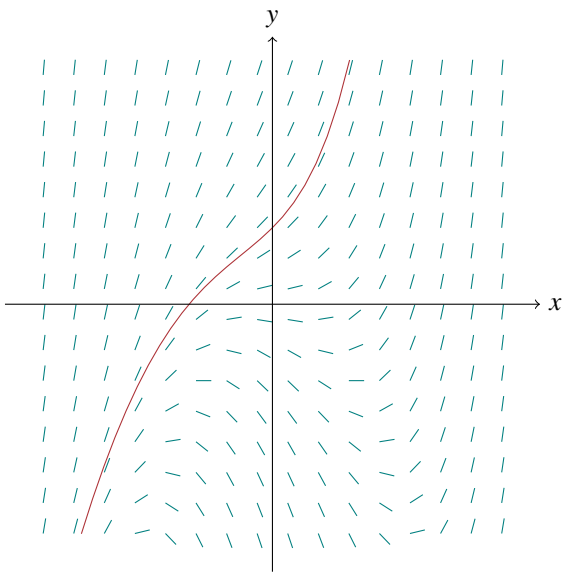


Figure 8.1: Direction field of the equation $y' = x^2 + y$ near the origin and a particular integral curve.

The requirement that integral curves be regular guarantees that we can always shrink their domain so that they become a graph, and hence a solution to our differential equation. The following result delves into that question.

Proposition 8.3. *Every point in a smooth regular curve belongs to an arc which is a smooth graph.*

Proof. Let $x = \lambda(t)$, $y = \mu(t)$, $\alpha < t < \beta$ be a smooth regular curve, and pick $t_0 \in (\alpha, \beta)$. The regularity of the curve guarantees that either $\lambda'(t_0)$ or $\mu'(t_0)$ is non-zero, so suppose without loss of generality that $\lambda'(t_0) \neq 0$. Since $\lambda'(t_0)$ is continuous by assumption, there exists an $\varepsilon > 0$ such that $\lambda'(t) \neq 0$ for $t \in (t_0 - \varepsilon, t_0 + \varepsilon)$, and then the equation $x = \lambda(t)$

has a unique smooth solution in t by virtue of the *inverse function theorem* (see [Apo74, p. 372]). Then, by eventually shrinking ε we may write $t = \psi(x)$, $\lambda(t_0 - \varepsilon) < x < \lambda(t_0 + \varepsilon)$. Substituting in $y = \mu(t)$ yields $y = \mu(\psi(x)) = \varphi(x)$, $\lambda(t_0 - \varepsilon) < x < \lambda(t_0 + \varepsilon)$, which is the equation of a smooth graph. ■

Remark. The conditions on integral curves in Definition 8.2 imply that our solutions are also smooth, that is, continuously differentiable.

We can illustrate the previous definitions and properties with a simple example.

Example 8.4. Consider the equation

$$\frac{dy}{dx} = \frac{y}{x},$$

which we can also write as

$$x\,dy = y\,dx.$$

Solving it by elementary methods gives the family of solutions

$$y = kx, \quad x \in \mathbb{R} - \{0\},\ k \in \mathbb{R}.$$

On the other hand, this equation defines a direction field in the whole plane minus the origin. To find its integral curves we consider the associated equation $dx/dy = x/y$ in the subset

$$\{(0, y) : y \in \mathbb{R} - \{0\}\}$$

of the plane. Combining both equations we can see that the set of all integral curves is given by the relation

$$ax + by = 0, \quad (x, y), (a, b) \in \mathbb{R}^2 - \{(0, 0)\}.$$

The direction field is shown schematically in Figure 8.2. Notice that we have found two integral curves for this equation that are not the graph of any solution, namely the two vertical rays emanating from the origin.

As a closing remark, we note that we could have ignored the directions parallel to the y-axis altogether, and we still would have ended up with a well defined, reasonable concept of integral curve. This is in fact what Arnold does in [Arn92]. However, taking these conflicting points into account will allow us to better study and understand *singularities* of the function f, allowing for a more general perspective and broadening the scope of this work.

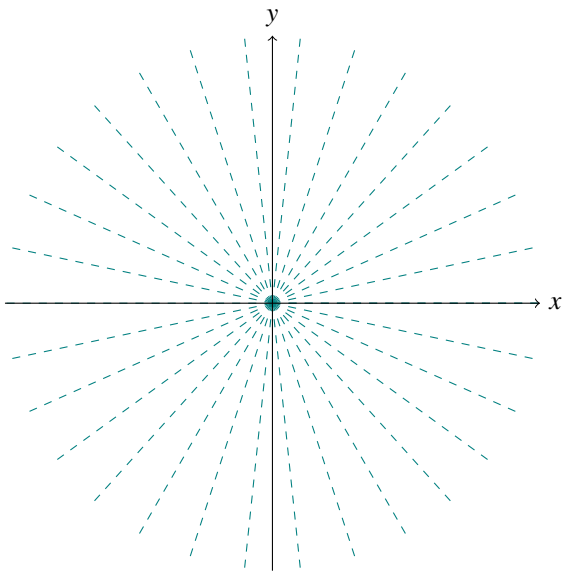


Figure 8.2: Direction field of the equation $x\,dy = y\,dx$ near the origin.

8.3 Existence and uniqueness of solutions

So far we have been talking about solutions of a differential equation without stopping to think even if such solutions exist in the first place. Could we come up with an equation which has no solutions at all? To answer this question, remember that any possible solution is assumed to be smooth, and thus its derivative verifies the *intermediate value theorem*[7]. But this derivative must be equal to f, so it suffices to choose a function that does not have the intermediate value property. For example, if we consider the famous Dirichlet function

$$g(x) = \begin{cases} 1 & \text{if } x \in \mathbb{Q}, \\ 0 & \text{elsewhere,} \end{cases}$$

then the equation $y' = f(x, y) = g(x)$ has no solutions.

From this example we can speculate that the lack of continuity of $f(x, y)$ is the key property that makes the equation unsolvable. This is indeed the case, as it turns out that existence of solutions is always guaranteed if we impose that f be continuous.

Theorem 8.5 (Peano's existence theorem). *Let $f(x, y)$ be a continuous function on a domain $D \subset \mathbb{R}^2$. Then the differential equation*

$$y' = f(x, y)$$

[7] A similar argument for non-smooth solutions may be given using *Darboux's theorem* [Apo74, p. 112], which states that the derivative of any differentiable function that takes two different values on an interval must take every value in between.

has at least one solution in D. Moreover, if f is continuous on a neighbourhood of a point $(x_0, y_0) \in D$, then there exists a solution φ defined on a neighbourhood of x_0 satisfying $\varphi(x_0) = y_0$.

Proof. See [Pea90] for Peano's original 1890 proof, which uses a method known as successive approximations. Shorter proofs have been proposed since, many using topological arguments such as fixed points theorems (see for example [Hal80, p. 14]). ∎

Remark. If we assume continuity of the function f, then the smoothness of any solution comes as a consequence and not as a hypothesis, since $\varphi'(x) = f(x, \varphi(x))$. In fact, the level of regularity of the function f translates in a similar manner to the solutions: if f has continuous derivatives up to order m, then any solution to $y' = f(x, y)$ has continuous derivatives up to order $m + 1$.

The concept of a solution passing through a predefined point is not incidental to our study, but it plays a major role in much of the subsequent theory, which is why it merits its own denomination. A differential equation $y' = f(x, y)$ coupled with an *initial condition* $y(x_0) = y_0$ is called an *initial value problem*, often abbreviated as IVP. With this new definition in mind, Peano's theorem states that every initial value problem defined by a continuous function has at least one solution, or more generally, that there is at least one integral curve of the equation passing through the selected point.

Together with the question of existence comes inevitably the issue of uniqueness. Peano's theorem guarantees that under mild conditions a solution to an initial value problem always exists, but it does not say anything about it being unique. It is not hard to think of an example in which uniqueness of solution is violated. For instance, it is immediate to verify that the IVP

$$\begin{cases} y' = 3y^{2/3}, \\ y(0) = 0 \end{cases}$$

has two valid solutions: $y = 0$ and $y = x^3$. Thus some kind of additional condition is needed to ensure that one and only one integral curve passes through a given point, as the following theorem asserts.

Theorem 8.6 (Picard-Lindelöf). *Let $D \subset \mathbb{R}^2$ be a domain in the plane, and suppose $f : D \to \mathbb{R}$ is a continuous function that locally satisfies a Lipschitz condition in the second variable, that is, for every $p \in D$ there exists a neighbourhood U_p of p in D and a constant $L > 0$ such that*

$$|f(x, y_2) - f(x, y_1)| \leq L\,|y_2 - y_1|$$

whenever $(x, y_1), (x, y_2) \in U_p$. *Then for every point* $(x_0, y_0) \in D$ *the initial value problem*

$$\begin{cases} y' = f(x, y) & \text{in } D, \\ y(x_0) = y_0 \end{cases} \tag{8.5}$$

has exactly one solution.

Proof. A proof in modern terms can be found in [Tes12, p. 36]. The idea is to construct a sequence of approximate solutions, the so-called method of *Picard iterations*, and then leverage the Lipschitz condition to apply the *contraction principle*[8]. ■

Remark. The Lipschitz condition on Theorem 8.6 can be relaxed to a condition on the partial derivative. The theorem remains true, via a direct application of the *mean value theorem*, if $\partial f / \partial y$ is locally bounded, which in turn holds if said partial derivative is continuous on D. This condition is often easier to check.

Bringing back the geometric interpretation of differential equations, it is straightforward to see that at points where there is uniqueness of solutions two integral curves can never be tangent to each other (they cannot cross either by definition). However, it is precisely the points in which uniqueness fails that interest us, and we will study them in depth in the following chapter.

[8] Also known as *Banach fixed point theorem*, after Polish mathematician Stefan Banach who first proved it in 1922 [Ban22].

9 Implicit equations

Having presented ordinary differential equations and their solutions in the previous chapter, we now focus on our main goal: the study and understanding of *implicit* differential equations, that is, equations which are not solved for the derivative y'. Since we cannot put this type of equations into explicit form, all the theory discussed earlier does not apply directly to this case, and thus we want to find a general framework that allows for the treatment of these equations from a theoretical point of view. For this task we will refer to [Pet66, § 25] and [Arn12, § 3].

Given that equations are expressed in the general or implicit form, it makes sense to think that the *implicit function theorem* will play a significant role in disentangling the complexities behind this formulation of differential equations, and that it will tell us something about their local behaviour. In an effort to provide all the necessary tools for the study we are about to undertake, we write down the statement of this famous theorem below.

Theorem 9.1 (Implicit function theorem). *Let D be an open set in $\mathbb{R}^{n+m}$ with coordinates (x, y) and let $F : D \to \mathbb{R}^m$ be continuously differentiable. If $(a, b) \in D$ verifies $F(a, b) = 0$ and the Jacobian matrix $\frac{\partial F}{\partial y}(a, b)$ is non-singular, then there exists an open set $U \subset \mathbb{R}^n$ with $a \in U$ and a unique C^1 function $f : U \to \mathbb{R}^m$ such that:*

(*i*) $f(a) = b$.
(*ii*) $F(x, f(x)) = 0$ *for all* x *in* U.
(*iii*) *The Jacobian matrix of f is given by the matrix product*

$$Jf(x) = -\left(\frac{\partial F}{\partial y}(x, f(x))\right)^{-1} \left(\frac{\partial F}{\partial x}(x, f(x))\right), \quad x \in U.$$

Proof. A standard proof employing the equally renowned inverse function theorem can be found in [Apo74, p. 374]. The last differentiation formula is obtained by differentiating the expression $F(x, f(x)) = 0$ and applying the chain rule. ∎

Even though we have stated the theorem for arbitrary dimensions, we will only apply it when $n = 2$ and $m = 1$, making the Jacobian matrix a single partial derivative. In fact, we recover the notation established in the previous chapter, and henceforth consider a function $F : \mathbb{R}^3 \to \mathbb{R}$ in (x, y, p)-space and the associated differential equation

$$F(x, y, y') = 0. \tag{9.1}$$

Also, following a standard notation, we will denote by F_x, F_y and F_p the partial derivatives of the function F with respect to each of its three variables x, y and p. Then, the theorem assumes the following form:

Corollary 9.2. *Let D be an open subset of $\mathbb{R}^3$ and let $F : D \to \mathbb{R}$ be continuously differentiable. If $(x_0, y_0, p_0) \in D$ verifies $F(x_0, y_0, p_0) = 0$ and $F_p(x_0, y_0, p_0) \neq 0$, then there exists an open set $U \subset \mathbb{R}^2$ containing (x_0, y_0) and a unique C^1 function $f : U \to \mathbb{R}$ such that $p_0 = f(x_0, y_0)$ and $F(x, y, f(x, y)) = 0$ for all $(x, y) \in U$. Moreover, the partial derivative of f with respect to its second variable is given by*

$$\frac{\partial f}{\partial y}(x, y) = -\frac{F_y(x, y, f(x, y))}{F_p(x, y, f(x, y))}. \tag{9.2}$$

9.1 Integral curves of implicit equations

In this case we will start with an example. If we consider the differential equation

$$(y')^2 = y, \tag{9.3}$$

we should immediately observe that this expression is an abbreviation for two distinct explicit differential equations: $y' = \sqrt{y}$ and $y' = -\sqrt{y}$, for $y > 0$. Each of these equations produces a direction field, as seen in Figure 9.1, and each of them separately satisfies an existence and uniqueness theorem on its domain, though clearly the uniqueness of solution fails when combining them. Nevertheless, we would like to examine this kind of equations as a whole, and establish general conditions for their treatment.

One thing to note from the beginning is that there are some occasions in which implicit equations can be put in explicit form, at least in theory. That is, there is a set of conditions that, when satisfied, guarantee that the equation is equivalent to another one in explicit form. Of course, in non-trivial cases this result will only hold locally, and as one would expect it is a direct consequence of the aforementioned implicit function theorem.

Theorem 9.3. *Let $F : D \to \mathbb{R}$ be a continuously differentiable function on some domain D of the (x, y, p)-space. Suppose that $(x_0, y_0, p_0) \in D$ simultaneously satisfies $F(x_0, y_0, p_0) = 0$ and $F_p(x_0, y_0, p_0) \neq 0$. Then there exists a unique function φ defined on an interval containing x_0 that is a solution of the equation $F(x, y, y') = 0$, satisfying $\varphi(x_0) = y_0$ and $\varphi'(x_0) = p_0$.*

Proof. Under these conditions we can apply Corollary 9.2 to $F(x, y, p)$ and express p as a function of x and y, that is, there exists a unique smooth f such that $F(x, y, f(x, y)) = 0$ on a neighbourhood U of the point (x_0, y_0), that also verifies $p_0 = f(x_0, y_0)$. Then the IVP

$$\begin{cases} y' = f(x, y) & \text{in } U, \\ y(x_0) = y_0 \end{cases}$$

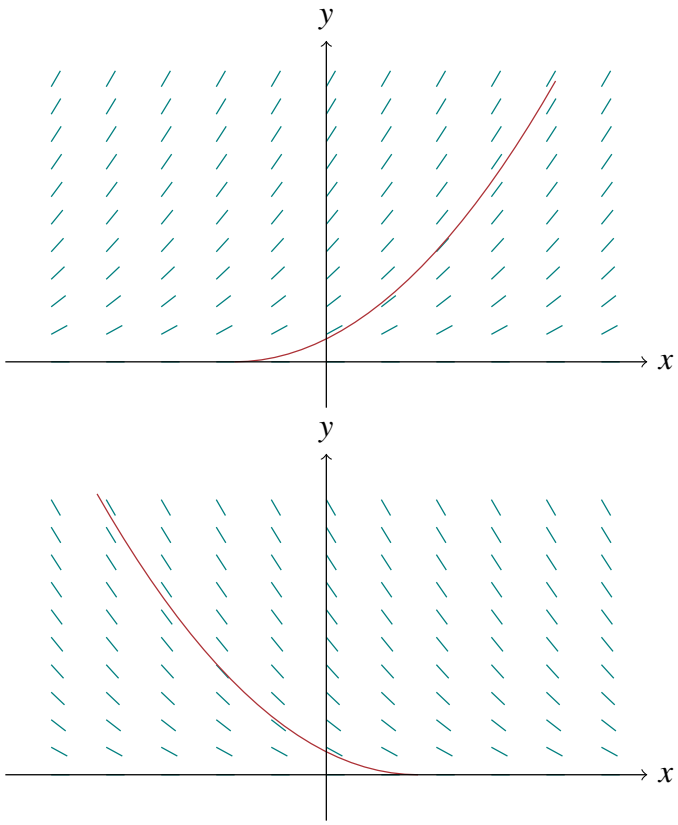


Figure 9.1: Direction fields of the two equations combined in the notation $(y')^2 = y$.

is well defined, and since f_y is continuous, it verifies the conditions of Theorem 8.6. Thus, there is a unique smooth function φ defined on a neighbourhood I of x_0 such that $\varphi'(x) = f(x, \varphi(x))$ on I and $\varphi(x_0) = y_0$. This concludes the proof, seeing as how putting it all together we have

$$p_0 = f(x_0, y_0) = f(x_0, \varphi(x_0)) = \varphi'(x_0)$$

and

$$F(x, \varphi(x), \varphi'(x)) = F(x, \varphi(x), f(x, \varphi(x))) = 0, \quad x \in I. \qquad \blacksquare$$

Remark. Uniqueness of solution may fail if the requirement that $F_p \neq 0$ is not satisfied. For example, consider the equation

$$(y')^2 - 2y' + 4y - 4x + 1 = 0,$$

for which at the point $(0, 0, 1)$ we have $F_p = 0$ but both x and $x - x^2$ are valid solutions that satisfy the initial conditions.

If we take a closer look at the above result, we observe that for every choice of p_0 there is a potentially different solution to the equation. But just how many suitable points are there? Is there any condition that guarantees that the number of possible solutions passing through a given point in the plane is finite? The following theorem (cf. [Pet66, p. 76]) gives a satisfactory answer, namely that we can limit the number of solutions in a neighbourhood if after setting a reference point (x, y) we can solve the (algebraic) equation for p.

Theorem 9.4. *Let $F : D \to \mathbb{R}$ be a continuous function defined on some domain $D \subset \mathbb{R}^3$ with coordinates (x, y, p). Suppose that $(x_0, y_0) \in \mathbb{R}^2$ is such that the equation $F(x_0, y_0, p) = 0$ has a finite number of distinct roots $p_1, \ldots, p_m$ when solved for p, verifying that $(x_0, y_0, p_i) \in D$ and that $F_p(x_0, y_0, pi) \neq 0$ for $i = 1, \ldots, m$. Further suppose that for $i = 1, \ldots, m$ there exists a neighbourhood $\mathcal{R}_i$ of the point (x_0, y_0, p_i) in which F is continuously differentiable. Then there is a neighbourhood $\mathcal{N}$ of (x_0, y_0) such that precisely m solutions of the equation*

$$F(x, y, y') = 0 \tag{9.4}$$

pass through each point of $\mathcal{N}$.

Proof. Since the set $\{p_1, \ldots, p_m\}$ is finite, we can suppose without loss of generality that the $\mathcal{R}_i$ are cylinders parallel to the p-axis, where the projection of the base of each cylinder onto the (x, y)-plane is the same neighbourhood $\mathcal{N}$ of (x_0, y_0), as shown schematically in Figure 9.2. This can be done, eventually shrinking the neighbourhoods, because they all enclose the same point (x_0, y_0) in the (x, y)-plane. More precisely, there exist $r > 0$ and $\varepsilon > 0$ such that

$$\mathcal{R}_i = B_r(x_0, y_0) \times (p_i - \varepsilon, p_i + \varepsilon), \quad i = 1, \ldots, m.$$

In addition, we can choose ε sufficiently small so that

- The cylinders are disjoint, that is, $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$, $i \neq j$.
- For every $(x, y, p) \in \bigcup_{i=1}^{m} \mathcal{R}_i$ we have $|F_p(x, y, p)| \geq c > 0$ (this follows from the continuity of F_p).

Next we take a compact set G such that

$$\bigcup_{i=1}^{m} \overline{\mathcal{R}}_i \subset G \subset D,$$

and we will henceforth work in G, denoting

$$\mathcal{R} = \bigcup_{i=1}^{m} \mathcal{R}_i.$$

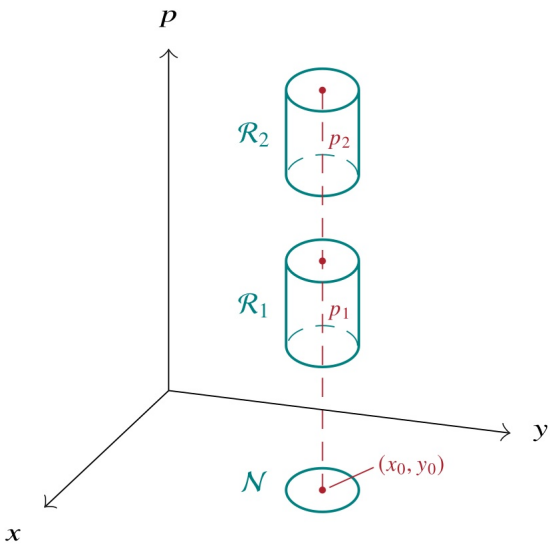


Figure 9.2: Representation of the cylindrical neighbourhoods of each p_i and the projection of every base onto the (x, y)-plane.

In this situation the implicit function theorem asserts that in each of the cylinders $\mathcal{R}_i$ (shrunk if necessary) the equation $F(x, y, p) = 0$ is satisfied by one and only one choice of p as a smooth function of x and y, that is, of the form

$$p = f_i(x, y), \quad i = 1, \ldots, m, \tag{9.5}$$

where in each case the derivative with respect to y is given by (9.2). Since $|F_p(x, y, f_i(x, y))| \geq c > 0$ by assumption and given that F_y is continuous, we can assure that each $\partial f_i/\partial y$ is bounded on $\mathcal{R}_i$.

Not only that, but we can also choose $\mathcal{N}$ so small that every point (x, y, p) in the cylinder generated by $\mathcal{N}$ for which $F = 0$ belongs to one of the surfaces (9.5). Indeed, if it were not the case, any such point would obviously have to lie outside the cylinders $\mathcal{R}_1, \ldots, \mathcal{R}_m$, and if they existed for arbitrarily small $\mathcal{N}$ they would have to be on the line $x = x_0$, $y = y_0$. To see this, we argue by contradiction and suppose that for all $n \in \mathbb{N}$ there exists a point $(x_n, y_n, p_n) \in G$ with the following properties:

(i) $F(x_n, y_n, p_n) = 0$.
(ii) $|(x_n, y_n) - (x_0, y_0)| < \frac{1}{n}$.
(iii) $(x_n, y_n, p_n) \in G \setminus \mathcal{R}$.

Since G is compact, $\{p_n\}$ is bounded, and we can suppose that it converges to some real value p_0. Together with property (ii), this information assures that

$$\{(x_n, y_n, p_n)\} \to (x_0, y_0, p_0).$$

Now, since $G \setminus \mathcal{R}$ is closed in G, (iii) implies that $(x_0, y_0, p_0) \notin \mathcal{R}$, and in particular that $p_0 \notin \{p_1, \ldots, p_m\}$. Finally, by virtue of (i) and the continuity of F, we conclude that

$$0 = F(x_n, y_n, p_n) \to F(x_0, y_0, p_0),$$

meaning that p_0 is another root of the equation $F(x_0, y_0, p) = 0$, which is impossible since its only roots were $p_1, \ldots, p_m$.

Thus we have proven that the point (x_0, y_0) has a neighbourhood $\mathcal{N}$ where $F(x, y, p) = 0$ has precisely m solutions (9.5). Each function f_i is continuous and has a bounded derivative with respect to y, so according to Theorem 8.6 each of the equations

$$y' = f_i(x, y)$$

has one and only one integral curve passing through any given point of $\mathcal{N}$. Repeating the reasoning in the proof of Theorem 9.3 we can see that these curves are indeed integral curves of (9.4). Because the values of y' associated with each integral curve are all different on $\mathcal{N}$ (since the cylinders $\mathcal{R}_i$ are disjoint), these integral curves are all different and no two make contact without intersecting. Moreover, if there was another integral curve apart from these ones, its derivative would have to take the values $\overline{p}_i \in \mathcal{R}_i$ and $\overline{p}_j \in \mathcal{R}_j$ for some $i \neq j$, and by the *intermediate value theorem for derivatives* (see footnote 7) it would have to take every value in between, which is impossible since F only vanishes on $\mathcal{R}$. Therefore, as asserted, precisely m solutions of equation (9.4) pass through each point of $\mathcal{N}$. ■

A careful look at the statement of this problem reveals that none of the direction fields defined by equation (9.4) can be parallel to the y-axis. However, as in the case of equations solved for y', we would like to include this possibility. Therefore, besides the equation

$$F\left(x, y, \frac{dy}{dx}\right) = 0 \tag{9.6}$$

we consider the associated equation

$$F_1\left(x, y, \frac{dx}{dy}\right) = 0, \tag{9.6'}$$

where F_1 is chosen in such a way that these equations are consistent.

As a final remark, we note that if we could explicitly solve for y' (think for example that F were a polynomial on y') then we could not only establish the existence and uniqueness of solutions, but also determine exactly what those solutions were, as in the next example.

Example 9.5. Let us revisit the equation

$$(y')^2 = y. \tag{9.7}$$

We know that two direction fields are combined in this equation (see Figure 9.1), and hence the direction field of (9.7) is obtained as the superposition of those two direction fields. Setting $F(x, y, p) = p^2 - y$ and solving $F = 0$ for p yields $p = \pm\sqrt{y}$, for every $y > 0$. We have $F_p = 2p$, so we can apply Theorem 9.4 to every point on the set

$$\{(x, y) \in \mathbb{R}^2 : y > 0\},$$

obtaining that there is a neighbourhood of each of these points in which there are two and only two integral curves of equation (9.7) passing through any given point. This is also obvious from Figure 9.1. What is more, in this case we can obtain an explicit expression of those integral curves by solving the two differential equations

$$y' = \pm\sqrt{y}, \quad y > 0, \tag{9.8}$$

whose joint family of solutions can be written as

$$y(x) = \frac{1}{4}(x + C)^2, \quad C \in \mathbb{R}.$$

They are convex parabolas with vertex on the x-axis, the upwards branch corresponding to the positive sign in (9.8) and the downwards branch to the negative sign.

Remark. The above example stresses the importance of working in open sets when it comes to uniqueness of solutions. If we were to solve equation (9.7) for $y \geq 0$, we would immediately note that the line $y = 0$ is an integral curve of the equation, and then an infinite number of solutions would pass through every point on the x-axis. In particular, for every $x_0 \in \mathbb{R}$ the functions parametrized by $k \geq x_0$ as

$$\varphi_k(x) = \begin{cases} 0, & x < k, \\ \frac{1}{4}(x - k)^2, & x \geq k \end{cases}$$

are all different solutions of the equation passing through the point $(x_0, 0)$.

9.2 Geometrical approach to implicit equations

When studying equations in implicit form, the previous results and techniques suggest a geometrical approach in which we consider the direction field not on the (x, y)-plane, but on the surface of the three-dimensional (x, y, p)-space given by the equation $F(x, y, p) = 0$, where $p = dy/dx$. This way, even though the equation might define several direction fields, we compensate for this by adding a new dimension in which to visualize them together. This space is known as the space of *1-jets* of functions $y(x)$, which is basically representing the truncated Taylor polynomial of a (differentiable) function at a given point. We can

regard its points as all the non-vertical directions[9] (those not parallel to the y-axis) at all points of the (x, y)-plane: a point (x, y, p) represents the direction of a line $dy = p\,dx$ at the point (x, y). In what follows, the direction of the p-axis will be referred to as the *vertical direction* in the space of 1-jets.

For the purposes of this section we will assume that F is a sufficiently differentiable function and that the surface

$$M = \{(x, y, p) : F(x, y, p) = 0\}$$

in the space of 1-jets is smooth[10]. We will show that a direction field arises on this surface and relate it to a certain direction field on the plane. Before we continue, we define a concept that will be essential in the development of the theory.

Definition 9.6. A point on the surface M is called a *regular point* if the tangent plane to the surface at that point is not vertical.

Let us recall that the tangent plane at a point (x_0, y_0, p_0) of the surface M is given by the equation

$$\langle \nabla F(x_0, y_0, p_0), (x - x_0, y - y_0, p - p_0) \rangle = 0, \tag{9.9}$$

so that it is non-vertical if and only if $F_p(x_0, y_0, p_0) \neq 0$. This condition should look familiar after seeing the results of the previous section. Let

$$\pi : M \to \mathbb{R}^2, \quad \pi(x, y, p) = (x, y)$$

be the projection of M to the (x, y)-plane in the vertical direction. A *critical point* of the mapping π will be a point for which $F = F_p = 0$, that is, a point on M that is not regular.

Thus, in a neighbourhood of a regular point we can apply yet again the implicit function theorem to show that M is the graph of a smooth function, say $p = f(x, y)$. In fact, it is a well-known result in the basic theory of differentiable surfaces that in this case the projection π is a local diffeomorphism (see [MRB09, p. 39]). Then, to each regular point corresponds its own differential equation (in a neighbourhood of the projection of said point) and the direction field that comes with it; all these direction fields on the (x, y)-plane are combined in the equation $F = 0$.

On the other hand, we forget the projection momentarily and consider a point (x, y, p) in the space of 1-jets and a vector ξ applied at this point, whose components will be denoted by $dx(\xi)$, $dy(\xi)$ and $dp(\xi)$. We now consider the plane of all such vectors at a point (x, y, p)

[9] We could circumvent this restriction with the usual trick of considering another function F_1, but we will not do it here for simplicity.

[10] This is not a strong restriction, since it holds if 0 is not a critical value of F, and by Sard's theorem this happens almost always. Even if it were not the case, a small perturbation by an additive constant should make it so.

for which $dy = p\,dx$[11], that is, the vectors that when projected onto the (x, y)-plane form an angle with the x-axis whose tangent is equal to p. Each of these planes is called a *contact plane*, and they are all vertical (they contain the direction of the p-axis, since the vector $(0, 0, 1)$ verifies the condition imposed). The set of all contact planes forms the *contact plane field* in the space of jets, and is called the *contact structure*[12]. Figure 9.3 can help visualize the various elements of this construction.

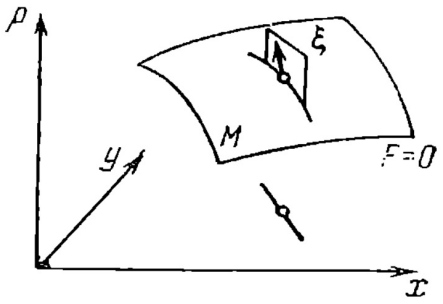


Figure 9.3: The surface M, the contact plane at a point and its projection. Taken from [Arn12].

Bringing back the projection, we study the contact plane at a regular point of the surface M. Since the contact plane is vertical, it intersects the corresponding tangent plane in a line, and this behaviour is reproduced in all the nearby points. In this way, in a neighbourhood of a regular point there arises a smooth direction field, given by the intersections of the contact and tangent planes at every point. The integral curves of equation (9.1) are, by definition, the integral curves of this direction field on M. We can even provide an explicit description of this direction field by analyzing the intersection of the two planes. Firstly, since the contact plane at a regular point (x, y, p) is given by the equation $p\,dx - dy = 0$, the normal vector to this plane at the point (x, y, p) is $(p, -1, 0)$, and secondly we know by equation (9.9) that the normal vector to the tangent plane at said point is $\nabla F(x, y, p)$. Then, the line resulting from the intersection of both planes at the point of interest is given by the direction

$$\nabla F(x, y, p) \times (p, -1, 0) = (F_p, pF_p, -(F_x + pF_y))\big|_{(x,y,p)},$$

and thus the direction field on M is defined by the vector field

$$V = (F_p, pF_p, -(F_x + pF_y)). \tag{9.10}$$

[11] In differential geometry this is known as (the kernel of) a 1-form.

[12] This term is not specific to this problem, but it is an important notion in a branch of geometry known as contact geometry, which falls outside of the scope of this work.

We are now ready to link these seemingly distinct direction fields (the one on M and the ones on the (x, y)-plane we mentioned earlier), though it should be clear by now what the conclusion will be: that the direction field on M projects locally onto a direction field on the plane.

Theorem 9.7. *The vertical projection onto the (x, y)-plane in the neighbourhood of a regular point maps integral curves of equation (9.1) on M into integral curves of the equation*

$$\frac{dy}{dx} = f(x, y) \tag{9.11}$$

in a neighbourhood of the projection of the point under consideration. Here f is a smooth function such that M is locally the graph of f.

Proof. We know by definition that the projection of a contact plane onto the (x, y)-plane is a straight line in the direction field of equation (9.11), because it holds that $p = f(x, y)$. Then, since π is a diffeomorphism in the neighbourhood considered, the direction field on M turns into the direction field of (9.11). Consequently, the integral curves turn into each other, as well. ∎

The conclusion we can extract from all this reasoning is that, in the neighbourhood of a regular point, an implicit equation can be turned into an explicit one and can be studied and solved with the usual methods. However, special attention should be paid to critical points, that is, points in which the equation does not reduce to an explicit one, and in fact we will expand on this matter in the next section. For now, we define a couple of concepts related to these points, and we bring this section to an end with an example.

Definition 9.8. The set C of critical points of the projection π is called the *criminant curve* of equation (9.1), and the set $\pi(C)$ of its images is called the *discriminant curve*.

Remark. The discriminant curve can be obtained in some cases by eliminating p in the equations $F = F_p = 0$.

Example 9.9. We want to study the equations $p^2 = x$ and $p^2 = y$ using the techniques of this section. For the first one, we write down the equations of a generic vector at a point (x, y, p) on M that belongs to the direction field:

$$\begin{cases} p^2 = x & \text{(the condition of belonging to } M\text{)}, \\ 2p\,dp = dx & \text{(the condition of being tangent to } M\text{)}, \\ dy = p\,dx & \text{(the condition of belonging to the contact plane)}. \end{cases}$$

We note that the criminant is given by the curve $p = 0$ on M, which results when projected in a discriminant curve consisting on the x-axis (it follows from the first condition that $x = 0$ when $p = 0$). In this case it is convenient to choose the coordinates (p, y) on M. Combining the conditions above we get that (in our chosen coordinates) the integral curves are defined by the equation

$$\frac{dy}{dp} = 2p^2,$$

which we can easily solve to get that the solutions are given by the relation $y + C = \frac{2}{3}p^3$. Projecting back to the (x, y)-plane yields integral curves defined parametrically by

$$x = p^2, \quad y = \frac{2}{3}p^3 + C, \quad C \in \mathbb{R}.$$

If we want, we can also reverse the change of variables to write down the solutions as a more direct relation between x and y:

$$(y + C)^2 = \frac{4}{9}p^6 \implies (y + C)^2 = \frac{4}{9}x^3, \quad C \in \mathbb{R}.$$

It can be checked that these solutions are semi-cubical parabolas with a cusp on the discriminant line $x = 0$, as seen in Figure 9.4.

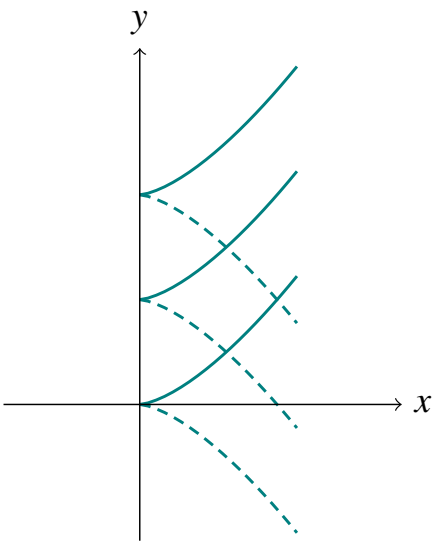


Figure 9.4: Integral curves of the equation $p^2 = x$ on the plane.

The other equation $p^2 = y$ is solved in a similar way. The conditions in this case are the following:

$$\begin{cases} p^2 = y, \\ 2p\,dp = dy, \\ dy = p\,dx. \end{cases}$$

The criminant is again $p = 0$, but in this case the discriminant curve is the x-axis. If we choose coordinates (x, p) on M the resulting equation outside of the criminant is

$$\frac{dx}{dp} = 2,$$

which yields solutions of the form

$$x = 2p + C, \quad y = p^2, \quad C \in \mathbb{R},$$

or equivalently,

$$(x + C)^2 = 4y, \quad C \in \mathbb{R}.$$

These are parabolas tangent to the line $y = 0$, as we already knew from Example 9.5.

9.3 Singular points and singular solutions

Another common terminology for referring to the points on the criminant curve is to call them *singular points* of the equation $F = 0$. It is worth pointing out that at some singular points it may well be the case that the contact plane is different from the tangent plane, so that they intersect in a straight line and thus still produce a direction field. In fact, we can extend the direction field defined in the previous section to include these well-behaved points, and say that this extended field is the *direction field of the equation* $F = 0$ *on* M. However, at singular points the conditions to apply the implicit function theorem do not hold (we have $F_p = 0$), so we can only assure that projections of the pieces of the integral curves (of the extended field) localized between singular points are locally integral curves of the corresponding equations $y' = f(x, y)$.

Following the distinction outlined above between singular points, and remembering the expression (9.10) for the direction field on M, we arrive at the following definition.

Definition 9.10. A singular point of equation $F = 0$ (a point on M for which $F_p = 0$) is called a *proper singular point* if $F_x + pF_y \neq 0$, and an *improper singular point* otherwise.

In the first case, if (x_0, y_0, p_0) is a proper singular point, the tangent and contact planes intersect each other at the straight line passing through (x_0, y_0, p_0) and having direction

$$v = (0, 0, -F_x(x_0, y_0, p_0) - p_0 F_y(x_0, y_0, p_0)).$$

In the second case, the contact and tangent planes coincide at the point (x_0, y_0, p_0) and no straight line is produced. Now, among proper singular points we distinguish those which verify an additional smoothness condition: we will refer by *regular singular point*[13] or *folded proper point* to a proper singular point that verifies

$$\operatorname{rank}((x, y, p) \mapsto (F, F_p)) = 2,$$

where the rank of a mapping is defined as the rank of its derivative (which is a linear map). Note that this condition guarantees via the implicit function theorem that the criminant curve is a smooth curve. Moreover, almost all singular points are of this type, by virtue of Sard's theorem (cf. [Arn12, p. 94]).

It turns out that at every regular singular point, the equation $F = 0$ is equivalent to another equation of the form $p^2 = x$ for a suitable change of coordinates. This is what is called the *normal form* of the equation $F = 0$.

Theorem 9.11 (Normal form). *Let (x_0, y_0, p_0) be a regular singular point of the equation $F(x, y, p) = 0$. Then, there exists a diffeomorphism of a neighbourhood of the point (x_0, y_0) in the (x, y)-space to a neighbourhood of the point $(0, 0)$ in the (X, Y)-space such that the equation $F = 0$ is reduced to the form $P^2 = X$, where $P = dY/dX$.*

Proof. A detailed proof can be found in [Arn12, p. 27], which in turn is based on the original 1932 article by Italian mathematician M. Cibrario [Cib32]. ∎

If we revisit Example 9.9 we realize that the equation in normal form can easily be solved, so the following result is immediate.

Corollary 9.12. *In a neighbourhood of a regular singular point, the family of solutions of the equation $F = 0$ is diffeomorphic to the family of semi-cubical parabolas $y = x^{3/2} + C$.*

The previous corollary allows us to call these singularities *cusp singularities*. We note that another type of singularities may arise in the discriminant curve, since in general several points of the criminant curve may be mapped by π onto the same point in the discriminant curve. These points will almost always be points of self-intersection of the discriminant curve, and they are called *fold singularities*. There are even more types of singularities that we could encounter, but a result by Whitney [Whi55] states that for an open dense set of functions F, the projection π is either a local diffeomorphism, a fold map or a cusp map at every point. That is to say, if a function presents another type of singularity, almost every small perturbation would eliminate them. Nevertheless, some attempts have been made at a classification of singularities of implicit equations; see [CR14] or [Dar75].

[13] Not to be confused with a regular point of π.

Finally, we explore the concept of singular solutions. A *singular solution* of equation $F = 0$ is a solution of the equation that is composed entirely of (projections of) singular points. Such solutions are special because they usually cannot be found as part of the general solution. More explicitly, if we have a parametrized family of solutions

$$\Phi(x, y, C) = 0, \tag{9.12}$$

generally there is no value of C such that when substituted in (9.12) a singular solution is obtained. For example, equation $p^2 = x$ in Example 9.5 presented a singular solution consisting on the line $x = 0$, which happens to be tangent at each point to the family of semi-cubical parabolas found as "regular" solutions. It turns out that this situation is not exceptional, and that most singular solutions occur in this way, so that the following definition is relevant[14].

Definition 9.13. An *envelope* of a family of plane curves is a curve that is tangent to each member of the family at some point, and these points of tangency together form the whole envelope.

Remark. It is obvious from the tangency condition that the envelope of a family of solutions to $F = 0$ is itself a solution. Moreover, at any point (x_0, y_0) of the envelope there are two solutions of the equation passing through it with the same slope p_0: the curve of the underlying family that is tangent to the envelope at that point and the envelope itself. Then we have necessarily that $F_p(x_0, y_0, p_0) = 0$, for if this were not the case it would contradict the uniqueness in Theorem 9.3. Thus we conclude that an envelope of a family of solutions to $F = 0$ is always a singular solution of that equation.

However, not every family of curves has an envelope. The following result provides some necessary conditions for this to happen, and in practice it gives us a method to find envelopes.

Theorem 9.14. *If a family of curves $\Phi(x, y, C) = 0$ has an envelope, then the points of the envelope must simultaneously satisfy the equations $\Phi = \Phi_C = 0$.*

Proof. Since the envelope is tangent to each member of the family of curves at some point, we can regard it as a map from the parameter C to the point of tangency. That is, we can write the points of the envelope as $x = x(C)$, $y = y(C)$. Since the envelope meets every curve of the family, it holds that

$$\Phi(x(C), y(C), C) = 0, \quad \text{for all } C,$$

[14] A study on the relation between singular solutions and envelopes was started by Darboux in 1873 [Dar73].

which is the first equation claimed to be satisfied by the envelope. Moreover, taking the derivative with respect to C in the previous expression and applying the chain rule yields

$$\Phi_x \frac{dx}{dC} + \Phi_y \frac{dy}{dC} + \Phi_C = 0. \tag{9.13}$$

On the other hand, the slope of the envelope at any point is given by

$$\frac{dy}{dx} = \frac{dy/dC}{dx/dC}, \tag{9.14}$$

and we can also take a particular curve and differentiate $\Phi(x, y, C)$ with respect to x (with C is held constant), getting that the slope at any point of that curve is expressed as

$$\Phi_x + \Phi_y \frac{dy}{dx} = 0. \tag{9.15}$$

Now, since for a fixed C the envelope is tangent to the curve $\Phi(x, y, C) = 0$ at $(x(C), y(C))$, we can combine (9.14) and (9.15) to get

$$\Phi_x \frac{dx}{dC} + \Phi_y \frac{dy}{dC} = 0,$$

which in light of (9.13) implies that $\Phi_C = 0$ at every point of the envelope, as asserted. ■

If we want to find an envelope for a given family of curves Φ, attending to the result just proved we could try to eliminate C from the equations $\Phi = \Phi_C = 0$ and check that the resulting curve verifies the conditions of an envelope. Even though this is the most common method to find envelopes, it is also interesting to analyze what are the sufficient conditions for their existence, to which the following theorem (proved in [Lan61, p. 10]) gives a complete answer.

Theorem 9.15. *If $\Phi(x, y, C)$ is a twice-differentiable function satisfying*

(i) $\Phi = \Phi_C = 0$.
(ii) $\Phi_{CC} \neq 0$.
(iii) $\begin{vmatrix} \Phi_x & \Phi_y \\ \Phi_{Cx} & \Phi_{Cy} \end{vmatrix} \neq 0,$

then there exists an envelope of the family of functions represented by Φ.

Example 9.16. Let us consider the differential equation

$$1 + (y')^2 = \frac{1}{y^2}. \tag{9.16}$$

The change of variables $t = 1 - y^2$ and a direct integration in both branches of the equation yields the family of solutions

$$\Phi(x, y, C) = (x + C)^2 + y^2 - 1 = 0, \quad C \in \mathbb{R}, \tag{9.17}$$

which are circles of radius 1 and center on the x-axis. To find the potential envelope, we differentiate this equation with respect to C, getting

$$\Phi_C(x, y, C) = 2(x + C) = 0,$$

which combined with $\Phi = 0$ implies that $y = \pm 1$. It is clear that the lines $y = 1$ and $y = -1$ are both tangent to the family (9.17), as seen in Figure 9.5, so they are indeed the envelopes of the equation (9.16). We can also check that these lines form the discriminant of the equation, since $F_p = 2p$ only vanishes at $p = 0$, which by (9.16) implies that $y = \pm 1$. Thus, we have found two singular solutions of the equation, and besides, they constitute the envelopes of the family of general solutions.

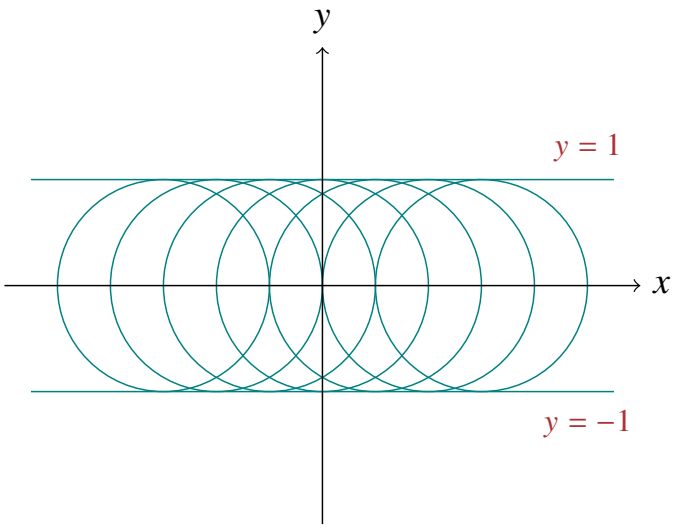


Figure 9.5: Representation of the family of general solutions of equation (9.16) and its envelopes $y = \pm 1$.

Example 9.17. A classical counterexample to the existence of envelopes is the family of concentric circles at the origin, described by the equation

$$\Phi(x, y, C) = x^2 + y^2 - C^2 = 0.$$

If we tried the usual trick of eliminating C from $\Phi = \Phi_C = 0$ we would end up with the equation $x^2 + y^2 = 0$, which in fact is satisfied only by the point $(0, 0)$, and thus does not constitute a curve, much less a curve tangent to the family of solutions. A look at Theorem 9.15 tells us that this family does indeed have no envelope, since condition (iii) does not hold at any point. This is also obvious from Figure 9.6.

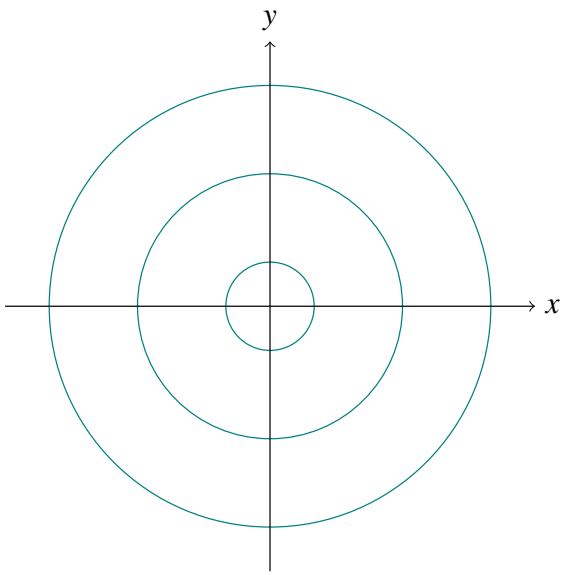


Figure 9.6: Representation of a family of curves with no envelope.

10 Classical examples of implicit equations

In this last chapter we will present, study and provide a general method to solve some classic implicit equations. These equations were first proposed in the 18th and 19th centuries by renowned mathematicians of the time, and have since been deeply studied. We feel it is illustrative to discuss them here and bring this exposition to an end with some relevant and non-trivial examples. We will try to provide a variety of perspectives into these equations, and apply when possible the techniques that we have studied throughout this work.

10.1 Clairaut equation

We begin with the study of an implicit differential equation introduced by French mathematician A. Clairaut in 1734, which appears in geometric problems in which it is required to determine a curve in terms of a prescribed property of its tangents at all points, as we will shortly verify. A Clairaut equation takes the form

$$y = xy' + f(y'), \tag{10.1}$$

where f is a continuously differentiable function. Clairaut was the first to point out the differences between the general and singular solutions of equations of this type, and this is why they bear his name. Since then, this equation has taken an important place in the theory of implicit equations; it has been generalized to higher dimensions and it has even been proposed as a model for a family of implicit equations with particular properties (see [Dar75]).

There are a number of ways to tackle this equation, ranging from an exhaustive theoretical study to a fast substitution method that works in practice. They all produce the same solutions, but the interpretation of the solving process is what changes among them. First we start with the geometrical approach. If we define $F(x, y, p) = y - xp - f(p)$ and revisit

the theory developed in Chapter 9, we should be able to write the equations of the surface $M = F^{-1}(0)$ and of the tangent and contact planes associated,

$$\begin{cases} y - xp - f(p) = 0, \\ p\,dx - dy - x\,dp - f'(p)\,dp = 0, \\ dy = p\,dx. \end{cases}$$

Now, from the second and third equations we get the relation

$$(x + f'(p))\,dp = 0. \tag{10.2}$$

On the other hand, we have $F_p = -(x + f'(p))$, so the criminant is composed by the points on M for which $x = -f'(p)$. Outside of those points, equation (10.2) implies that the integral curves on M are the lines $p = C$, and when projected on the (x, y)-plane (substituting $p = C$ in the expression of F) we get the family of lines

$$y = xC + f(C), \tag{10.3}$$

which is the general solution of the equation. Thus we can say that *a Clairaut equation is the equation of a family of lines parametrized by the slope*.

Turning our attention towards the points on the criminant, we note that they are all improper singular points, since $F_x + pF_y = 0$ holds everywhere. We can project these singular points and get the expression for the discriminant curve in parametric form as

$$x = -f'(p), \quad y = -pf'(p) + f(p), \tag{10.4}$$

where p is now the parameter along the curve. This singular solution turns out to be the envelope of the family (10.3) of straight lines with arbitrary slope, say p. Indeed, a quick calculation of the tangent at each point of the singular curve yields

$$\frac{dy}{dx} = \frac{dy/dp}{dx/dp} = \frac{-f'(p) - pf''(p) + f'(p)}{-f''(p)} = p,$$

given that $f'' \neq 0$ at the point of interest. It can also be checked that curves consisting on arbitrary segments of the singular solution and the two lines of the family (10.3) tangent to it at each end of the segment are solutions of the equation as well, including the degenerate case in which the segment is a single point. In other words, infinitely many solutions of the equation pass through each point of the envelope.

Another method for solving this equation more directly but arguably losing the intuition behind it is to consider the original expression $y - xy' - f(y') = 0$, introduce a parameter $p = dy/dx$ and differentiate with respect to x, obtaining

$$p - p - xp' - f'(p)p' = 0 \implies -(x + f'(p))p' = 0.$$

Now either $p' = 0$ or $x + f'(p) = 0$. In the first case we have $p = y' = C$, and substituting back in (10.1) one gets the general solution $y = xC + f(C)$. In the second case proceeding as before we can express the result in the same parametric form as in (10.4).

Example 10.1. We will now analyze the Clairaut equations with $f(p) = p^2$ and $f(p) = p^3$. For the first case the lines $y = xp + p^2$, $p \in \mathbb{R}$ constitute the family of general solutions, while the curve

$$x = -2p, \quad y = -p^2, \quad p \in \mathbb{R}$$

is the singular solution, which can also be written explicitly eliminating the parameter as $y = \frac{-x^2}{4}$. The general and singular solutions are shown in Figure 10.1.

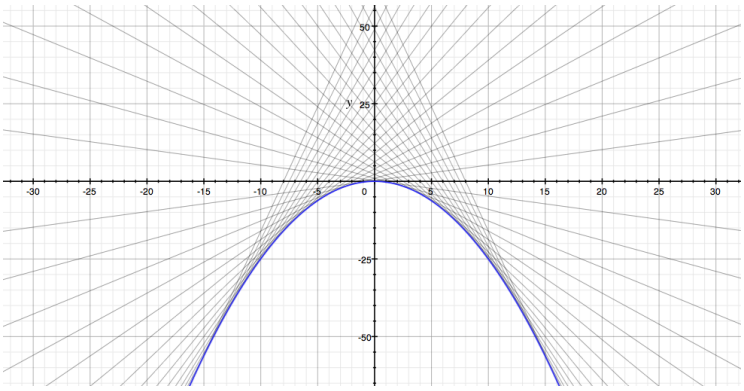


Figure 10.1: Representation[15] of the solutions of the equation $y = xy' + (y')^2$.

We can see from Figure 10.1 that if we choose a point (x_0, y_0) under the singular curve, there is no solution passing through this point. On the other hand, we observe that the singular parabola and the line $y = xp + p^2$ intercept at the point $(-2p, -p^2)$, so for example the curves

$$\varphi_p(x) = \begin{cases} xp + p^2, & x < -2p, \\ \dfrac{-x^2}{4} & x \geq -2p \end{cases}$$

are also solutions of the equation for every choice of $p \in \mathbb{R}$.

In a completely analogous way, the general solution of $y = xy' + (y')^3$ is the family of lines $y = xp + p^3$, $p \in \mathbb{R}$, and the singular solution is the curve given implicitly by the equation

$$y^2 = \frac{-4x^3}{27}, \quad x \leq 0,$$

as shown in Figure 10.2. Unlike the previous example, in this case there is at least one solution passing through every point of the (x, y)-plane.

[15] Both this figure and the next one were created by Alex Hall and are freely distributed on Wikimedia Commons under a CC-BY-SA 3.0 license.

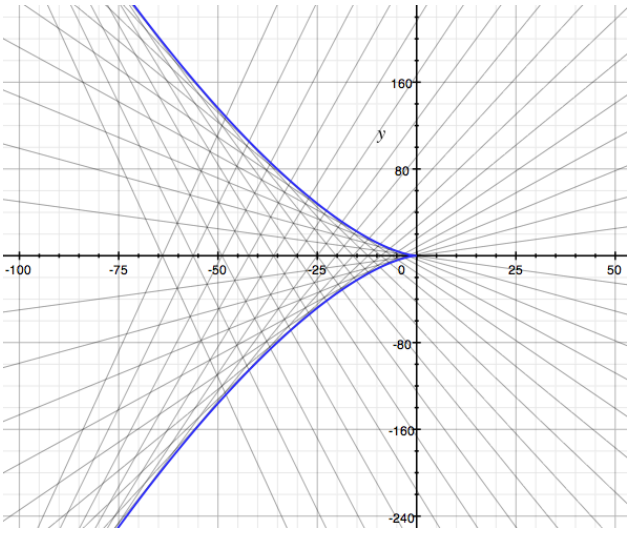


Figure 10.2: Representation of the solutions of the equation $y = xy' + (y')^3$.

10.2 Lagrange equation

The next equation we consider is the family of Lagrange equations,

$$y = xf(y') + g(y'), \tag{10.5}$$

where both f and g are defined and continuously differentiable in some open interval $I \subset \mathbb{R}$. These equations are named after Joseph Louis Lagrange[16], who also studied the theory of implicit differential equations and singular solutions in the second half of the 18th century, and discussed this specific equation in [Lag59]. One thing to note from the start is that when f is the identity function, equation (10.5) reduces to a Clairaut equation, so in some sense this new equation is a generalization of the former.

To solve this equation the most straightforward method is to introduce the parameter $p = dy/dx$ and differentiate the equation with respect to x, obtaining

$$p = f(p) + xf'(p)p' + g'(p)p' \implies p - f(p) = (xf'(p) + g'(p))p'. \tag{10.6}$$

Now we consider firstly the case when $p' = 0$, that is, $p = y' = C \in I$. Equation (10.6) implies that $p = f(p)$, and substituting this information back into (10.5) gives us that the straight line $y = xC + g(C)$ is a solution of the equation. Thus, the family

$$\{y = xC + g(C) : C \text{ is a root of } p - f(p) = 0\}$$

is a family of solutions of the equation, which might or might not be singular. It can also be shown that these are the only straight lines that are solutions of the equation.

[16] They are also associated with d'Alembert, so in some texts they are referred to as d'Alembert equations.

Assuming that $p \neq f(p)$, equation (10.6) implies that $dp/dx \neq 0$, and then we can apply the inverse function theorem to write $x = x(p)$. In this case, equation (10.6) reads

$$\frac{dx}{dp} = \frac{xf'(p) + g'(p)}{p - f(p)}. \tag{10.7}$$

This is a linear differential equation in the independent variable p, which will have a general solution, say $x = \Phi(p, C)$. Substituting back into the original equation, we have found the family of general solutions of equation (10.5), which is expressed as a function of the parameter p as

$$x = \Phi(p, C), \quad y = \Phi(p, C)f(p) + g(p).$$

Example 10.2. Let us consider the Lagrange equation

$$y = 2xy' - (y')^2. \tag{10.8}$$

We identify the functions $f(p) = 2p$ and $g(p) = -p^2$. Solving $f(p) = p$ for p yields $p = 0$, so the only solution line is that of slope 0 passing through the point $(0, g(0)) = (0, 0)$, that is, the horizontal line $y = 0$. If we analyzed the criminant of this equation, we would realize that it is composed of the points on the solution surface for which $x = p$, so the line $y = 0$ is not a singular solution of the equation.

If $p' \neq 0$, setting $y' = p$ in the equation and differentiating with respect to x yields

$$-p = p'(2x - 2p),$$

which since $p' \neq 0$ we can rewrite as the linear ODE

$$\frac{dx}{dp} = \frac{-2}{p}x + 2.$$

This equation gives the general solution $x(p) = \frac{2}{3}p + \frac{C}{p^2}$, which we can substitute in our equation to get the family of general solutions of (10.8), depicted in Figure 10.3:

$$\begin{cases} x = \dfrac{2}{3}p + \dfrac{C}{p^2}, \\ y = \dfrac{1}{3}p^2 + \dfrac{2C}{p}, \end{cases} \qquad p \neq 0.$$

10.3 Chrystal equation

Lastly we present an equation studied at the end of the 19th century by G. Chrystal, who took a closer look at its potential singular solutions (see [Chr97] for the original article). A Chrystal equation assumes the form

$$(y')^2 + Axy' + By + Cx^2 = 0, \tag{10.9}$$

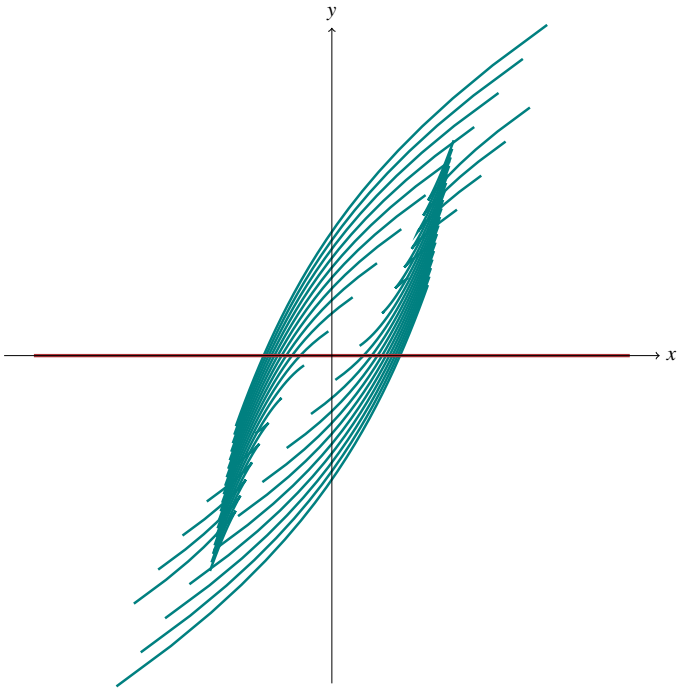


Figure 10.3: Representation of the family of solutions of the equation $y = 2xy' - (y')^2$.

where A, B, C are real-valued constants. As we will see below, under certain conditions this equation is yet another generalization of the Clairaut equation, which stresses the importance of the latter in the theory of implicit equations throughout the modern history of mathematics. The Chrystal equation is discussed in some detail in [Inc56] and [Dav62].

The solving process for this equation begins by solving (10.9) for y' and considering at the same time the two resulting differential equations, that is,

$$y' = -\frac{A}{2}x \pm \frac{1}{2}(A^2x^2 - 4By - 4Cx^2)^{1/2}. \tag{10.10}$$

Now we introduce a new variable $z = z(x)$ by means of the relation

$$4By = (A^2 - 4C - z^2)x^2, \tag{10.11}$$

so that equation (10.10) reduces to

$$xzz' = A^2 + AB - 4C - z^2 \pm Bz.$$

This equation is separable and can be written as

$$\frac{dx}{x} = \frac{z\,dz}{A^2 + AB - 4C - z^2 \pm Bz}. \tag{10.12}$$

The next step is to factorize the denominator in the right hand side of (10.12). If we denote by a and b the two combined roots of the equation

$$A^2 + AB - 4C - z^2 \pm Bz = 0, \tag{10.13}$$

we can rewrite equation (10.12) in the form

$$\frac{dx}{x} = \frac{z\,dz}{(z-a)(z-b)}. \tag{10.14}$$

It can be proven (via direct integration) that if $a \neq b$, the solutions of this equation take the form

$$x\frac{(z-a)^{a/(a-b)}}{(z-b)^{b/(a-b)}} = k, \quad k \in \mathbb{R},$$

and if $a = b$ then the solutions[17] are

$$x(z-a)\exp\left(\frac{a}{a-z}\right) = k, \quad k \in \mathbb{R}.$$

In each case we can substitute back into (10.11) using these expressions, and hence we get the solutions to our equation in parametric form, where now z is the parameter.

Apart from this general procedure, the case when one of the roots a or b is 0 merits special attention. Suppose without loss of generality that $a = 0$. Then by equation (10.13) we have

$$A^2 + AB - 4C = 0, \tag{10.15}$$

so the other root is $b = \pm B$, which we will suppose is non-zero for the sake of the argument. In this case the solutions to (10.14) are expressed as

$$x(z \pm B) = k, \quad k \in \mathbb{R},$$

and then we assert that the solutions of (10.9) are recovered as

$$4By = -ABx^2 - (k \pm Bx)^2, \quad k \in \mathbb{R}. \tag{10.16}$$

To see this, note that from $x(z \pm B) = k$ we get $xz = k \pm Bx$, and substituting this expression into (10.11) yields

$$4By = x^2(A^2 - 4C) - (k \pm Bx)^2.$$

Finally, equation (10.15) allows us to write $A^2 - 4C = -AB$, arriving at the desired result.

It is worth pointing out that the family of parabolas we get as solutions in this case has an envelope, namely

$$y = -\frac{A}{4}x^2.$$

[17] A more profound analysis of the transcendental solutions that arise in this case can be found in [Jor10].

This expression can be derived employing the usual techniques, and it can be seen that this parabola is tangent to the family (10.16) at each point, so it is indeed its envelope. In other words, when condition (10.15) holds we have a singular solution of the equation. If we take a closer look at the process this fact should not come as a surprise, since under the conditions mentioned equation (10.11) reduces to the Clairaut form.

11 Conclusions and future work

Throughout this document we have presented the theory of first order implicit differential equations, exploring the classical texts related to them and establishing general conditions for their treatment. We feel that our initial goals have been accomplished, since a fairly complete study has been developed. In addition, some interesting examples have been presented, which illustrate how this theory can be applied to solve specific equations in a satisfactory way.

The theory of differential equations not solved for the derivative goes back centuries, to the time of Newton and Leibniz, but the written materials discussing them in modern terms are scarce. A significant contribution of our work has been to put together those resources and unify them in a complete theory that tackles the principal aspects of these equations. When possible, we have included the references of the original books and articles where a certain technique or equation first appeared, and in doing so we have obtained a collection of sources in which a treatment of this type of equations is considered, which can be useful for further research.

Another consequence of the unification mentioned earlier is that different points of view are presented when it comes to dealing with implicit equations. In the first part of the work we explored a more algebraic approach to solving these equations, while later we moved on to a very geometrical interpretation of them. In the latter approach a significant amount of concepts from differential geometry and the geometry of surfaces are studied as tools for modelling implicit equations, and they can be complex in the sense that they require a firm understanding of the underlying theory and its implications. We paid special attention to how the definitions and results were stated so that they would be consistent with every part of the theory developed, which again was not an easy task due to the lack of a consolidated work that served as a reference for every part. A relevant example of this situation are singular solutions, as their precise definition varies from author to author.

In regard to future areas of research and extensions of this work, several paths are possible. Firstly, we could look at the generalization of the theory to second order implicit ODEs, which are usually utilized to model and describe the behaviour of many systems in physics and other sciences. A good reference for this would be [Tak07].

Secondly, we could delve deeper into the matter of singular solutions, for example studying the behaviour of the equation near a singular point, distinguishing whether or not there is a whole neighbourhood of singular points, as done in [Rab89].

Lastly, we could also study other classical examples of equations not solved for the derivative, analyzing the changes of variables needed to efficiently solve them or even provide a strategy to tackle equations in specific forms, such as $y' = f(y, y')$ or $F(y, y') = 0$. It is worth noting that many first order implicit equations arise in the field of calculus of variations when looking for critical points of functionals of functions of one variable, since normally the necessary conditions imposed by the Euler-Lagrange equations result in an equation in implicit form. This is the case, for example, of the problem of finding the curve of fastest descent, the *brachistochrone*, for which the Euler-Lagrange equation takes the form $y(1 + (y')^2) = C^2$.

In conclusion, a sufficiently deep study of implicit differential equations has been carried out, focusing mostly on the theoretical aspects but also providing relevant examples that have been largely studied by the mathematical community. Furthermore, this theory can serve as a starting point for further research on this topic, from which many practical applications are sure to emanate as well.

Acronyms

COA Centroid of Area. 22
FCM Fuzzy C-Means. 39
FIS Fuzzy Inference System. 20
FRBS Fuzzy Rule-Based System. 21
IVP Initial Value Problem. 71
MF Membership Function. 7
MSE Mean Squared Error. 48
ODE Ordinary Differential Equation. 65
RDD Resilient Distributed Dataset. 27

Acknowledgements

I would like to thank my supervisors Margarita Arias, José Manuel Benítez and Miguel Lastra for their assistance, advice and guidance while developing this work. Their explanations and corrections on both the content and the style of this document have helped me greatly and are very much appreciated.

I would also like to thank my family and friends for supporting me every step of the way, especially since this work was developed during the difficult times of the COVID-19 pandemic. I am grateful to everyone who read drafts of this document and provided valuable feedback.

Finally, I would like to extend my gratitude to the whole scientific community, and in particular to Alexandra Elbakyan, for believing in open science and contributing to making it more accessible for everyone.

Bibliography

[Ros58] F. Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386 (cit. on p. 3).

[IIL+67] A. Ivakhnenko, A. Ivakhnenko, V. Lapa, V. Lapa, V. LAPA, R. McDonough. *Cybernetics and Forecasting Techniques*. American Elsevier Publishing Company, 1967 (cit. on p. 3).

[Iva71] A. G. Ivakhnenko. “Polynomial Theory of Complex Systems”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-1.4 (1971), pp. 364–378 (cit. on p. 3).

[WXP17] H. Wang, Z. Xu, W. Pedrycz. “An overview on the roles of fuzzy set techniques in big data processing: Trends, challenges and opportunities”. In: *Knowledge-Based Systems* 118 (2017), pp. 15–30. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2016.11.008 (cit. on p. 4).

[FCDH16] A. Fernández, C. J. Carmona, M. J. Del Jesús, F. Herrera. “A View on Fuzzy Systems for Big Data: Progress and Opportunities”. In: *International Journal of Computational Intelligence Systems* 9 (May 2016), pp. 69–80. DOI: 10.1080/18756891.2016.1180820 (cit. on p. 4).

[Zad65] L. Zadeh. “Fuzzy sets”. In: *Information and Control* 8.3 (1965), pp. 338–353. DOI: https://doi.org/10.1016/S0019-9958(65)90241-X (cit. on p. 7).

[CP00] G. Chen, T. Pham. *Introduction to Fuzzy Sets, Fuzzy Logic, and Fuzzy Control Systems*. CRC Press, 2000. ISBN: 9781420039818 (cit. on p. 7).

[JSM97] J. Jang, C. Sun, E. Mizutani. *Neuro-fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. Prentice Hall, 1997. ISBN: 9780132610667 (cit. on p. 7).

[Ker11] E. Kerre. “A tribute to Zadeh’s extension principle”. In: *Scientia Iranica* 18.3 (2011), pp. 593–595 (cit. on p. 11).

[Mar04] A. V. Markovskii. “Solution of Fuzzy Equations with Max-Product Composition in Inverse Control and Decision Making Problems”. In: *Autom. Remote Control* 65.9 (Sept. 2004), pp. 1486–1495. DOI: 10.1023/B:AURC.0000041426.51975.50 (cit. on p. 12).

[LF01] J. Loetamonphong, S.-C. Fang. “Optimization of fuzzy relation equations with max-product composition”. In: *Fuzzy Sets and Systems* 118.3 (2001), pp. 509–517. DOI: https://doi.org/10.1016/S0165-0114(98)00417-5 (cit. on p. 12).

[Sug93] M. Sugeno. “Fuzzy measures and fuzzy integrals—a survey”. In: *Readings in fuzzy sets for intelligent systems*. Elsevier, 1993, pp. 251–257 (cit. on p. 15).

[Yag79] R. R. Yager. “On the measure of fuzziness and negation part I: membership in the unit interval”. In: (1979) (cit. on p. 15).

[MA75] E. Mamdani, S. Assilian. “An experiment in linguistic synthesis with a fuzzy logic controller”. In: *International Journal of Man-Machine Studies* 7.1 (1975), pp. 1–13. DOI: https://doi.org/10.1016/S0020-7373(75)80002-2 (cit. on p. 21).

[LK99] W. V. Leekwijck, E. E. Kerre. “Defuzzification: criteria and classification”. In: *Fuzzy Sets and Systems* 108.2 (1999), pp. 159–178. DOI: https://doi.org/10.1016/S0165-0114(97)00337-0 (cit. on p. 22).

[TS85] T. Takagi, M. Sugeno. “Fuzzy identification of systems and its applications to modeling and control”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-15.1 (1985), pp. 116–132 (cit. on p. 22).

[SK88] M. Sugeno, G. Kang. “Structure identification of fuzzy model”. In: *Fuzzy Sets and Systems* 28.1 (1988), pp. 15–33. DOI: https://doi.org/10.1016/0165-0114(88)90113-3 (cit. on p. 22).

[RRG18] J. Rydning, D. Reinsel, J. Gantz. “The digitization of the world from edge to core”. In: (2018) (cit. on p. 23).

[KM16] R. Kitchin, G. McArdle. “What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets”. In: *Big Data & Society* 3.1 (2016), p. 2053951716631130. DOI: 10.1177/2053951716631130 (cit. on p. 23).

[EU16] “Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)”. In: *OJ* L 119 (4-05-2016), pp. 1–88. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679 (cit. on p. 25).

[CKRS10] R. Chansler, H. Kuang, S. Radia, K. Shvachko. “The Hadoop Distributed File System”. In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST 2010)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2010, pp. 1–10. DOI: 10.1109/MSST.2010.5496972 (cit. on p. 26).

[DG04] J. Dean, S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. San Francisco, CA, 2004, pp. 137–150 (cit. on p. 26).

[Gon19] J. González López. "Distributed multi-label learning on Apache Spark". PhD thesis. Virginia Commonwealth University and University of Córdoba, 2019. DOI: https://doi.org/10.25772/HGBD-P556 (cit. on p. 28).

[Sue12] J. Suereth. *Scala in Depth*. Manning Publications, 2012. ISBN: 9781935182702 (cit. on p. 29).

[OSV08] M. Odersky, L. Spoon, B. Venners. *Programming in Scala*. Artima, 2008. ISBN: 9780981531601 (cit. on p. 29).

[KKWZ15] H. Karau, A. Konwinski, P. Wendell, M. Zaharia. *Learning Spark: Lightning-Fast Big Data Analysis*. O'Reilly Media, 2015. ISBN: 9781449359058 (cit. on p. 29).

[RBHB15] L. S. Riza, C. Bergmeir, F. Herrera, J. M. Benítez. "frbs: Fuzzy Rule-Based Systems for Classification and Regression in R". In: *Journal of Statistical Software* 65.6 (2015), pp. 1–30. URL: http://www.jstatsoft.org/v65/i06/ (cit. on p. 30).

[LBH15] Y. LeCun, Y. Bengio, G. Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444 (cit. on p. 30).

[KK99] J. Kim, N. Kasabov. "HyFIS: adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems". In: *Neural networks: the official journal of the International Neural Network Society* 12.9 (1999), pp. 1301–1319. DOI: 10.1016/s0893-6080(99)00067-2 (cit. on p. 31).

[Cor11] O. Cordón. "A historical review of evolutionary learning methods for Mamdani-type fuzzy rule-based systems: Designing interpretable genetic fuzzy systems". In: *International journal of approximate reasoning* 52.6 (2011), pp. 894–913 (cit. on p. 31).

[CHHM01] O. Cordon, F. Herrera, F. Hoffmann, L. Magdalena. *Genetic Fuzzy Systems: Evolutionary Tuning And Learning Of Fuzzy Knowledge Bases*. Advances In Fuzzy Systems-applications And Theory. World Scientific Publishing Company, 2001. ISBN: 9789814494458 (cit. on p. 31).

[CHV01] O. Cordón, F. Herrera, P. Villar. "Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base". In: *IEEE Transactions on fuzzy systems* 9.4 (2001), pp. 667–674 (cit. on p. 31).

[Thr91] P. Thrift. "Fuzzy Logic Synthesis with Genetic Algorithms". In: *ICGA*. 1991 (cit. on p. 31).

[CH97] O. Cordón, F. Herrera. “A three-stage evolutionary process for learning descriptive and approximate fuzzy-logic-controller knowledge bases from examples”. In: *International Journal of Approximate Reasoning* 17.4 (1997), pp. 369–407 (cit. on p. 31).

[WM92] L.-X. Wang, J. M. Mendel. “Generating fuzzy rules by learning from examples”. In: *IEEE Transactions on systems, man, and cybernetics* 22.6 (1992), pp. 1414–1427 (cit. on pp. 32, 33).

[CYP96] Z. Chi, H. Yan, T. Pham. *Fuzzy Algorithms: With Applications to Image Processing and Pattern Recognition*. Advances in fuzzy systems - applications and theory. World Scientific, 1996. ISBN: 9789810226978 (cit. on p. 34).

[ÁM18] D. Álvarez-Estévez, V. Moret-Bonillo. “Revisiting the Wang–Mendel algorithm for fuzzy classification”. In: *Expert Systems* (Feb. 2018). DOI: 10.1111/exsy.12268 (cit. on p. 34).

[Chi94] S. Chiu. “Fuzzy Model Identification Based on Cluster Estimation”. In: *Journal of the Intelligent and Fuzzy Systems* 2 (Jan. 1994), pp. 267–278. DOI: 10.3233/IFS-1994-2306 (cit. on p. 35).

[YF94] R. R. Yager, D. P. Filev. “Approximate clustering via the mountain method”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 24.8 (1994), pp. 1279–1284 (cit. on p. 35).

[Pha01] T. D. Pham. “Image segmentation using probabilistic fuzzy c-means clustering”. In: *Proceedings 2001 International Conference on Image Processing*. Vol. 1. 2001, 722–725 vol.1 (cit. on p. 40).

[AV06] D. Arthur, S. Vassilvitskii. *k-means++: The advantages of careful seeding*. Tech. rep. 2006 (cit. on p. 43).

[BMV+12] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, S. Vassilvitskii. “Scalable k-means++”. In: *arXiv preprint arXiv:1203.6402* (2012) (cit. on p. 43).

[BSW14] P. Baldi, P. Sadowski, D. Whiteson. “Searching for exotic particles in high-energy physics with deep learning”. In: *Nature communications* 5.1 (2014), pp. 1–9 (cit. on p. 46).

[AAC+06] R. Alcalá, J. Alcalá-Fdez, J. Casillas, O. Cordón, F. Herrera. “Hybrid learning models to get the interpretability-accuracy trade-off in fuzzy modeling”. In: *Soft Computing* 10 (July 2006), pp. 717–734. DOI: 10.1007/s00500-005-0002-1 (cit. on p. 60).

[GAH11] M. J. Gacto, R. Alcalá, F. Herrera. “Interpretability of linguistic fuzzy rule-based systems: An overview of interpretability measures”. In: *Information Sciences* 181.20 (2011), pp. 4340–4360 (cit. on p. 60).

[AFG04] T. Archibald, C. Fraser, I. Grattan-Guinness. “The History of Differential Equations, 1670–1950”. In: *Oberwolfach Reports* (Jan. 2004), pp. 2729–2794. DOI: 10.4171/OWR/2004/51 (cit. on p. 63).

[Arn12] V. Arnold. *Geometrical Methods in the Theory of Ordinary Differential Equations*. Springer New York, 2012. ISBN: 9781461210375 (cit. on pp. 63, 73, 81, 85).

[Arn92] V. I. Arnold. *Ordinary Differential Equations*. Springer Textbook. Springer Berlin Heidelberg, 1992. ISBN: 9783540548133 (cit. on pp. 63, 69).

[Pet66] I. G. Petrovsky. *Ordinary Differential Equations*. Prentice-Hall, 1966 (cit. on pp. 63, 65, 73, 76).

[Apo74] T. M. Apostol. *Mathematical Analysis, Second Edition*. Addison-Wesley, 1974 (cit. on pp. 69, 70, 73).

[Pea90] G. Peano. “Démonstration de l’intégrabilité des équations différentielles ordinaires”. In: *Mathematische Annalen* 37 (1890), pp. 182–228. DOI: 10.1007/BF01200235 (cit. on p. 71).

[Hal80] J. K. Hale. *Ordinary Differential Equations*. 2nd ed. Krieger, 1980 (cit. on p. 71).

[Tes12] G. Teschl. *Ordinary differential equations and dynamical systems*. Vol. 140. American Mathematical Soc., 2012 (cit. on p. 72).

[Ban22] S. Banach. “Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales”. In: *Fundamenta Mathematicae* 3 (1922), pp. 133–181. DOI: 10.4064/fm-3-1-133-181 (cit. on p. 72).

[MRB09] S. Montiel, A. Ros, D. Babbitt. *Curves and Surfaces*. Graduate studies in mathematics. American Mathematical Society, 2009. ISBN: 9780821847633 (cit. on p. 80).

[Cib32] M. Cibrario. “Sulla reduzione a forma canonica delle equazioni lineari alle derivative parzialy di secondo ordine di tipo misto”. In: *Rendiconti Instituto Lombardo* 65 (1932), pp. 889–906 (cit. on p. 85).

[Whi55] H. Whitney. “On singularities of mappings of Euclidean spaces. I. Mappings of the plane into the plane”. In: *Annals of Mathematics* (1955), pp. 374–410 (cit. on p. 85).

[CR14] R. Chertovskih, A. Remizov. “On pleated singular points of first-order implicit differential equations”. In: *Journal of Dynamical and Control Systems* 20.2 (2014), pp. 197–206 (cit. on p. 85).

[Dar75] L. Dara. “Singularites generiques des equations differentielles multiformes”. In: *Boletim da Sociedade Brasileira de Matemática - Bulletin/Brazilian Mathematical Society* 6 (1975), pp. 95–128 (cit. on pp. 85, 91).

[Dar73] G. Darboux. “Sur les solutions singulières des équations aux dérivées ordinaires du premier ordre”. In: *Bulletin des sciences mathématiques et astronomiques* 4 (1873), pp. 158–176 (cit. on p. 86).

[Lan61] H. B. Lane. “On Envelopes and Extraneous Loci of Differential Equations of Order One and Higher Degree”. PhD thesis. University of North Texas, 1961. URL: https://digital.library.unt.edu/ark:/67531/metadc130469/m2/1/high_res_d/n_02827.pdf (cit. on p. 87).

[Lag59] J.-L. Lagrange. “Sur l’intégration d’une équation différentielle à différences finies, qui contient la théorie des suites récurrentes”. In: *Miscellanea Taurinensia* 1 (1759), pp. 33–42 (cit. on p. 94).

[Chr97] G. Chrystal. “On the p-discriminant of a Differential Equation of the First Order, and on Certain Points in the General Theory of Envelopes connected therewith”. In: *Transactions of the Royal Society of Edinburgh* 38.4 (1897), pp. 803–824. DOI: 10.1017/S0080456800033494 (cit. on p. 95).

[Inc56] E. Ince. *Ordinary Differential Equations*. Dover Books on Mathematics. Dover Publications, 1956. ISBN: 9780486603490 (cit. on p. 96).

[Dav62] H. Davis. *Introduction to Nonlinear Differential and Integral Equations*. Dover books on advanced mathematics. Dover Publications, 1962. ISBN: 9780486609713 (cit. on p. 96).

[Jor10] P. Jordan. “A note on Chrystal’s equation”. In: *Applied mathematics and computation* 217.2 (2010), pp. 933–936 (cit. on p. 97).

[Tak07] M. Takahashi. “On Implicit Second-Order Ordinary Differential Equations: Completely Integrable and Clairaut Type”. In: *Journal of Dynamical and Control Systems* 13 (2007). DOI: 10.1007/s10883-007-9013-9 (cit. on p. 99).

[Rab89] P. J. Rabier. “Implicit differential equations near a singular point”. In: *Journal of Mathematical Analysis and Applications* 144.2 (1989), pp. 425–449. DOI: https://doi.org/10.1016/0022-247X(89)90344-2 (cit. on p. 99).

All links were last followed on September 7, 2020.

A Cost estimation and planning

Since as part of this bachelor's thesis we were to undertake a software development project, we prepared an estimation of the monetary costs we would incur, including the labour costs. We established an hourly price of 30€, and we divided the work in three different aspects: the theoretical study needed to understand the algorithms, the implementation itself, and the composition of the documentation describing the design and the results obtained. We also added a category for the cost of the infrastructure used to deploy our algorithms. The result of this cost estimation can be seen in Table A.1.

Table A.1: Cost estimation for our software development project.

Concept	Amount (hours)	Unitary price (€/h)	Total cost (€)
Study of fuzzy theory	20	30	600
Study of the MapReduce model	15	30	450
Learning Scala and Spark	40	30	1200
Design of algorithms	50	30	1500
Implementation of algorithms	30	30	900
Testing of algorithms	15	30	450
Comparative study	5	30	150
Documentation writing	20	30	600
Corrections	5	30	150
Cluster (40 CPUs)	–	–	10000
Cluster installation	–	–	500
Cluster maintenance	–	–	500
Total	200	–	17000

Apart from the cost estimation, we also made a temporal planning of the project, starting on September 2019 and with an expected conclusion date at the end of June 2020, as we can see in Figure A.1. The examination and holidays periods are accounted for.

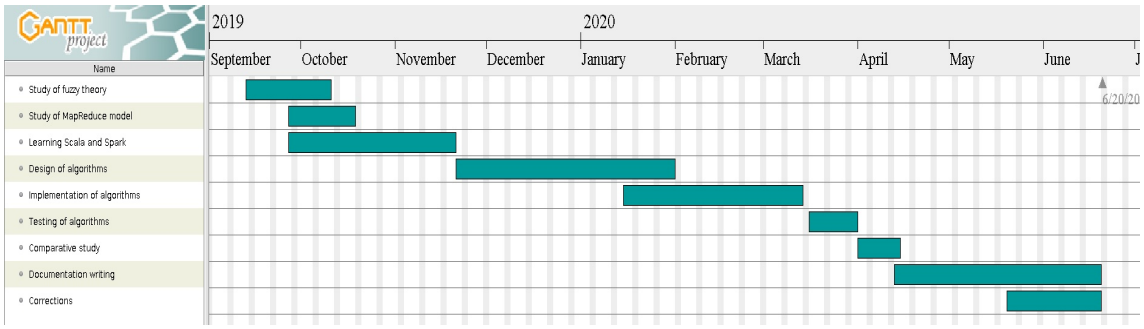


Figure A.1: Temporal planning of our software development project.

However, this planning turned out to be overly optimistic. We spent more time than planned in the implementation and testing of the algorithms, mainly because some unexpected difficulties arose with Spark and the way it handled some implementation details. Because of this, we had to go back and re-design some algorithms so as to avoid these obstacles. Furthermore, while conducting these tasks we entered in a worldwide crisis situation regarding the COVID-19 pandemic, so as a result the work was slower and even stopped for a short period. A more realistic estimation can be seen in Figure A.2.

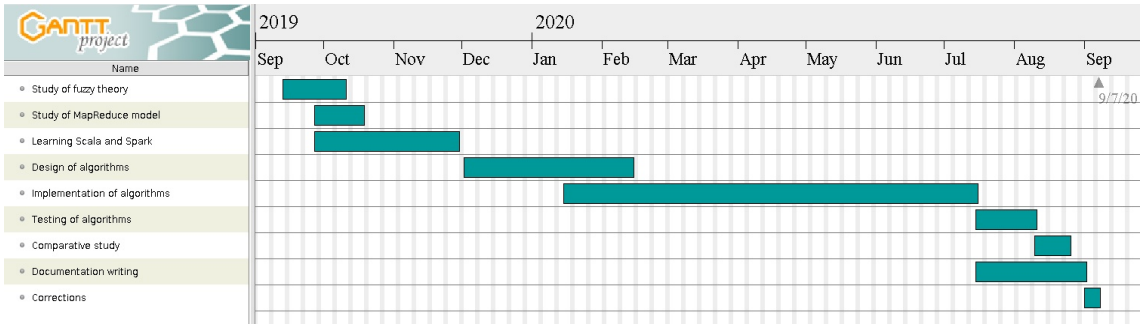


Figure A.2: Realistic temporal planning of our software development project.

B About the software developed

All the code developed for this bachelor's thesis is freely available at `https://github.com/antcc/fuzzyspark` under a GPLv3 license. It consists roughly on 2200 lines of code written in Scala 2.11.8 and Spark 2.3.2. Every function, class and method is properly documented in the source code.

The algorithms developed come bundled as a package, called *fuzzyspark*, which consists on a subpackage for the clustering algorithms and another one for more generic FRBSs. In addition, a couple of separate testing files are provided with the code used to perform the comparative study on Chapter 5. The project can be compiled using Scala's interactive building tool, *sbt*, for which two build files are provided (one for the algorithms suite and one for the testing files). Since the package developed is not published anywhere, some additional steps have to be taken:

1. In the first place, the package *fuzzyspark* needs to be published to the local repository, executing the order `sbt publishLocal` inside the *fuzzyspark* folder. Another alternative is to manually package it into a JAR file and add it as an unmanaged dependency.
2. In the second place, the resulting package needs to be bundled with the testing application as a dependency, for which we can use the *assembly* plugin, executing the order `sbt assembly` inside the *testing* directory.

Following the above steps we end up with a JAR file ready to be executed in Spark. For this execution we use the script *spark-submit* that comes with the installation. In particular, a JAR file containing Spark code can be executed with the following line, where N, M and C are adjustable parameters used to specify the configuration of the YARN cluster:

```
spark-submit --class MAIN_CLASS --num-executors N --executor-cores C \
  --executor-memory M [JAR_FILE] [ARGS]
```

Regarding the file structure, two source files have been developed for each algorithm: one which contains the learning phase and methods to fit the model to the data, and another one which contains the trained model and implements the prediction phase. Specifically, they are:

- `FuzzyCMeans.scala` and `FuzzyCMeansModel.scala` for the Fuzzy C-Means algorithm.

- `WM.scala` and `WMModel.scala` for the Wang and Mendel algorithm.
- `SubtractiveClustering.scala` and `ModelIdentification.scala` for the subtractive clustering algorithm and its variants.

As we mentioned earlier, there are two additional files to test the implementations; they are called `ClusteringTest.scala` and `ModelTest.scala`. There is also another file, `Utils.scala`, that comprises some utility methods.

No external libraries apart from Spark were used in the code. The principal data structures used are of course Spark's RDDs, and each data point was modelled as a `Vector` of floating point values.

Lessons learned

Finally, we summarize some of the difficulties that arose in the programming phase, especially those having to do with how Spark works internally. We found that iterative algorithms are still a challenge to be properly implemented in Spark. For example, we observed that when we modified an RDD on each iteration and the number of iterations grew large, a special operation called a *checkpoint* needed to be performed in order to restart the *lineage* of the RDD (that is, the history of the sequence of transformations executed upon it), or else this history would grow so big that memory problems would start to appear. Also, the data should always be cached so that it is kept in memory between iterations to increase performance.

In some cases we also needed to tune a few of Spark internal parameters, such as the garbage collector or the maximum response time for the worker nodes, to prevent some errors in the execution. In addition, the number of partitions was also an essential factor that greatly influenced execution time, and finding a suitable value for each data set was a challenging task.