POLITECNICO DI MILANO

School of Industrial and Information Engineering
Computer Science and Engineering (CSE) - Master Degree
Department of Electronics, Information and Bio-engineering (DEIB)

POLITECNICO
MILANO 1863

Dealing with Uncertainties in Availability Models using Fuzzy Logic and Neural Networks

Supervisor: *Raffaela Mirandola*

Co-supervisor: *Diego Pérez Palacín* (Affiliation: Linnaeus University, Sweden)

Student: *Francesco Marchesani* (Mat. 852444)

Academic Year: 2016/2017

A chi dà spazio alle proprie idee,

a chi ha ancora il coraggio di inseguire i propri sogni.

A chi riesce a far tesoro di un abbraccio,

di una frase, di uno sguardo.

A chi sa essere riconoscente, a chi sa donare un sorriso,

a chi dà sempre una mano senza chiedere nulla in cambio.

A chi lotta ogni giorno per un nobile ideale,

a chi riesce a pensare con la propria testa.

A chi è fiero di essere l'eccezione e non la regola.

A chi ama l'arte, la musica, la scienza, la filosofia,

a chi riconosce la straordinaria forza delle parole.

A chi è caduto e, nonostante le ferite,

si è rialzato ed ora cammina più forte di prima.

Ai miracoli del quotidiano,

alle piccole cose che sono il segreto della felicità.

A chi, nonostante tutto,

crede ancora nell'amore,

nelle favole

e nella poesia.

A. Ringraziamenti (Greetings)

Un primo, doveroso, ringraziamento lo devo alla Professoressa Raffaela Mirandola e all'Ingegnere Diego Pérez Palacín, che ho avuto modo di conoscere durante il mio percorso universitario e che hanno creduto in me e nelle mie capacità, sempre con la massima gentilezza e professionalità.

Ringrazio poi la mia famiglia, a partire dai miei straordinari genitori: Celestino e Maria. Cinque anni or sono mi hanno dato l'opportunità concreta di investire sul mio futuro, con l'iscrizione presso il *Politecnico di Milano*. Io ho cercato di fare sempre del mio meglio, con la massima serietà… e le cose sono andate per il verso giusto! Un saluto ed un abbraccio anche alle mie sorelline, Paola e Miriam. Un ringraziamento sentito anche agli altri parenti, soprattutto a coloro che mi sono stati più vicini in questi anni. Zia Tina e zio Tonino, assieme ai miei cugini Stefano e Daniele, sono stati una seconda famiglia per me. Per quanto riguarda gli altri zii e parenti, vicini e lontani: grazie! In qualche modo ognuno di voi ha avuto un contributo nella mia formazione e nel mio percorso di crescita interiore. Prima di chiudere questa sezione dei ringraziamenti ai parenti, vorrei fare una dedica al mio nonno ultranovantenne: nonno Antonio. Dovevamo farcela e ce l'abbiamo fatta, nonno!

E agli amici? Ragazzi miei, non potevo non pensare anche a voi! Io ho avuto la fortuna di conoscere tante persone stupende nel corso della mia vita, sia nel mio paese d'origine che a Milano. Purtroppo qui sussiste il problema della vastità delle persone da nominare. Mi limiterò dunque a promulgare un ringraziamento generale, chi si riconoscerà in queste parole sarà certamente tra i vari amici a cui sto pensando mentre scrivo questi ringraziamenti. Un grazie dunque a chi mi è stato vicino da sempre, a chi non mi ha mai voltato le spalle, a chi è stato presente tanto nei momenti di gioia, quanto in quelli di dolore. Un grazie agli amici di università, che mi hanno supportato (e talvolta sopportato) durante questi cinque anni. Un grazie agli amici storici, quelli che fanno parte dell'ossatura delle mie relazioni. Un grazie agli eroi e soprattutto agli anti-eroi, che in un modo o nell'altro hanno condizionato la mia vita. Grazie ancora ai miei compagni di ventura e di sventura, a quelli che hanno condiviso con me momenti di ogni tipo: feste, passeggiate, momenti di musica e di sport, oltre alle avventure più disparate. Siete tutti nella mia mente e, statene certi, ancora di più nel mio cuore.

Grazie a chi c'era ed ora non c'è più.

Perché badate bene, la vita funziona così e dobbiamo accettarlo.

Ma se abbiamo vissuto appieno i momenti passati, non c'è nessun rimpianto da serbare. Penso ai miei nonni defunti (persone semplici, oneste, virtuose!), ma anche ad altre persone che per un motivo o per l'altro sono uscite dalla mia vita. Sarebbe bello avere anche loro qui, al mio fianco. Ma "*C'est la vie*," come diceva un certo Chuck Berry in una delle sue più celebri canzoni: *You Never Can Tell*. E noi questa vita la viviamo così… 'come viene'.

Grazie infine a tutte le altre persone che non ho citato esplicitamente, ma che mi hanno dato importanti insegnamenti di vita anche al di fuori del mondo strettamente accademico. Mi avete aiutato a crescere, giorno dopo giorno.

Grazie a tutti.

B. Table of Contents

C. List of Figures

D. List of Acronyms and Abbreviations

The following list contains the complete collection of *acronyms* and *abbreviations* that we are going to encounter in this master thesis, in alphabetic order.

- **ANN** = Artificial Neural Network
- **API** = Application Programming Interface
- **CC** = Cloud Computing
- **CNN** = Convolutional Neural Network
- **CWW** = Computing With Words
- **D&C** = Divide and Conquer
- **DL** = Deep Learning
- **DNN** = Deep Neural Network
- **DNS** = Domain Name System
- **DST** = Dempster-Shafer Theory
- **EEM** = Elementary Effect Method
- **EU** = Expected Utility
- **EUT** = Expected Utility Theory
- **FAST** = Fourier Amplitude Sensitivity Test
- **FIS** = Fuzzy Inference System
- **FL** = Fuzzy Logic
- **FSM** = Finite State Machine
- **FTDNN** = Focused Time-Delay Neural Network
- **GLS** = Generalized Least Squares
- **HITL** = Human In The Loop
- **HMDR** = High Dimensional Model Representation
- **IoT** = Internet of Things
- **LR** = Linear Regression
- **LSTM** = Long Short-Term Memory

- $\mathbf{M_{AV}}$ = Availability Sample Mean
- **ML** = Maximum Likelihood
- **MSE** = Mean Squared Error
- **NAR** = Nonlinear Autoregressive
- **NARX** = Nonlinear Autoregressive with External (Exogenous) Input
- **NMP** = Normalized Measure Precision
- **NN** = Neural Network
- **OAT** = One-at-A-Time
- **OFAT** = One-Factor-at-A-Time
- **OLS** = Ordinary Least Squares
- **OS** = Operative System
- $\mathbf{P_i}$ = Probability of *i-th* element
- **QoS** = Quality of Service
- **RNN** = Recurrent Neural Network
- $\mathbf{S^2_{AV}}$ = Availability Sample Variance
- **SA** = Self-Adaptiveness
- **TSA** = Time Series Analysis
- **TSK** = Takagi-Sugeno-Kang (*FIS*)
- **VBSA** = Variance Based Sensitivity Analysis
- $\mathbf{W_i}$ = Weight of *i-th* element
- **WN** = White Noise
- α = Input Parameters Availability
- β = Input Parameters + Model Structure Availability
- χ = Overall Availability
- μ = Membership Function
- η = Normalization Factor
- σ^2_{AM} = Accessing Media Availability Variance
- σ^2_{MD} = Mobile Devices Availability Variance
- σ^2_{SP} = Service Providers Availability Variance
- $\sigma^2_{OVERALL}$ = Overall Availability Variance

E. Abstract

In a specular way of what happens in our life experiences, also the world of engineering and computer science is subject to numerous causes of uncertainty. They may alter the desired behaviour of algorithms, systems (more or less complex) and practical applications. In this work, after a first presentation and investigation phase concerning the phenomenon of uncertainty, with suitable definitions and taxonomies, we will discuss about uncertainty and related problems. We will deal in particular with uncertainty and its management in the context of *self-adaptive* systems. Knowing the runtime behaviour of a system during its execution can help to understand if is the case to change running configuration (in terms of self-adaptiveness) in order to achieve better results.

A lot of malfunctioning or bad performance causes can be related to various sources of uncertainty (and their variables), it should not be a surprise for us. This is the starting point of this work: knowing that uncertainty affects our system, try to face off the linked problems with appropriate paradigms imported from the field of *AI (Artificial Intelligence).*

In particular, we will adopt methodologies and approaches regarding *Fuzzy Logic* and A*NN (Artificial Neural Networks).* Before starting the experimentation phase with these methodologies, we will analyse some alternatives. Some of these alternatives are imported from other fields and adapted as possible.

The real world application under study will be presented in **[Chapter 3]**; all data adopted in this work are public domain data, properly extracted from real world observations in (servers') logs. We will start from this data to extract knowledge concerning the application under analysis, in order to possibly improve the self-adaptive module. In this way we explore both the theoretical and the practical aspects regarding the themes of this work.

F. Sommario

In maniera speculare a quanto succede per le nostre esperienze di vita, anche il mondo dell'ingegneria e dell'informatica è soggetto a numerose cause di incertezza che possono alterare il normale corso di algoritmi, sistemi (più o meno complessi) ed applicazioni pratiche. In questo elaborato, dopo aver presentato ed investigato in merito al fenomeno dell'incertezza, con opportune definizioni e tassonomie, ci occuperemo della gestione dell'incertezza stessa, con un occhio di riguardo nel contesto dei sistemi *auto-adattivi*. Conoscere il comportamento di un dato sistema durante la sua esecuzione può farci comprendere che bisogna optare per una diversa configurazione (in termini auto-adattivi) per ottenere migliori risultati.

Non c'è da stupirsi se tra le possibili cause relative a malfunzionamenti o performance non in linea con gli standard ci sono proprio variabile legate alle varie forme di incertezza. Da qui sorge l'esigenza di questo elaborato: ovvero far fronte alle cause di incertezza adoperando paradigmi importati dal campo dell'*Intelligenza Artificiale*.

Nello specifico adopereremo metodologie ed approcci inerenti la *Logica Fuzzy* e le *Reti Neurali*. Prima di avventurarci nel campo sperimentale con i metodi sopra citati, analizzeremo alcune alternative. Tra queste figurano interessanti alternative importate da altri campi ed adattate ad hoc.

L'applicazione del mondo reale che sarà oggetto di studio verrà descritta ed analizzata nell'apposito **[Capitolo 3]**; tutti i dati che verranno adoperati sono di dominio pubblico e riguardano osservazioni reali estratte opportunamente da log (quindi questi ultimi non sono frutto di simulazioni o generazioni casuali). Questo *modus operandi* è stato stabilito per l'abbondante disponibilità dei dati e per avere un maggiore contatto con il mondo reale, in modo tale da toccare con mano l'influenza dell'incertezza non solo nella teoria, ma anche nella pratica.

Chapter 1 – Introduction

"Call me Ishmael", recalling one of the most famous book introductions of all times, directly from Herman Melville and his masterpiece, *Moby Dick*. That book can be seen a full investigation over the human nature and the uncertainties related to the life. In this master thesis we will investigate about the sources of uncertainty in the field of **model-based software engineering**. In addition, with a good design phase, a self-adaptive system will be able to handle and manage as possible the sources of uncertainty, changing its (*runtime*) behaviour. Recognizing uncertainty and manage it as possible: this will be our main goal, also with the real world application that we are going to consider in our work.

Before entering in the details of the thesis structure, with explicit reference to the chapters, let us try to explain and understand the *'Ariadne's thread'*[1] of this work. It is clear that the main problem that we are going to mitigate and handle as possible is the *uncertainty in models*, explicitly in the field of *model-based software engineering*. From a formal point of view a **model** is *"something that represents another thing, either as a physical object that is usually smaller than the real object, or as a simple description that can be used in calculations"*[2]. Therefore, we can say that a model is a simplified representation of a real world entity. A good model captures some meaningful aspects for the subsequent analysis, with a certain level of abstraction (note that different models of the same entity may have different levels of abstraction and effectiveness). We investigate on the shapes of uncertainty according to the taxonomies that we are going to present later on. Then we analyse a real world case study, based on a video streaming application (with user side mobile devices). We will notice that this real world system is subject to many sources of uncertainty; they may afflict both functional and non-functional aspects of the desired behaviour. After a brief digression on the state of the art techniques to handle uncertainty in models, we introduce two different kind of approaches that we are going to adopt in this work, related to two main *Artificial Intelligence* research fields (*Fuzzy Logic* and *Neural Networks*). The

---

[1] According to Greek mythology, **Ariadne** gave **Theseus** a *thread* with which he found his way out of the *Minotaur's labyrinth*. Nowadays the expression is used to denote the train of thought.

[2] Definition of **Model** from the *Online Cambridge Dictionary* (https://dictionary.cambridge.org)

purpose of the first part is to estimate the overall availability of the system, while the second is to have predictions related to the number of providers available in the future. Hence, a self-adaptive system can be properly trained, programmed and structured in order to change its behaviour according to this information (for example it can switch reference provider or access point, keeping its reliability as high as possible). But we are going see these aspects (with a clearer contextualization and additional explanations) in the next chapters of this work; this is only an introduction with some previews on the thesis' contents.

Coming back to the structure of this thesis, in **[Chapter 2]** we start from the state of the art regarding the uncertainty from our point of view, answering to *Research Questions* like: what is uncertainty? Is it possible to classify the sources of uncertainty? Which are the state of the art methods to handle (as possible) uncertainty? Which are the mechanisms under a self-adaptive system that must work in a proper way also in presence of uncertainty? Which are the new instruments and methodologies to deal with uncertainty?

As we will see, there are different definitions and different approaches, each of one has its own advantages and disadvantages. Starting from the uncertainty definitions **[Chapter 2.1]** and taxonomies **[Chapter 2.2]**, after a brief *excursus* on the state of the art techniques **[Chapter 2.3]**, we conclude the chapter with two interesting approaches, which are gaining importance in the last years: *Fuzzy Logic* **[Chapter 2.4]** and *Game Theory* **[Chapter 2.5]**. The last part of the chapter, **[Chapter 2.6]**, is dedicated to an introduction to the *MAPE-K Adaption Control Loop*, an interesting blueprint towards the world of self-adaptive systems.

The third chapter has two major objectives: describe the real world problem that we are going to analyse (at least under some aspects concerning the uncertainty) **[Chapter 3.1]** and present a preliminary analysis regarding the various categories of uncertainty afflicting the analysed system, with references to the original [*D. Perez-Palacin, R. Mirandola, 2014*] work and approaches **[Chapter 3.2]**. We can state that *self-adaptiveness* of running configurations, as a function of some input parameters, is the key for optimal performances regarding the considered real world application. Dealing with uncertainty means also deal with additional (internal/external) variables, which we can consider as additional inputs of the system.

After the problem presentation and the considerations of the preliminary analysis, we apply the first approach inherited from the encounter between *Artificial Intelligence* and *Logic*: in other words, **[Chapter 4]** is dedicated to *Fuzzy*

Logic. The idea here was to estimate the *Overall Availability* of the considered system, starting from three sub-categories of uncertainty-related availability (*Input Parameters, Model Structure* and *Model Context*), both from an epistemic and an aleatory point of view. The first subchapter, **[Chapter 4.1]**, explains the approach that we are going to use in this chapter, with additional information concerning the mechanisms of *Fuzzy Logic*. Successively, we investigate on the specific availability estimations concerning *Input Parameters Uncertainty* **[Chapter 4.2]**, *Model Structure Uncertainty* **[Chapter 4.3]** and *Model Context Uncertainty* **[Chapter 4.4]**. Then, **[Chapter 4.5]** regards the *Overall Availability* computation, taking in consideration every form of uncertainty. Results and considerations are available in the last part, see **[Chapter 4.6]**.

The second *Artificial Intelligence* approach is the one that we meet in the whole **[Chapter 5]**. Here, in fact, we introduce *Neural Networks*, used in the field of *Data Prediction*[3]. There we try to build a model with two goals: predict both the number of online providers in the future and their dependency in crashes, with a sort of heuristic for the correlation. After a brief introduction to the Neural Network paradigm **[Chapter 5.1]**, the details of the approach **[Chapter 5.2]** and a *Preliminary Curve Fitting Analysis* **[Chapter 5.3]** we go through the '*core*' of this section. In fact, in the following chapters we build and see the predictive results for different neural networks, with and without transients in chapters **[Chapter 5.4]**, **[Chapter 5.5]**, **[Chapter 5.6]**. In conclusion, we perform additional considerations and results' analyses in **[Chapter 5.7]**.

[Chapter 6] is fully dedicated to the conclusion of this works and highlights some possibilities for extensions and future research directions.

The final **[Appendix]** explains the structure of *MATLAB* code available on *GitHub*, for *Fuzzy Logic*, *Neural Networks* and the available *Datasets***.**

[3] We refer here to **Predictive Data Analytics**, a variety of mathematical and statistical techniques from *predictive modelling, artificial intelligence, machine learning* and *data mining* that analyse current and historical facts to make predictions about future or otherwise unknown/uncertain events.

Chapter 2 - Uncertainty: State of the art

2.1 What is Uncertainty?

"There are some things that you know to be true, and others that you know to be false; yet, despite this extensive knowledge that you have, there remain many things whose truth or falsity is not known to you. We say that you are uncertain about them. You are uncertain, to varying degrees, about everything in the future; much of the past is hidden from you; and there is a lot of the present about which you do not have full information. Uncertainty is everywhere and you cannot escape from it."

Dennis Lindley, Understanding Uncertainty, (2006)

Uncertainty is strongly related with our lives, from our birth to the day of our death. The human being has (by nature) a limited perception of the world and its phenomena, therefore it is subject to uncertainty from different points of view. From a formal point of view, uncertainty is a situation which involves **imperfect and/or unknown information**. It arises in subtly different ways in a number of fields, including insurance, philosophy, physics, statistics, economics, finance, psychology, sociology, engineering, metrology, and information science. It applies to predictions of future events, to physical measurements that are already made, or to the unknown.

Uncertainty may be purely a consequence of a **lack of knowledge** of obtainable facts. Let us present a first example: there may be uncertainty about whether a new rocket design will work, but this uncertainty can be removed with further analysis, testing and experimentation. At the subatomic level, however, uncertainty may be a fundamental and unavoidable property of the universe. In quantum mechanics, the *Heisenberg Uncertainty Principle* puts limits on how much an observer can ever know about the position and velocity of a particle (referring to the same tame instant). This may not just be ignorance of potentially obtainable facts but that there is no fact to be found. Of course, there is some controversy in physics as to whether such uncertainty is an irreducible property of nature or if there are '*hidden variables*' that would describe the state of a particle even more exactly than Heisenberg's uncertainty principle allows. But

once again, this is related to the limited amount of knowledge about real world phenomena that human beings have. In other words, still uncertainty.

Let us come back to our domain: knowing how to deal with uncertainty is the key for the design and the deployment of *adaptive* systems. Uncertainty arises in partially observable and/or stochastic environments, as well as due to ignorance and/or indolence. There is not a single definition of uncertainty in the real world and the same is in the field of *system/software engineering*. The common sense suggests that uncertainty of a general system is related to the degree of unpredictability of its behavior. The more a system is unpredictable, the more it is subject to uncertainty. This idea is fundamental in the field of science called *Prediction Theory*, where the purpose is to predict something (for example a variable) knowing the past history of the system and considering the possible influence of uncertainty (in different shapes, it may be a *White Noise* [WN][4] for example).

When an event is uncertain, we cannot predict it in a deterministic way, but we know that can happen. For example, there are not state of the art methods that can predict in a deterministic way an earthquake, but we know that they can happen. A civil engineer should design the houses of the cities in such a way that they are robust against these kind of events. If we do a parallelism with the field of Computer Science, the situation is similar: we know that in some applications there are forms of uncertainty and we have to design our system in order to handle these situations (typical idea under the self-adaptive systems).

Understanding what is uncertainty from an intuitive point of view it is not hard, we need only a little bit of common sense. But handling uncertainty in our cases of interests may be a big problem to solve. In the next chapter we will present the different forms of uncertainty according to the taxonomies presented in literature and we will try to give a systematic approach in order to manage this phenomenon, typical of the real world applications as we have understood. Note that in this work we will start from a very general definition of uncertainty, but our main focus will be explicitly to deal with **uncertainty in models**.

---

[4] A **White Noise** [WN] is a *random signal* having equal intensity at different frequencies, giving it a constant power spectral density. The term is used, with this or similar meanings, in many scientific and technical disciplines, including physics, acoustic engineering, telecommunications, statistical forecasting, and many more.

2.2 Taxonomy of Uncertainty

"Knowledge is an unending adventure at the edge of uncertainty. "

Jacob Bronowski, The Origins of Knowledge and Imagination, (1979)

Once we have understood the dualism between knowledge and uncertainty, let us try to explore the *'edge of uncertainty'* in a systematic way. As for the major forms of conceptualization, also uncertainty has a lot of possible forms of classification and taxonomies. These are collected from different perspectives, both in the humanistic and in the scientific field. We want to use this knowledge (collected from different fields) and apply it in our specific domain. A first subdivision may be the one proposed by [C. *Tannert et al. 2007*] between the *Objective Uncertainty* (that is related to the *Epistemological* and the *Ontological* sub-classes) and the *Subjective Uncertainty* (related to the *Moral* and the *Rule* sub-classes). Let us present a self-explaining scheme of this taxonomy (from an ethical perspective):

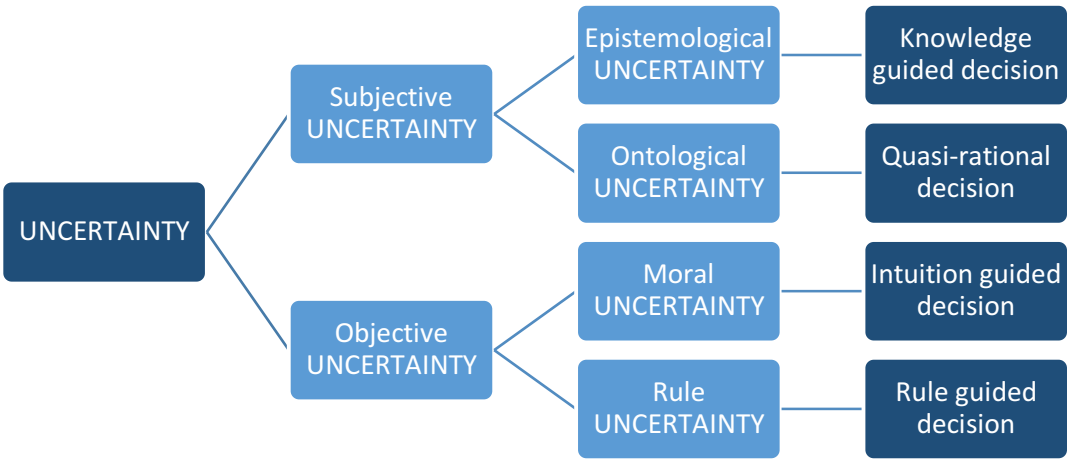


Figure 1. Ethical Taxonomy of Uncertainty with the different decision types.

Coming back to the domain of engineering (and especially Computer Science) we will refer to some relevant taxonomies in literature. In particular, we will consider the [*D. Perez-Palacin, R. Mirandola, 2014*] one, that will be adopted as reference in the next chapters of this work.

Our reference taxonomy is built on the bases of the following definition of uncertainty given by [*Walker et al. 2003*].

Definition of Uncertainty:
"Any deviation from the unachievable ideal of completely deterministic knowledge of the relevant system"

The **taxonomy** reflects the categorization of Walker's paper adapted on our specific domain of interest. Therefore, there a are *three orthogonal dimensions*, with their specific sub-classes. Let us first present the scheme and then the explanation for this specific taxonomy:

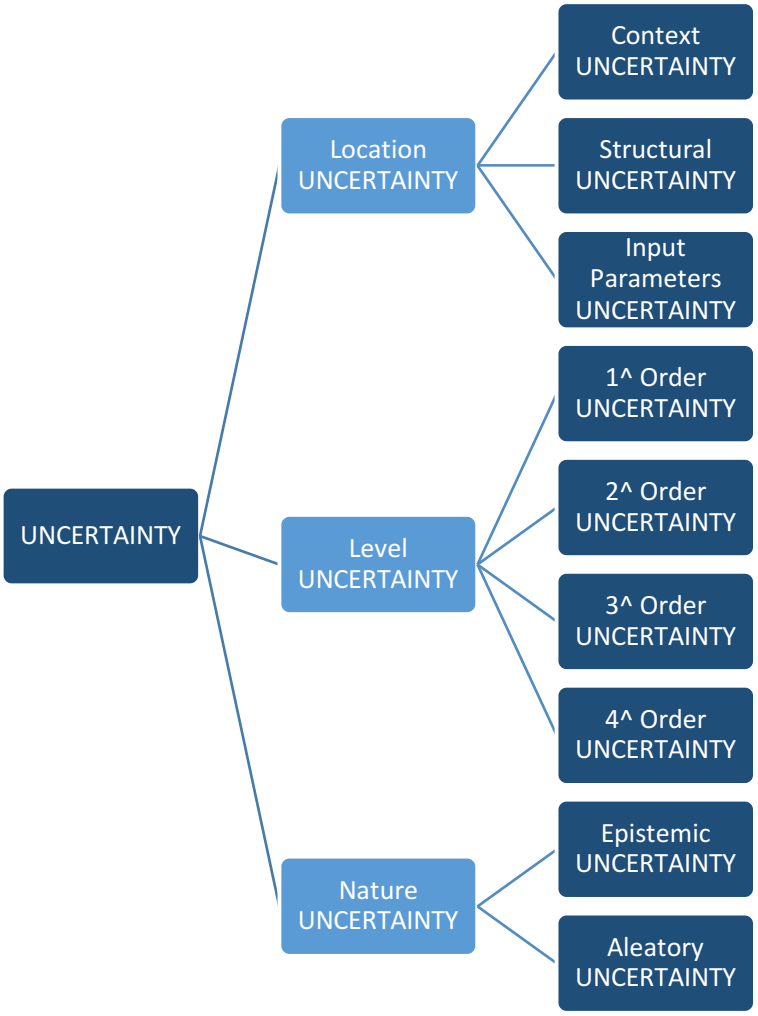


Figure 2. Taxonomy of Uncertainty (Location, Level, Nature) with the specific sub-categories according to our domain from [*D. Perez-Palacin, R. Mirandola, 2014*].

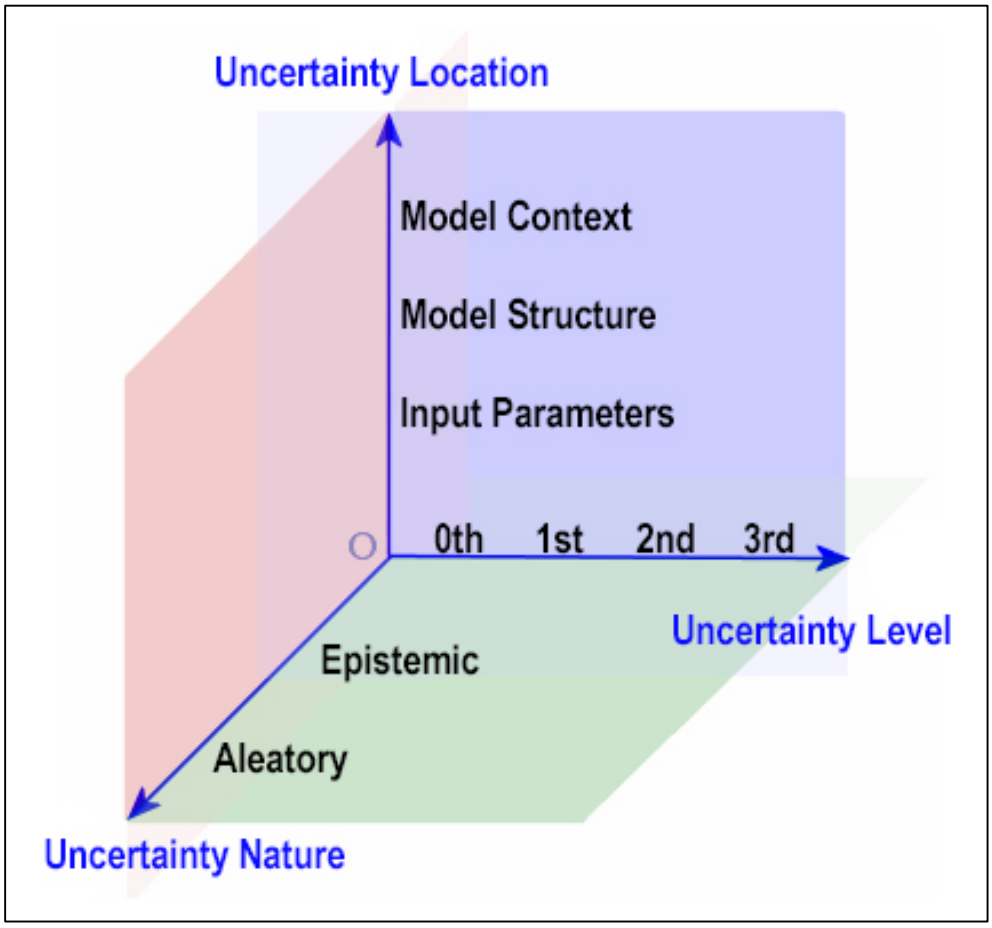


Figure 3. [*D. Perez-Palacin, R. Mirandola, 2014*] taxonomy of uncertainty (dimensions with their discrete values, graphical representation).

Let us present in details the **three main categories** (**Location**, **Level**, **Nature**) with related subcategories:

- **LOCATION UNCERTAINTY** – It is related to the specific **location** of uncertainty in the considered model. It has three subclasses: *Context, Structural* and *Input Parameters Uncertainty*.

  - *Model Context Uncertainty*: it is related to the identification of **model's boundaries**, process that aims to specify the part of the real world to be effectively modelled (and the part that we want to abstract with the model, simplified representation of the reality).

- *Model Structural Uncertainty*: regards the form of the **model itself**; in practice concerns what kind of model is the more accurate for the specific goals and objectives. Clearly different model's structures can have a different impact on this category of uncertainty.

- *Input Parameters Uncertainty*: it is related to the **input parameters** of the model with their degree of uncertainty (for example with respect to the methods used for parameters calibration or different types of errors). This form of uncertainty is typical of different fields of engineering where measurement accuracy has an important role (often critical, for example in a nuclear power plant).

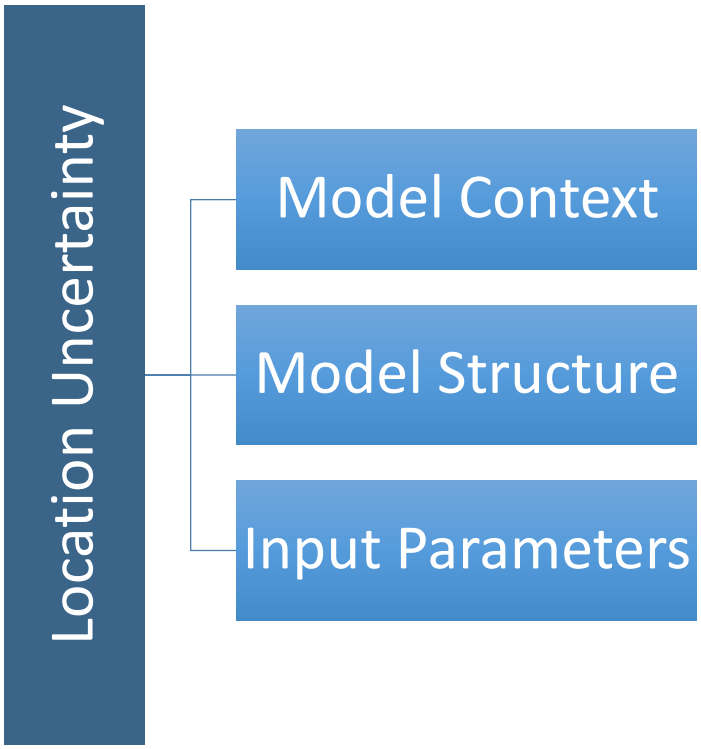


Figure 4. Location Uncertainty (with *Model Context, Model Structure* and *Input Parameters* sub-classes).

- **LEVEL UNCERTAINTY** – It measures the **degree of uncertainty** with a specific order (from 0 to 3 or from 0 to 4 if we consider the *Meta-Uncertainty* as a possible level). But let us see the details of this classification. Knowing that the real world is *'not only black or only white'*,

as also *Fuzzy Logic* states, there are not just *'Full Knowledge'* and *'Full Uncertainty'*, but there is also *'something'* in the middle. Therefore, there are different degrees of knowledge and related uncertainty.

The following one is the taxonomy of **levels of ignorance (uncertainty)** already proposed in [*Armour, 2000*]:

- **0th order of uncertainty. Lack of uncertainty**, i.e., knowledge.
- **1st order of uncertainty. Lack of knowledge.** The subject lacks knowledge about something but he/she is aware of such lack (i.e., known uncertainty).
- **2nd order of uncertainty. Lack of knowledge and lack of awareness**. The subject does not know that he/she does not know.
- **3rd order of uncertainty. Lack of process to find out the lack of awareness.** The subject does not have any way to move from not knowing that he/she does not know to, at least, be aware of the existence of the uncertainty.
- **4th order of uncertainty. Meta uncertainty.** Uncertainty about orders of uncertainty. This form of uncertainty sometimes is not considered in the taxonomy (after all, it is a *'Meta'* form of uncertainty). Note that as in [*Armour, 2000*], this fourth level is included for completeness.

The most interesting step is the one from the *level 2* (lack of knowledge and lack of awareness) to *level 1* (lack of knowledge), called **uncertainty identification**. This is a crucial point for the applications: only once we have understood the sources of uncertainty we can analyse and try to solve the problem.

In self-adaptive systems this can be done with the assistance of a **control loop**. We will see an example with the **MAPE-K controller model** in **[Chapter 2.6]**. The idea is to improve the runtime behaviour and efficiency on the bases of the *feedback* given by the controller (a similar idea is used in order to feed Neural Networks with the *Backpropagation algorithm*[5] in the field of Artificial Intelligence).

[5] See '*A brief introduction to Neural Networks*' **[Chapter 5.1]** for more details concerning the **Backpropagation** training algorithm; it is used in order to improve the weights of the *artificial neurons' synapses* (connections).

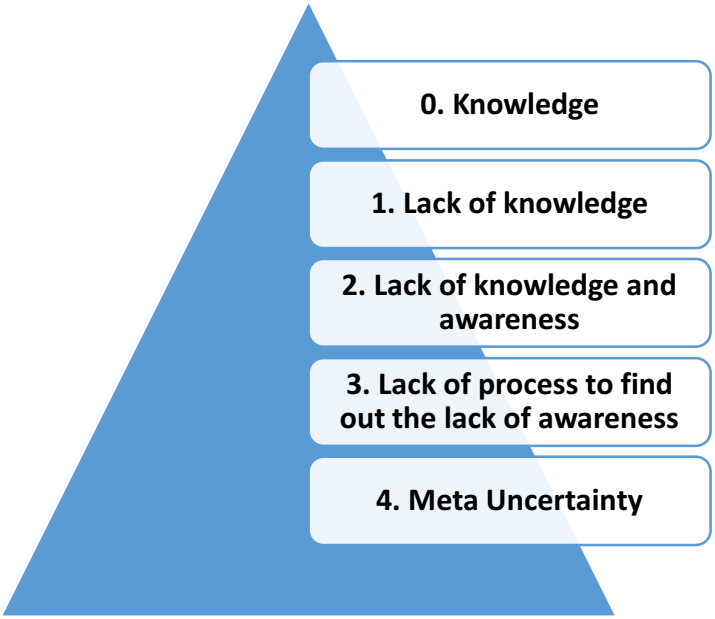


Figure 5. Armour's classification [*Armour, 2000*] of Uncertainty (Level) that ranges from 0 to 4.

- **NATURE UNCERTAINTY -** The **nature** of uncertainty reflects whether the uncertainty is due to imperfection of the acquired knowledge (**Epistemic**) or due to inherent variability (form a probabilistic point of view) of the phenomena being described (**Aleatory**). In the first case we can theoretically improve the knowledge acquisition process with more sophisticated instruments. In the second case the uncertainty is inherently related to the stochastic randomness of some events of the real world. Therefore, we can summarize:

  - **Epistemic uncertainty** is related to the lack of enough data to build reliable knowledge, imperfection of the acquired data or imperfection in the process of building the knowledge from the data.

  - **Aleatory uncertainty** is related to inherent variability of some parts under consideration or randomness of events.

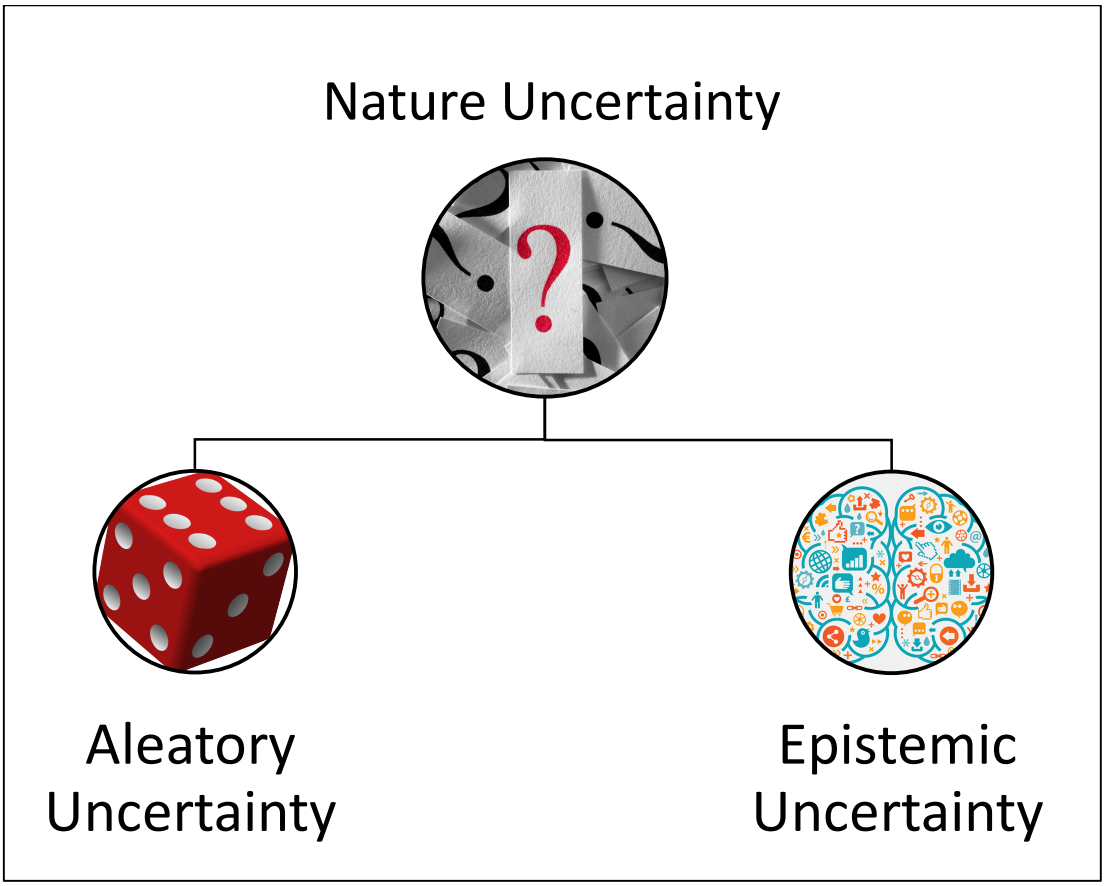


Figure 6. Nature Uncertainty (with *Aleatory* and *Epistemic* sub-classes).

Given that output uncertainty is *unavoidable*, the main goal of a system designer will be deal with uncertainty, with specific *'mitigation'* techniques. Note that our purpose will be *uncertainty management*[6] (after its *identification*), according to the proposed dimensions. Let us present now a tabular taxonomy of uncertainty in the context of **modelling**.

[6] **Uncertainty management** will be pursued in this work with two main techniques: *Fuzzy Logic* **[Chapter 4]** and *Neural Networks* **[Chapter 5]**.

Sources of uncertainty in models (Taxonomy)

Coming back to our domain of interest, let us understand how to connect different forms of uncertainty to the taxonomy proposed before.

The following table (from [*D. Perez-Palacin, R. Mirandola, 2014*]) relates the sources of uncertainty in a self-adaptive system to the classification according to our reference taxonomy (only *Location* and *Nature*):

SOURCE OF UNCERTAINTY	CLASSIFICATION	
	LOCATION	NATURE
Simplifying assumptions	Structural/Context	Epistemic
Model drift	Structural	Epistemic
Noise in sensing	Input Parameters	Epistemic/Aleatory
Future parameters value	Input Parameters	Epistemic
Human in the loop (HITL)	Context	Epistemic/Aleatory

Objectives	Input Parameters/Context	Epistemic
Decentralization	Context/Structural	Epistemic
Execution context/Mobility	Context/ Structural/ Input parameters	Epistemic
Cyber-physical system	Context/ Structural/ Input parameters	Epistemic
Automatic Learning	Structural/Input Parameters	Epistemic/Aleatory
Rapid evolution	Structural/Input Parameters	Epistemic
Granularity of models	Context/Structural	Epistemic
Different sources of information	Input parameters	Epistemic/Aleatory

Figure 7. Table which relates the source of uncertainty in models (and self-adaptive systems) with the dimensions of Location and Nature of our taxonomy. Additional information and details available in [*D. Perez-Palacin, R. Mirandola, 2014*].

2.3 State of the art methodologies to handle Uncertainty

"Leaders, policy makers, authorities and all parties in society are making decisions everyday under uncertainty, possibly, with some complexities. With complexities, we mean that the knowledge about both the environment and the consequences of decisions is not perfect. […] Uncertainty is a phenomenon that has a deep root in daily life. It makes us free to choose. Our brain converts uncertainty into fear in order to create a motivation to do something. "

Ibrahim Özkan and I.Burhan Türks,
Uncertainty and Fuzzy Decisions, (2014)

There are several methods that can be used to handle and reduce uncertainty as much as possible. Some of these are consolidated from decades, other are new approaches proposed in the last years. The main goal remains to use these approaches in order to attenuate uncertainty in practical applications, for example in self-adaptive systems. Some important 'classical' approaches are the following five, as presented in the [*D. Perez-Palacin, R. Mirandola, 2014*] paper**:**

- **Model averaging**: first of all, several models are generated (with different modelling languages and domains if needed); then they are integrated with ad-hoc averaging (for example with probabilities or weights) into a final model. For example, if we want to estimate a parameter we will use the following formula:

$$\mathbf{avg(\theta)} = \frac{\sum_{i=1}^{N=Number\ of\ Models} (\theta_i \mathbf{x}\ W_i)}{\sum_{i=1}^{N=Number\ of\ Models} W_i}$$

  Where **avg(θ)** is the average parameter that we want to estimate for the integrated model, **N** represents the number of auxiliary models considered in the preliminary phase, θ_i is the value of the parameter in the *i-th* model and W_i represents the weight that we want to give to the *i-th* auxiliary model. Note that we can assign also probabilities instead of the weights if needed, with the following formula:

$$\mathbf{avg(\theta)} = \sum_{i=1}^{N=Number\ of\ Models} (\theta_i \mathbf{x}\ P_i)$$

Where clearly the **Pi** are the probabilities of the auxiliary models. Note that, according to the *Probability Theory*, the summation probabilities of the auxiliary models must be equal to 1; regarding the weights instead we can use a compact term $\mathbf{W_{TOT}}$ in order to indicate their algebraic summation. Therefore:

$$\sum_{i=1}^{N=Number\ of\ Models} \mathbf{W_i} = \mathbf{W_{TOT}}$$

And:

$$\sum_{i=1}^{N=Number\ of\ Models} \mathbf{P_i} = \mathbf{1}$$

They assume a similar meaning and it is possible to compute a version from the other with simple mathematics. This method assumes the fact that different point of views (from the different models) linked together into an integrated model have better results in terms of robustness and veracity. We want to remark that remains the fact that the system designer must select in an efficient way the right auxiliary models and the weights/probabilities to use. As a drawback, this may be hard in several application and may request a lot of time or capacities.

- **Model discrepancy**: this second approach assumes that the model is not the 'true' model of the system, therefore the idea is to minimize the discrepancy between the model output and the 'true' output value. We can visualize the idea as a *feedback loop* that shows the difference between the desired behaviour and the actual one. In order to observe the difference between the observed data and the model a *discrepancy function* is used as evaluation mechanism. As anticipated before, large values of the discrepancy function outcome (or discrepancy term) indicate a poor fit of the model to the real world data. There are several discrepancy function well-known in literature, including:

    - *Maximum Likelihood (ML)*
    - *Generalized Least Squares (GLS)*
    - Ordinary *Least Squares (OLS)*

  These are considered the *"classical"* discrepancy functions. Clearly it is possible to define and use other discrepancy functions, but they have to

meet some criteria. In fact, they must be non-negative (for example, always greater than or equal to zero), they can be zero only if the fit is perfect (when the model and parameter estimates perfectly reproduce the observed data). We want to remark that the original idea of this technique is then developed in different methodologies used in the field of Artificial Intelligence (from the 'basic' *Linear Regression* to the *Convolutional Neural Networks*).

- **Sensitivity analysis**: the idea here is to perform little variations over the parameters with respect to the optimal situation and see how the output changes. From an intuitive point of view, a robust model must perform well (in terms of functional and non-functional requirements) also with variations related to uncertainties. From a formal point of view this technique studies how the uncertainty in the output of a mathematical model can be apportioned to the different sources of uncertainty in input. The sensitivity analysis is based on the process of recalculating outcomes under different starting assumptions and with little variations of some parameters of interest, in order to observe what happens. It is clear that this technique aims to test the robustness of the model in presence of noise (and sources of uncertainty in general). In addition, it helps us to identify and understand better the dependencies and the relationships between input and output variables of the model under analysis. Note that some optimization methods, like *Monte Carlo filtering*, use the principle of sensitivity analysis in order to find either minima or maxima in the search space of the problem, but this is an additional application of the method. For the sake of completeness, here are listed the most famous and used methodologies based on the sensitivity analysis:

    - *One-at-A-Time (OAT),* also called *One-Factor-at-A-Time (OFAT)*
    - *Scatter Plots Method*
    - *Regression Analysis* (for example *LR, Linear Regression*)
    - *Local methods* (involving the partial derivative)
    - *Variance-based Methods* (also called *VBSA - Variance Based Sensitivity Analysis*)
    - *Screening (*for example the *EEM - Elementary Effect Method)*
    - *Machine Learning Approaches*, for example the *Emulators* (also known as meta-models, surrogate models or response surfaces)
    - *HMDR – High-dimensional model representations*
    - *FAST – Fourier Amplitude Sensitivity Test*
    - *Monte Carlo-based filtering model*

Note that these methodologies (based on *Sensitivity Analysis*) are then subdivided into sub-classes, each one with its specific features and algorithmic characteristics.

- **Possibility theory**: this approach provides a reasoning framework for uncertainty in model parameters, assuming less information with respect to the probability theory. The main difference with respect to the probability theory is that there is not restriction for a value when other values are known. Historically speaking, it was introduced in 1978 by Professor Lofti Zadeh, the inventor of Fuzzy Logic and Fuzzy Sets, as an extension to his original theory. This theory substitutes the idea of probability (with a given value belonging to the 0 to 1 interval) with two new concepts: the *possibility* and the *necessity* of the events. It is important to underline that by construction this theory is based on a multi-valued logic like the Fuzzy Logic (whereas the classical Probability Theory is based on the two-valued logic). However, possibility and necessity measures can also be the basis of a full-fledged representation of partial belief that parallels probability. It can be seen either as a coarse, non-numerical version of probability theory, or a framework for reasoning with extreme probabilities, or yet a simple approach to reasoning with imprecise probabilities. The two extremes of this theory, from the conceptual point of view, are *Complete knowledge* and *Complete ignorance*. A great reference with the formalization of the theory, the terminology, the axioms, the properties, the history and the recent developments is available in [*Didier Dubois, Herni Prade, 2011*].

- **Evidence theory**: also in this case, there is a reasoning framework that requires less information (or assumptions) with respect to the probability theory (as we have already seen with the *Possibility Theory*). In literature it is known also as **Theory of belief functions** or **DST** (*Dempster-Shafer Theory*, from the name of the creators). This theory allows the system designer to combine evidence from different sources and reach a level of belief, represented by the *belief function*. From a taxonomical point of view, it can be seen as a generalization of the *Bayesian theory of subjective probability*, where the concept of degree of belief (or mass) is represented as a special function (called belief function, as we have said) instead of a Bayesian probability distribution. It is based on two main functions

(**plausibility** and **belief**), where the first one is always greater or equal with respect the second one. Therefore, given a general event E:

Belief(E) ≤ Plausibility(E)

Once the evidences are recognized and collected, are combined with a specific rule of combination (for example the *Dempster's rule of combination*). More technical details are available in the book *'A Mathematical Theory of Evidence'* by Glenn Shafer [*Glenn Shafer, 1976*].

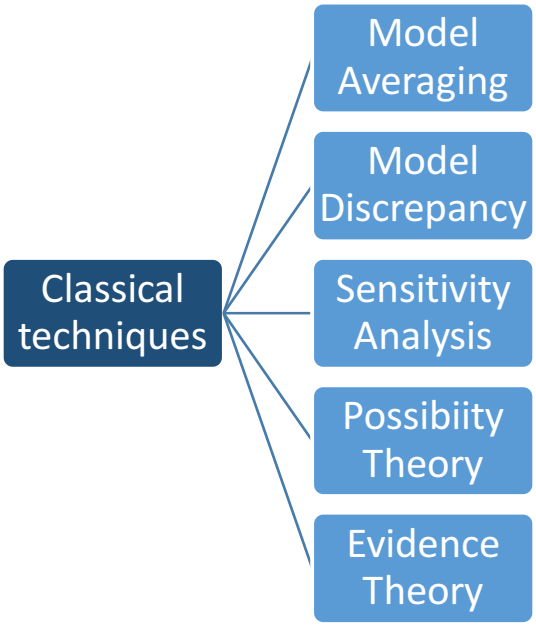


Figure 8. Macro-categories of 'classical' techniques to handle uncertainty.

Note that in the last years, new techniques to deal with uncertainty (also inspired to these categories) are gaining attention due to their interesting results. Some examples are *Fuzzy Controllers, Neural Networks* and most recently *Deep Learning* (based on the *Model Discrepancy*). We will see some of these techniques applied in our case study in **[Chapter 4]** and **[Chapter 5]**, trying to point out their advantages and disadvantages with respect to the other techniques. As anticipation, in **[Figure 9]** is presented a *Fuzzy Controller*, while in **[Figure 10]** is presented a *Neural Network* model.

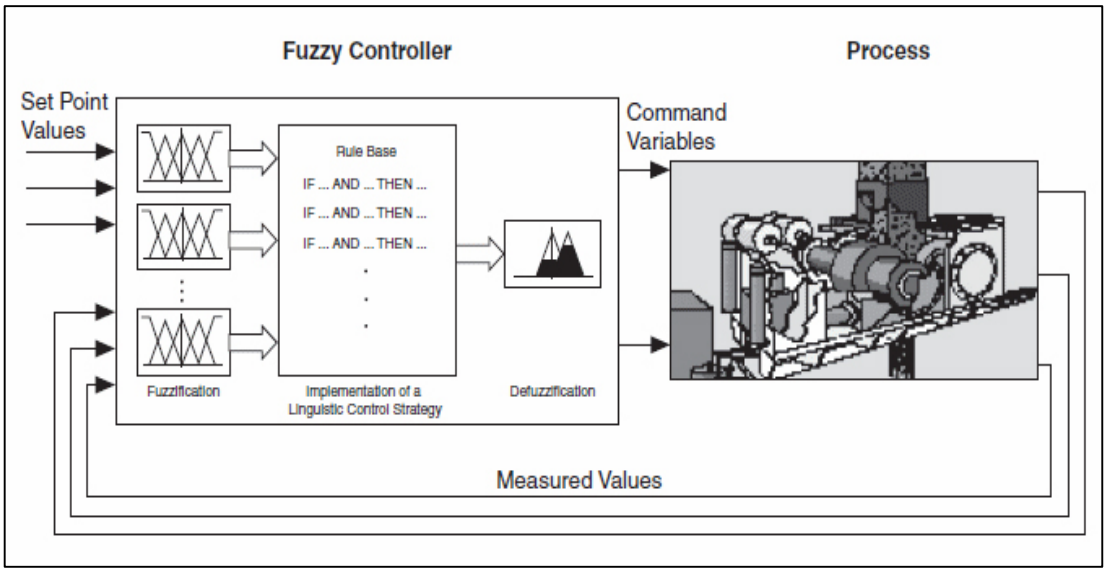


Figure 9. An example of *Fuzzy Controller* (with its white-box schema) used in the context of an industrial process.

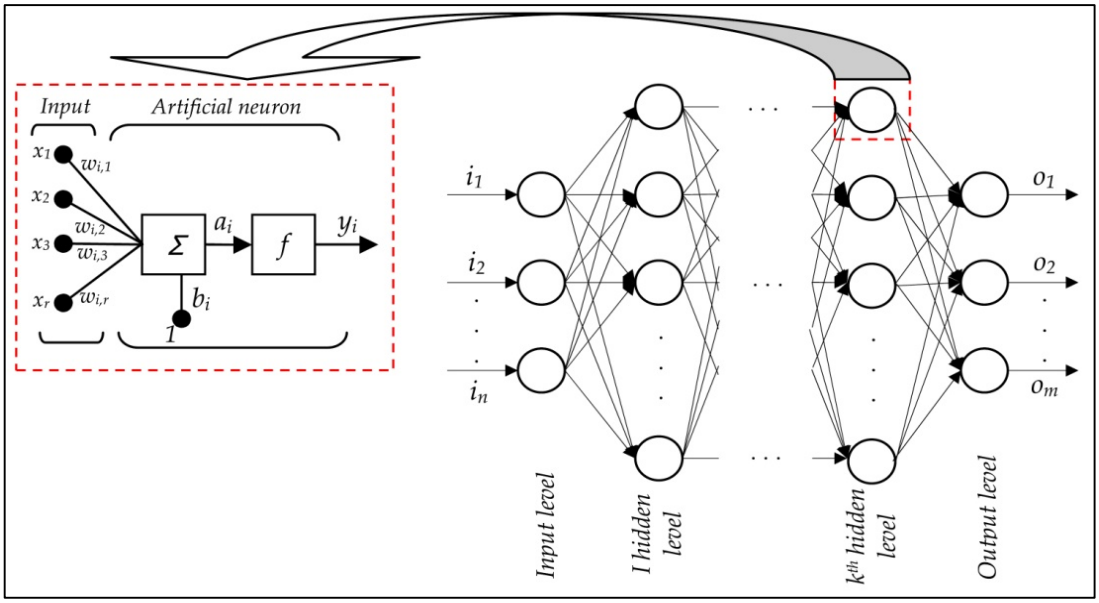


Figure 10. An example of a *multi-layer Neural Network* with the highlighted structure of the artificial neuron.

2.4 Fuzzy Logic and Uncertainty

"The advent of the Computer age has stimulated a rapid expansion in the use of quantitative techniques for the analysis of economic, urban, social, biological and other types of systems in which it is the animate rather than in dominant role. At present, most of the techniques employed for the analysis of humanistic, i.e., human centered systems are adaptations of the methods that have been developed over a long period of time for dealing with mechanistic systems, i.e., physical systems governed in the main by-the laws of mechanics, electromagnetism, and thermodynamics. The remarkable successes of these methods in unraveling the secrets of nature and enabling us to build better and better machines have inspired a widely held belief that the same or similar techniques can be applied with comparable effectiveness to the analysis of humanistic systems."

Lofti Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes" in IEEE Transactions on Systems, Man and Cybernetics (1973)

Another powerful instrument that can be used in order to deal with uncertainty is *Fuzzy Logic* (and its extensions) by Lofti Zadeh. Let us try to understand what is Fuzzy Logic and how can be helpful for our purposes. Fuzzy Logic is a many-valued logic where truth values of the variables range in the interval 0 to 1 (differently from the *Boolean Logic*, where a truth value can be either *True* [1] or *False* [0]). Therefore, in Fuzzy Logic there is the notion of partial truth of the considered value. The more a truth value will be true, the more its value will be close to 1. From the opposite side, a truth value close to 0 suggests a high degree of falseness. In addition, linguistic variables are used (with values managed by *membership functions* μ_x, as we will see later). We want to remark that historically there were other multi-valued logics before, like the logical systems introduced by *Łukasiewicz* and *Tarski* at the beginning of the XX century (see [*Revaz Grigolia, 2015*] for more details).

Coming back to Fuzzy Logic, it was introduced in 1965 by Lofti Zadeh together with its *Fuzzy Set Theory*. Nowadays Fuzzy Logic has several application domains, that range from *Control Theory* to *Artificial Intelligence*. The main conceptual purpose is the approximation of the human reasoning mechanism (with specific *labels*) in order to handle uncertainty associated with human perception and real world phenomena.

These approximations in **human decision making process** are done through *'fuzzy modifiers'.* Let us show some examples:

- *"About", "low", "high", "big", "fast", "slow", "near", "far" …*

There is a clear symmetry with the human language and the human decision making process according to these fuzzy modifiers. It is interesting to know that there is a specific research field who focuses on the usage of this words, called *CWW (Computing with Words).*

Now let us introduce the principles of **Fuzzy Theory**, with a brief description:

- In **Fuzzy Theory** every element belongs to a *concept class* (for example class A). There is a partial *degree of membership* of the element with respect the class, reflected by the outcome of the so called membership function. Formally we will say that:

$$\mu_A: X \rightarrow [0,1],\ \mu_A(x) = a \in [0,1],\ x \in X$$

  where $\mu_A(x)$ is the membership assignment of an element 'x' to a concept class (A in our example) in a certain proposition. The above representation is generally accepted as *Type-1 Fuzzy Sets*, which assumes the membership values are certain. There are three major steps that we need to perform in order to achieve results using Fuzzy logic:

  1. **Fuzzification:** *fuzzify all input values into fuzzy membership functions. In the general case we start from real values, going towards the fuzziness through membership functions.*

  2. **Application of the Fuzzy rules:** *Execute all applicable rules in the rule-base to compute the fuzzy output functions. Typically, this step is done through IF-THEN rules.*

     Example: *IF A is a1 and B is b2 and C is c3 THEN X is x1 and Y is y1*

  3. **De-fuzzification:** *de-fuzzify the fuzzy output functions to get "crisp" output values. This is the opposite of what performed in the first step of the process.*

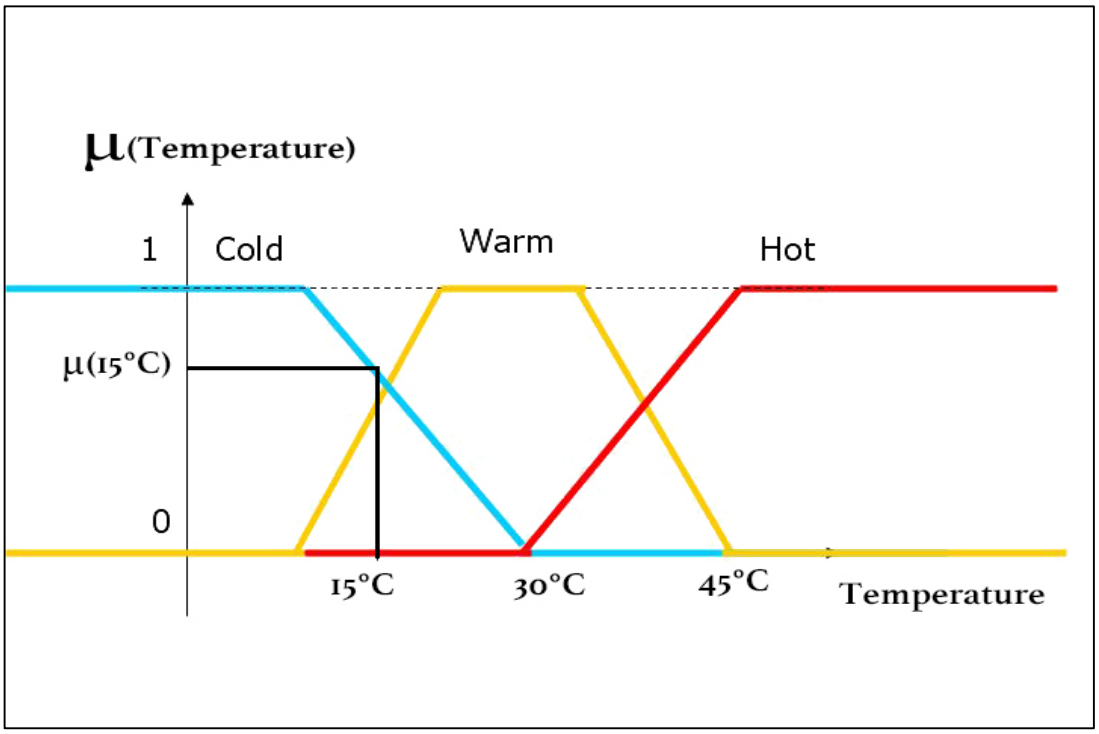


Figure 11. An example of a Fuzzy Membership Function related to the Temperature variable. Note the linguistic labels: *'Cold', 'Warm', 'Hot'*.

Unfortunately, the majority of concepts in **Fuzzy Theory** are assumed to be definable to be true to a degree. Because of this, Zadeh introduced also an extension of his Theory where the membership values are not certain. In this case we will refer to the *Type-2 Fuzzy Sets.* There are many fields of research both for *Type-1 Fuzzy Logic* (values of membership functions known) and *Type-2 Fuzzy Logic* (values of membership functions are fuzzy).

Let us come back to uncertainty and related concepts, as vagueness and ambiguity. From a formal point of view, *vagueness* or *ambiguity* are sometimes described as "*second order uncertainty*", where there is uncertainty even about the definitions of uncertain states or outcomes (as we have seen before). The difference here is that this uncertainty is about the human definitions and concepts, not an objective fact of nature. It is usually modelled by some variations and extensions of Lofti Zadeh's Fuzzy Logic.

In [*Smithson, 1989*]'s taxonomy, **uncertainty** is a part of **incompleteness** which is product of **errors**. For him, **fuzziness** can be seen as a specific type of **vagueness**.

Historically speaking, from the years 1970s-1980s principles of fuzzy theory applied to **classical statistical decision theory**. Here are some example of application fields and extensions to the original theory:

- *Fuzzy acts (or actions)*
- *Fuzzy events*
- *Fuzzy probabilities*
- *Fuzzy utilities*
- *Fuzzy information*
- *Fuzzy linguistic modeling*

But how we can explain the success of Fuzzy Logic in these several fields? There are two major reasons:

1. *Power of linguistic explanation with resulting ease of understating.*
2. *Tolerance to imprecise data/noise which provides flexibility and stability for prediction and decision making.*

It is clear that the second aspect is the key when we deal with *uncertainty* in our problems. Starting from heuristics to the modern tools of handling uncertainty is a vast area of research. Fuzzy logic is an approach that has been used (and can be still used) effectively to decide under uncertainty. After Zadeh's original paper about fuzzy logic [*Lofti Zadeh, 1965*], researchers were attracted to apply the tools created with fuzzy logic for complex problems in almost every fields, as we have also seen before. The body of research has been increasing very fast. Generally, first discussions are about *Type 1* fuzzy logic where the degree of memberships are assumed precise. The new discussions where the degree of membership become fuzzy took place after realizing the certain degree of membership may not be the founding ground for fuzzy approaches. We understood that this new approach is called as *Type 2* (and of course higher order up to *Type n* introduced) fuzzy systems.

Fuzzy logic and fuzzy system modelling is proved to be a close to approximation of human decision making and perception processes. Therefore, as the application of fuzzy system modelling, the advances in fuzzy decisions and as a particular case, perception based decisions, may project this approach as one of the most profitable and effective when we deal with uncertainty in real world applications.

2.5 Game Theory and Uncertainty

"Finding dominant strategies is considerably easier than the search for the Holy Grail. Consider Alfred, Lord Tennyson's familiar line: << Tis better to have loved and lost that never to have loved at all. >> In other words, love is a dominant strategy. "

Avinash K. Dixit,Barry J. Nalebuff, Thinking Strategically: The Competitive Edge in Business, Politics, and Everyday Life, (1993)

We already said that uncertainty is studied in several scientific areas, from different perspectives. Let us briefly try to explain what is **Game Theory** and why it is important. From a formal point of view, Game Theory is "*the study of mathematical models of conflict and cooperation between intelligent rational decision-makers*". Game theory has several application fields, such as economics, political science, and psychology, as well as logic, computer science and biology. Originally, it addressed *zero-sum games*, in which one person's gains result in losses for the other participants. Nowadays, Game Theory applies to a wide range of behavioral relations, and is now a reference term for the science of logical decision making in humans, animals, and computers. Once we understood that real world decisions are subject to uncertainty, we can imagine that Game Theory can give a massive contribution in order to select the right action (or series of actions) to perform. Also a self-adaptive system can choose its execution mode on the bases of the perceived environment and some criteria according to Game Theory; this application can be very useful for our purposes.

Historically speaking, modern game theory began the last century with the idea regarding the existence of mixed-strategy equilibria in two-person zero-sum games and its proof by John von Neumann. His paper was followed by the 1944 book '*Theory of Games and Economic Behavior*', co-written with Oskar Morgenstern, which considered cooperative games of several players. The second edition of this book provided an axiomatic theory of expected utility, which allowed mathematical statisticians and economists to treat decision-making under uncertainty.

In the following years, this theory was developed extensively and deeply by many scholars and nowadays has several application fields as said before. Coming back to the core of the theory, we will see that the objectives of Game Theory are *decision making problems* with series of negotiations in which the purpose is to

reduce uncertainty (and maximize the expected utility, given the *Payoff Matrix* where utilities are represented in a tabular form).

Acts\Events	Event '1'	Event '2'	…	Event 'h'
Acts '1'	Payoff (1,1)	Payoff (1,2)	…	Payoff (1,h)
Acts '2'	Payoff (2,1)	Payoff (2,2)	…	Payoff (2,h)
…	…	…	…	…
Acts 'k'	Payoff (k,1)	Payoff (k,2)	…	Payoff (k,h)

Figure 12. An example of a (parametrical) Payoff Matrix used in Game Theory.

The idea, based on **EUT (Expected Utility Theory)** is to maximize the expected utility (with the **best policy**, a sequence of actions usually indicated as **Π***). Given that there are sources of uncertainty (maybe in other players' strategies or in the perception of the environment), which are the best actions to perform in order to maximize the utility? Game Theory aims to answer to questions like this, given its mathematical background and principles.

Let us go a step backwards, starting from problem modeling and arriving to the decision criterion to be chosen. We know from sociology that there are two main types of **human decision process**: *descriptive* and *prescriptive*. In these approaches, descriptive modeling attempts to identify system structure that capture the behavior characteristics as best as it can, whereas the prescriptive modeling attempts to determine the best approximate reasoning schemas that produce the best prediction of system behavior for a given descriptive model. Human decision processes depend on the perceived world and decision maker faces uncertainties at any stage of a decision process.

In order to model a decision problem in a formal way, it is required:

1. *Courses of action (acts)*
2. *States of nature (events)*
3. *Payoff associated with each action-states*
4. *Degree of knowledge (and uncertainty!) about states and nature*
5. *Decision criterion that help to select the course of action (called 'solution concept' in Game Theory)*

In the general case, every event has associated probabilities that are assigned. These probabilities reflect the degree of knowledge available about the states of the nature. It may happen that we can have a certain event: in that case it will have its probability set to one (as a parallel situation, the impossible event will have probability equal to zero, as we know from *Probability Theory*).

As said before, the main target of Game Theory is to maximize the utilities following the best policy as a strategy (it depends on the solution concept chosen, different solution concepts have different selected policies in the general case).

Here are listed some examples of **solution concepts**[7]:

- *Nash equilibrium*
- *Dominant strategy*
- *Pareto optimality*
- *Social welfare*
- *Random choice*
- *Backward induction*
- *Forward induction*
- *Perfect Bayesian equilibrium*

Typically, there is also uncertainty over the modeled values in the payoff matrix, the same kind of uncertainty called *Type 2 Uncertainty* previously in the Fuzzy Logic part. Note that are available extensions of Game Theory that try to handle also this specific (and very general) situation.

[7] Find the best **solution concept** for a specific (real world or theoretical) problem is a relevant area of scientific research. Let us just consider that from 1970 (with Paul A. Samuelson) to our days twelve *Game Theorists* won the **Nobel Prize**. The most famous (and influent) scientist of this list is probably John F. Nash Jr. who won the Nobel Prize for Economics in 1994. A little curiosity: from his personal story is derived the famous film *'A Beautiful Mind'* (directed by Ron Howard, 2001).

2.6 MAPE-K Adaption Control Loop: an example of self-adaptive system

"It is not the most intellectual of the species that survives; it is not the strongest that survives; but the species that survives is the one that is able to adapt to and to adjust best to the changing environment in which it finds itself. So says Charles Darwin in his 'Origin of Species'."

Leon C. Megginson, Key to Competition is Management, (1964)

We want to understand the mechanism under the self-adaptive systems. In order to understand the conceptual idea, let us present a famous example of *Self-Adaptive system*, the **MAPE-K Adaption Control Loop**. The original idea was introduced by IBM in the paper '*An architectural blueprint for automatic computing'* [*IBM, 2005*]. A similar idea was previously discussed in the field of *self-adaptive systems* by Kephart and Chess in their work *'The Vision of Automatic Computing'* (2003) [*Kephart and Chess, 2003*]. Later on the idea of the controller (originally introduced by IBM) was discussed in the field of **self-adaptive systems** (as in our case of interest) by Brun et al., in their work *'Engineering Self-Adaptive Systems Adaptive Systems through Feedback Loops'* [*Brun et al, 2009*].

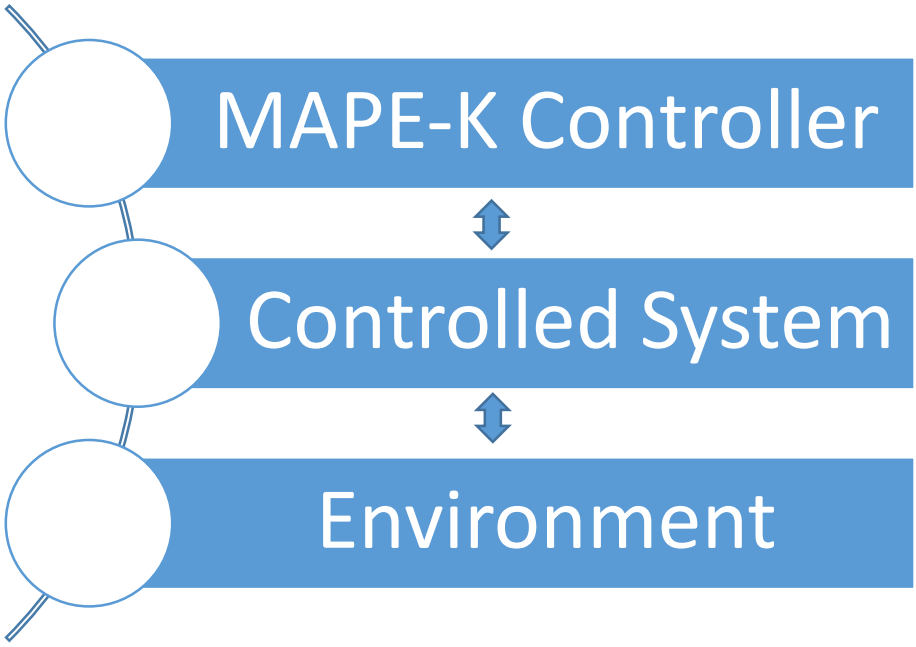


Figure 13. *MAPE-K* conceptual schema (with the three main components).

The main idea of the *MAPE-K Controller* is to build a computing environment with the ability to manage itself and dynamically adapt (run-time) to change in accordance with business objectives and encountered problems, dealing with real world events. Therefore, the system adapts itself to new situations in order to achieve the desired behaviour (its objective).

The concept of the *MAPE-K Adaption Control Loop* is a dynamic change management (both in the phase of planning and in the phase of controlling).

From a conceptual point of view, the *MAPE-K Adaption Control Loop* can be seen as a *system* (with a *controller*) interacting with the *external environment*. The three main modules are then:

- *The Managing System (MAPE-K self-adaptive controller)*
- *The Managed System*
- *The External Environment*

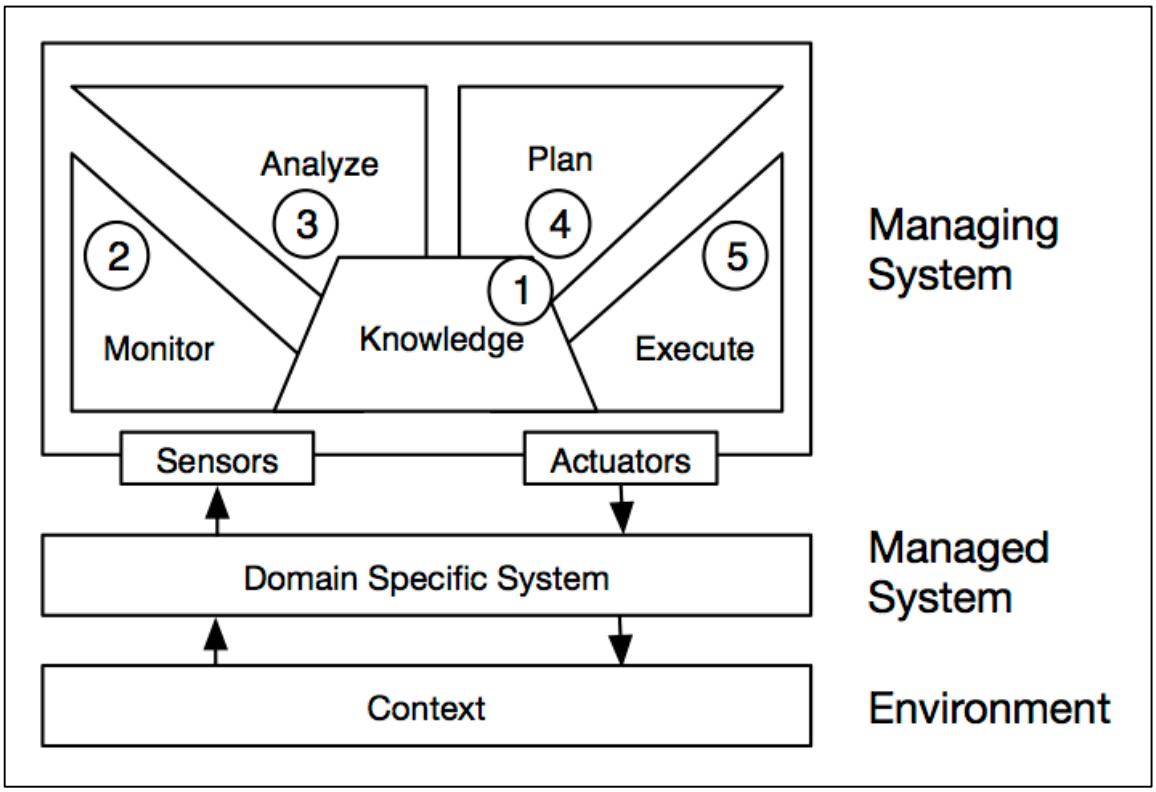


Figure 14. *MAPE-K Controller* conceptual schema with its internal modules (*Knowledge, Monitor, Analyze, Plan, Execute*).

After this graphical representation, let us 'open the box' of the controller in order to understand its functional modules and their specific role in order to guarantee self-adaptiveness.

Let us present in details the **modules** of the *MAPE-K*:

- **MONITOR**: it collects the details (metrics, parameters, other information) from the manages resources, typically acquiring sensors' data. This module aggregates, correlates and filters these details until determines a symptom that needs to be analyzed (from the *Analyze* module).

- **ANALYZE**: given specific algorithms, it performs data analysis and reasoning in order to detect problems related to symptoms provided by the monitor function of the conceptual model. It is influenced by data stored in the *Knowledge* module (we will explain it later on). On the basis of the performed analyses, if changes are requested in the actual model, a *change request* is submitted to the *Plan* module.

- **PLAN**: given the objective behaviour of the system, it structures the actions needed to achieve goals and objectives. It creates or modifies specific procedures in order to handle the alteration detected before. It can plan different functions with different degrees of complexity (from a single action to a very complex workflow, it really depends on the application domain and the specific detected anomaly).

- **EXECUTE**: it changes the behaviour of the controlled system using the effectors (also called actuators, especially in the field of robotics/automation). This action reflects the orders received by the *Plan* module.

- **KNOWLEDGE**: it collects data shared among the other modules (*Monitor, Analyze, Plan* and *Execute*). There is a wide range of data that may be stored in this module (topology information, historical logs, metrics, policies, symptoms, association rules, APIs, events, configuration files and so on). It is continuously updated runtime; more knowledge often implies better solutions to run-time problems (more context-awareness in practice).

What we can say about knowledge and uncertainty management? Let us present a specific graphical representation of the *MAPE-K* controller in order to show the sources of uncertainty, highlighted in red (for the environment, the managed system and the managing system):

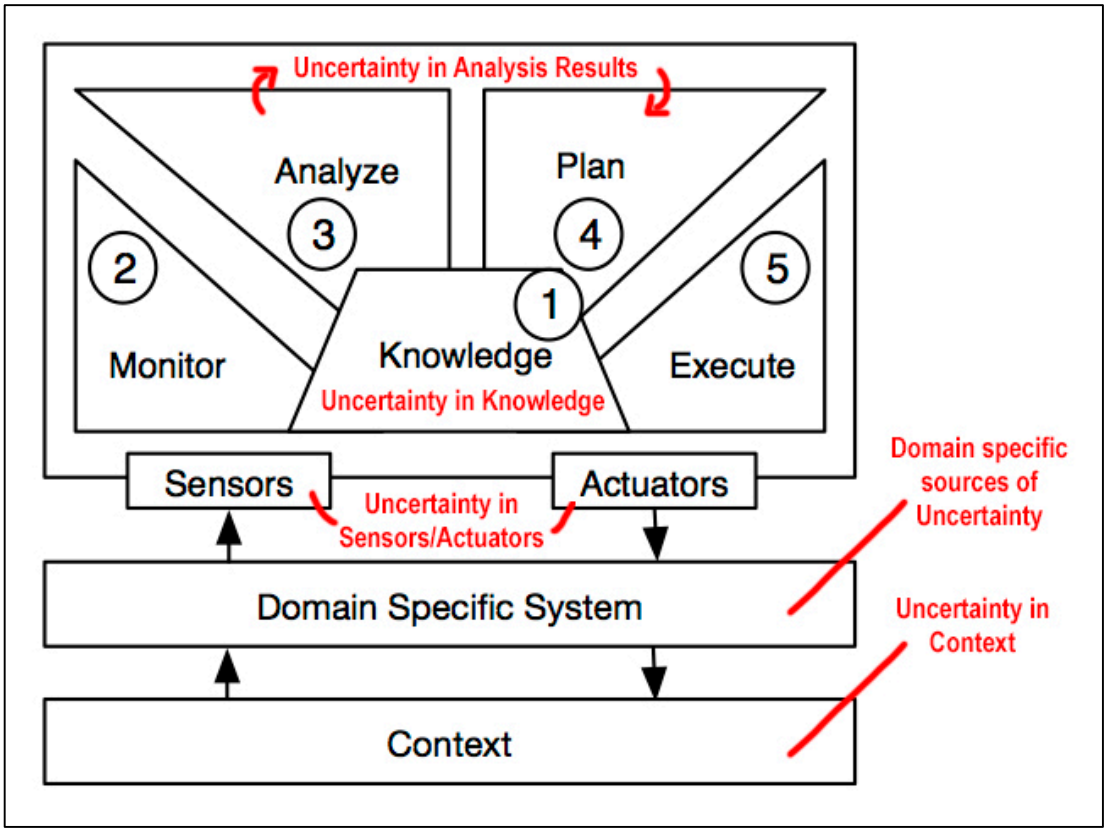


Figure 15. *MAPE-K Controller* conceptual schema with uncertainty sources highlighted in red.

With respect to the sources of uncertainty, we want to highlight the *Knowledge module* (uncertainty already in that box by design limits of real-world systems) and the step between *Analyze* and *Plan* (uncertainty in analysis results, clearly the analysis performed is not always perfect). This section about the *MAPE-K* can be used a blueprint as a general reference for further analyses. In order to be complete, let us present also the other sources of uncertainty to be considered in the general case. These are the ones related to the environment itself (context), to the domain specificities and in some cases also to the sensors/actuators of the controller, see **[Figure 15]**.

Chapter 3 - Problem statement and preliminary analysis

3.1 Problem statement

Let us present the real world problem that we are going to study and analyze in the next chapter, trying to understand how to deal with the different forms of uncertainty with specific techniques. Here is presented the problem with the original statement from the [*D. Perez-Palacin, R. Mirandola, 2014*] paper:

"Consider a software application whose functionality is the viewing of video in streaming (real-time events, films, etc.). To meet its mission, it requires services that are offered by third-party service providers over the Internet; e.g., streaming video servers. The mission is accomplished when the user can watch the complete video, whose expected duration is published and hence it is known by the application in advance. As there may be multiple providers for each required service, to increase the application's quality, it will be engineered with ***self-adaptive capabilities in terms of dynamic service provider selection****; e.g., if it is using a service provider and it becomes unavailable, the application will be able to autonomously zap to bind another different provider. Let us assume that there are N third party-providers, named* sp_n *such that* $n \in [1..N]$*. [...] In this example, the software application resides in a mobile device and can connect to the internet using several access media and protocols. Let us assume that there are M access media, named* am_m *such that* $m \in [1..M]$*.* ***The application can adapt its behaviour for availability reasons*** *(the kind of network that it is using and the third-party server that is requested to execute the service). If the user starts watching a stream and it disrupts, he will be dissatisfied with the application."*

The following **[Figure 16]** illustrates in a graphical way the problem: starting from the left we have the user side (with smartphones and/or tablets and/or similar mobile devices), then the possible access points (router Wi-Fi and Antenna), the network (Internet) and in conclusion on the external right side the servers of the service providers. Note that we will use a model with one mobile device, two access points and three service providers for our experiments in the next chapters.

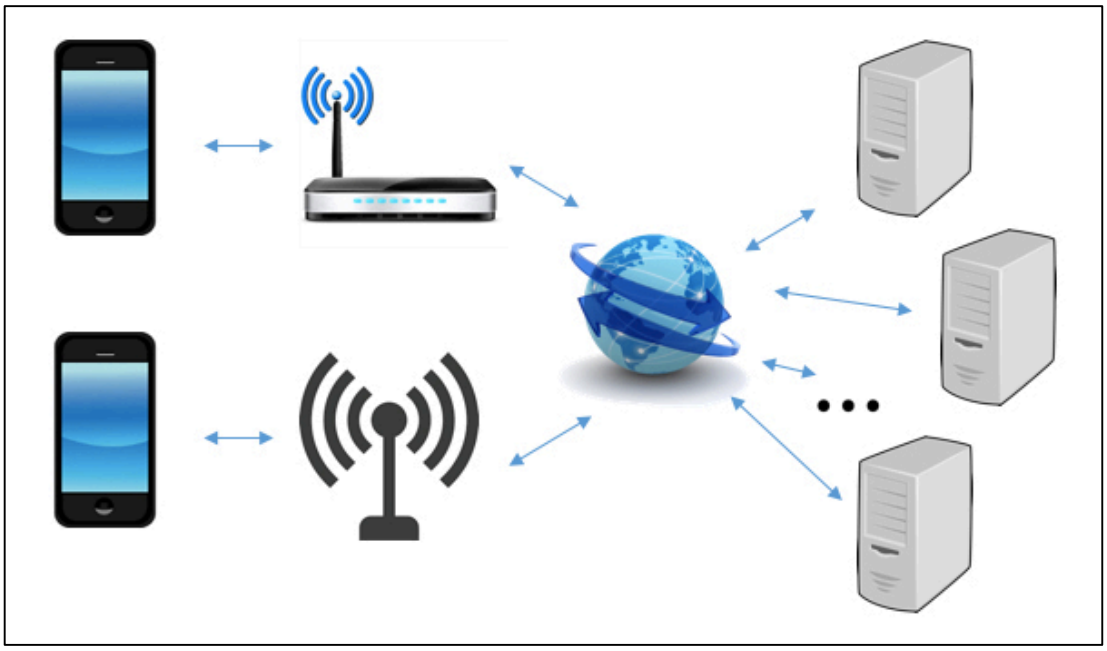


Figure 16. Real world problem graphical representation.

Given that we will use the idea of MAPE-K Controller, the availability models of the self-adaptive application are stored into the **knowledge module** of the application. A simple possibility may be to implement an *IF-THEN(-ELSE) Rule-Base* that manages the controlled system according to the observed availability. This approach is typical of similar problems in the area of Artificial Intelligence called *Knowledge Engineering*. Knowing that our main goal is to understand the availability of the considered system (with the usage of *A.I.* techniques) we will go in detail in that direction. The study is performed according to *Location* dimension (*Input Parameters, Structure* and *Context*). Given that real data are used the example is then realistic.

The logs used in this work as reference are the same presented in the original paper regarding the availability of different internet servers. They are presented and explained in [*Bakkaloglu et al. 2002; Bolosky et al. 2000; Guha et al. 2006; Pang et al. 2004; Stribling 2005*] and collected together into the integrated repository [*Brighten Godfrey, 2010*].

The part concerning the application of *Fuzzy Logic* will be a general integrated approach to deal with uncertainty: several possibilities will be considered and merged into the final model.

Let us present the 'modus operandi'. As anticipated before, for each of the three macro-categories of the *Location* dimension of uncertainty (*Input Parameters, Structure* and *Context*) the uncertainty analysis will be performed both for the Aleatory sources of uncertainty and for the Epistemic ones.

- **Input parameters**: Aleatory and Epistemic Analyses;
- **Structure**: Aleatory and Epistemic Analyses;
- **Context**: Aleatory and Epistemic Analyses.

It is clear that in a real world example like this there may be several sources of uncertainty for each field of analysis. In our case we will treat the original sources of uncertainty presented in the original paper.

3.2 Preliminary analysis

Let us try to understand what we are going to do in the next chapters, starting from the analysis performed in [*D. Perez-Palacin, R. Mirandola, 2014*]. First of all, we start analysing which are the different forms of uncertainty in the model that can affect the availability of the system. Secondarily, we study how they are managed in the original paper and how we can manage them alternatively.

- **Input Parameters (Aleatory Nature)**: the first analysis performed on the original paper concerns the classical availability identification, according to the block diagram decomposition. Every component in the aforementioned real world problem graphical representation can be seen as a block, with a status (*Available* or *Not Available*) indicating whether the component is available in a given temporal instant. Collecting data for a time interval, we can infer the probabilities for each component and then determine the availability model in terms of mathematical computations. We are dealing in this case with Input Parameters uncertainty (the availability of the components and the overall system in the future) and with probabilities, therefore the nature is inherently aleatory. Similar situations are very studied in literature; the most accepted probability model is the *Bernoulli trial within a Bernoulli process* used for the accessibility of an element in a certain moment, as in our case of interest. The formulation on the original paper recognizes three macro-blocks (related to the mobile device, the accessing media and the third-party provider). The availability computation is done through the classical

series/parallel block diagrams mathematical approach. Here is presented an example of block diagram, as in the original paper, for completeness purposes. Note that the probabilities of *Status=Available* for each component are reported into the blocks.

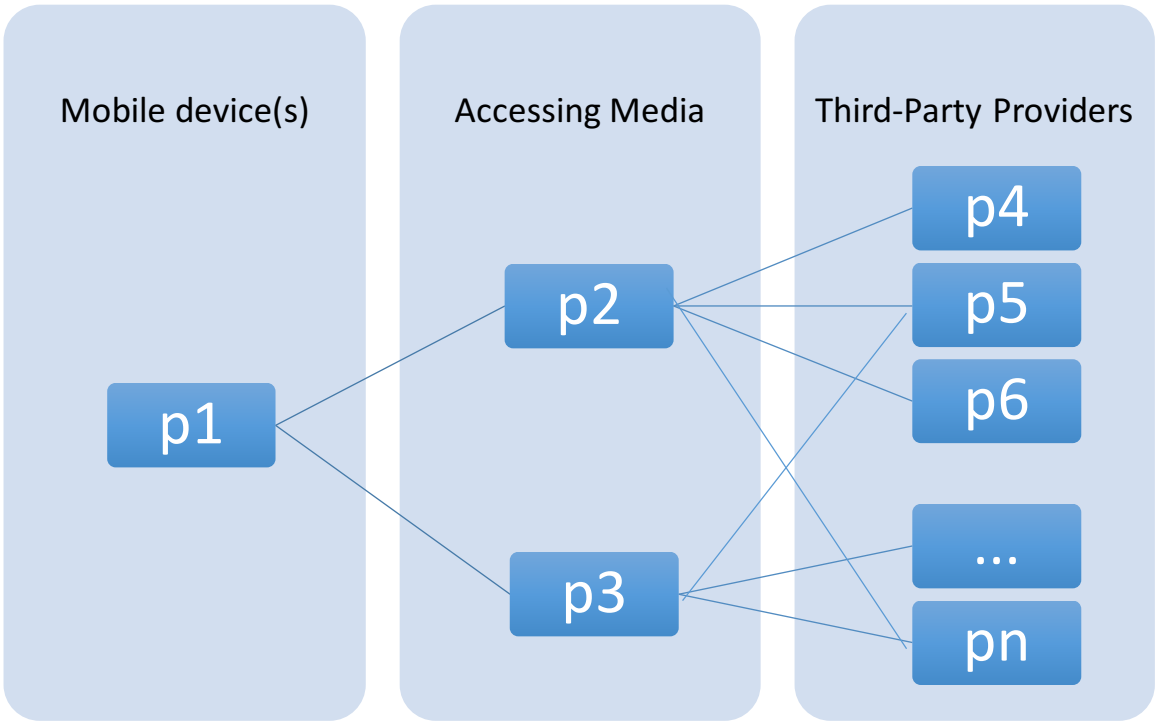


Figure 17. Probabilities of *Status=Available* mapped into a classical block diagram for the considered system.

- **Input Parameters (Epistemic Nature)**: after the analysis of the aleatory nature concerning the input parameters uncertainty, it is the moment to investigate which is the cause of the epistemic uncertainty and how it is tackled in the original paper. The aforementioned probabilities $\mathbf{p_i}$ are not given *'a priori'*, but are the results of measures over the status of the components during the considered observation time (or components lifetime). The source of uncertainty is the following: all data concerning the providers are collected with the underlying assumption that each log contains data regarding the steady-state behaviour of uptimes and downtimes. Unfortunately, these data can be biased or can contain errors. In fact, the real steady-state availability behaviour of the service providers can be different from the one inferred from the logs. This may happen for several causes, but let us present a simple example. The availability of a

service provider may be very low for a software/hardware bug. But after the identification and the resolution of the bug, the future availability of the provider will be notably higher than the one given by the data. Of course, we can give also an opposite example. For example, a server with a long lifetime, that does not receive more maintenance from external sources, is likely to reduce (also in a consistent way) its availability in the near future. In this case there is not a problem with the randomness of the parameters as before. Here there is simply lack of knowledge regarding the interpretation phase of the data collected from the providers, used in order to build the availability model.

In order to tame the uncertainty, there are as usual several strategies (*Sensitivity Analysis, Confidence Intervals* and son on). The proposed solution recalls the idea derived from the military strategy by Gaius Julius Caesar, explained in his book (*De Bello Gallico*) and known *as 'Divide Et Impera'*. Note that in literature there is also a second name for the technique: it is '*Divide and Conquer' (D&C)*. Technically speaking, in computer science is an algorithm design paradigm based on multi-branched recursion. In practice, a divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

In our case we will not deal explicitly with recursion and recombination of different solutions (from sub-problems), but we will perform only an analysis over the (temporal) fragments of the initial temporal observation time. In fact the main idea is to divide the temporal interval available from the collected data in more intervals and see what happens to the availability. In some case it will be stable, but in other (more interesting) cases there will be a strong variance (and therefore a considerable variation in the availability). This as anticipated before can happen for several causes, from a transitory state (for example at the very beginning) to a bug that afflicts the service provider (periodically or for a certain interval of time).

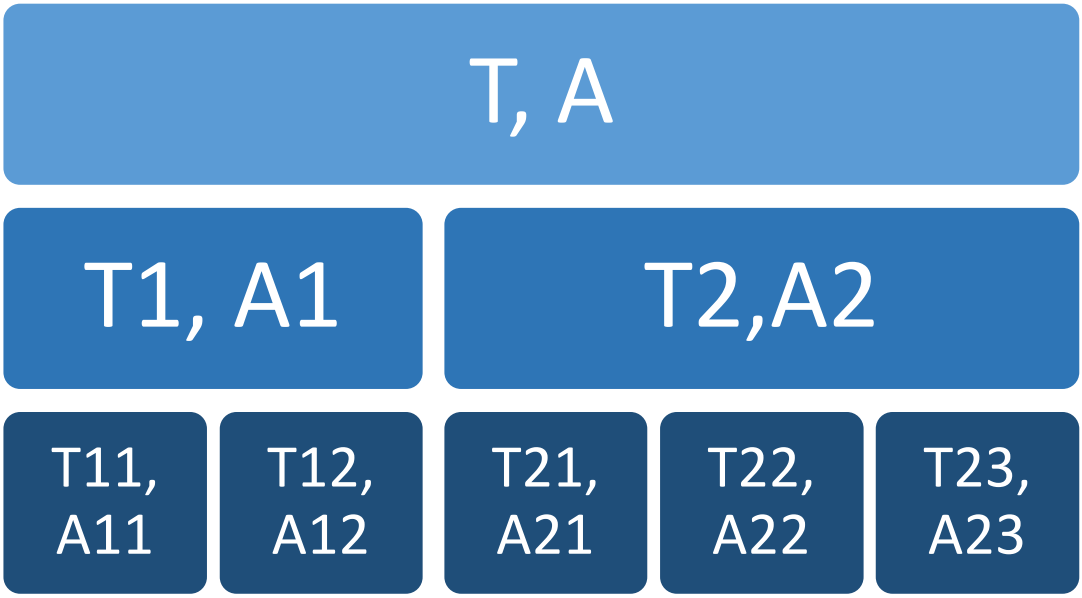


Figure 18. The first time interval (T) with its own Availability parameter (A). According to the *Divide Et Impera* technique, each interval is divided in two or more intervals and the availability of each sub-interval is computed.

- **Model Structure (Aleatory Nature)**: let us move on the aleatory uncertainty in the model structure. Given that the system operates in the real world, with real service providers, it is clear that from one hand new service providers that offer the same service may appear at a certain time instant. From the other hand, instead, existing service providers may disappear for various reason at a given moment. The appearance and the (possible) disappearance of a provider can be seen as two time units on the time axis. The main problem here is that we compute the availability also considering providers with a calculated availability that no longer exists. In fact, the appearance and the disappearance of the service providers depend on third-parties' decisions, external events and also purely random events The key aspect here is to understand that the anomalous situations are the ones where the providers disappear for a certain reason. A long disappearance (for example one week) may be a symptom of service provider demission. Clearly, with this assumption we can have prediction errors (*false positives* and *false negatives*). The false positives regard the service providers that are going to come back on, the false negatives regard the service providers left some hours or days before the observation time but are not going to work again in the future.

Regarding the *model averaging technique*, the modus operandi is the following: build several models with different assumptions and then integrate them all into a final model.

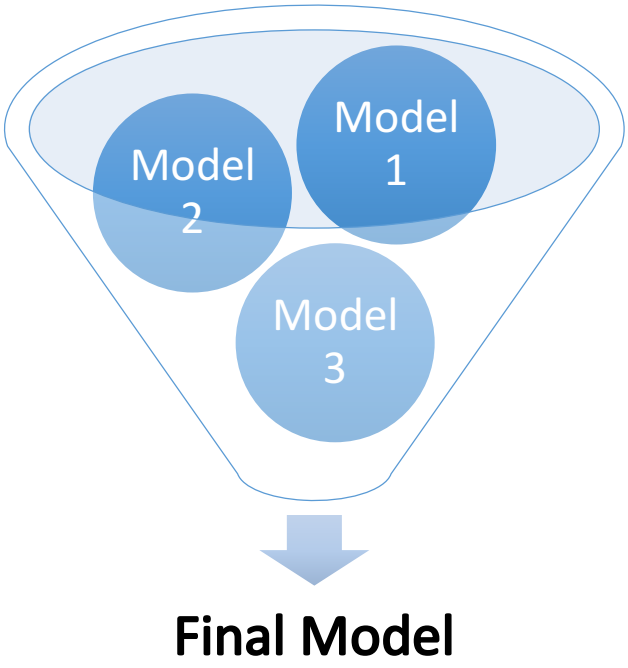


Figure 19. Model Averaging technique with three auxiliary models as stated in the [*D. Perez-Palacin, R. Mirandola, 2014*] paper.

- **Model Structure (Epistemic Nature)**: the considered application has two access points in parallel, the Wi-Fi and the Antenna. From the point of view of the infrastructure, there can be failures for several reasons. For example, there can be problem with the Internet Access Point of the providers, with the DNS servers or with the traffic router on the communication path. When a service is not available, it can be for these kind of problems instead of a malfunctioning of the same provider. Another problem is that often more providers go down at the same time. This dependence in crashes may be related to several causes, for example they can share the same computing infrastructures (typical situation, especially nowadays with the *Cloud Computing*). From an intuitive point of view this source of uncertainty has an epistemic nature: we just cannot model the adequate degree of knowledge. In order to face off this source of uncertainty, a possible approach (as suggested in the paper) is to study the likelihood of concurrent unavailability of media access services (and

so the dependence in crashes). This can be done through several instruments, from mathematical formalisms to original approaches based on Neural Networks. From a practical point of view, we know that there is this source of uncertainty and we have to consider it in the design of the self-adaptive systems.

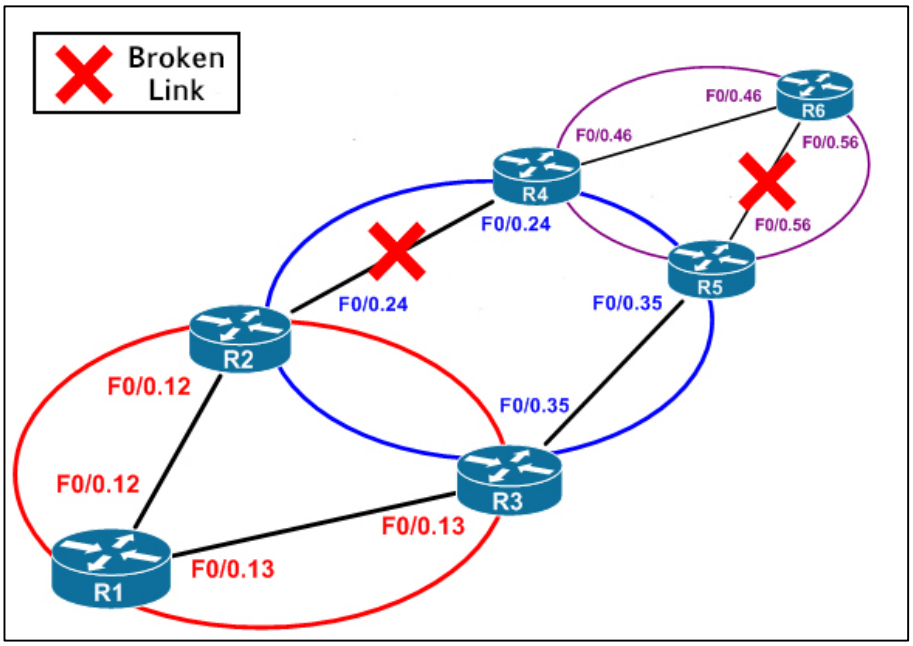


Figure 20. An example of routing path with two broken links, highlighted with the red cross. In principle, they can have also dependences.

- **Model Context (Aleatory Nature)**: after the model structure, let us focus on the uncertainties related to the model context. In particular we want to deal with *Location Context* uncertainty, which afflicts the modelling of the system and its boundaries (third party providers, accessing media and mobile devices). From the point of view of the application user there is a problem related to the battery life. It is clear that in order to satisfy the streaming demand the smartphone must have enough battery. There are methodologies that aim to predict the battery life given the actual situation, the usage history and other exogenous factors. But in any case, there is still an aleatory factor, the probability of satisfying the streaming mission with the given battery. Modern smartphones have an interesting function from this point of view: they show a pop-up when the battery level is low, requesting to use a battery charger in order to accomplish the tasks. Hence, we have to consider the possible usage of the battery charger from the point of view of the user. After all, we have to remember that the

human behaviour is not completely deterministic**:** in fact, everyone acts with its specific thoughts. In practice, there is an aleatory component in the human nature and in its behaviour. But then is the real question: how we can manage this kind of uncertainty? There are several methodologies that aim to model the models' boundaries and handle the related sources of uncertainty; it is often done with the usage of equations, specific algorithms or slack variables (similar to the ones used in linear programming). But now let us present the same problem from a more general point of view, trying to identify other possible approaches (especially in the case of self-adaptive systems). First of all, an important consideration: energy is one of the most problematic bottlenecks in smartphone systems. Now, knowing the status of the battery lifetime and being able to use it efficiently is an important requirement from users. At this point we want to model a system context-aware approach for predicting battery lifetime, which allows a user to know the accurate battery status and to utilize the power efficiently. We change our perspective in this modelling phase, pushing towards another location form of uncertainty, in this case the battery can be seen more as an input parameter (and therefore that is the source of uncertainty). Obviously also here we have to deal with different sub-form of uncertainties: from the influence of *external exogenous variables* (for example the temperature, especially during hot days in summer) to *different profiles of usage* (gaming, texting, watching a video…). The profiling phase is crucial here: as the intuition suggests each profile of usage will reflect a different battery consumption and hence a different predicted battery life. We refer to a collection of system component states as system context and model the quantitative relation between system context attributes and the battery discharge rate by **multiple linear regressions**. When the user changes applications or operations, the idea is to dynamically predict the remaining battery lifetime as well as its variations by monitoring system context attributes. Clearly, it is possible to adopt a feedback mechanism like the one presented before (the *MAPE-K Controller*). Once the system is implemented, it is important to see the results in the phases of testing and validation. The experiments must show that the model describes how the changes of system component states affect the battery lifetime, and that it improves the accuracy of online battery lifetime prediction. An accurate prediction will be an important blueprint in order to understand if a mobile phone can achieve its mission with the available quantity of battery (view a video on *YouTube*, use *WhatsApp* or other types of typical applications). From the state of the art point of view, there exist several

applications that work as battery life logs and predictors. Some examples are *Google Pixel* and *Google Pixel XL* for Android and *Battery Life* and *Battery Magic* for iOS.

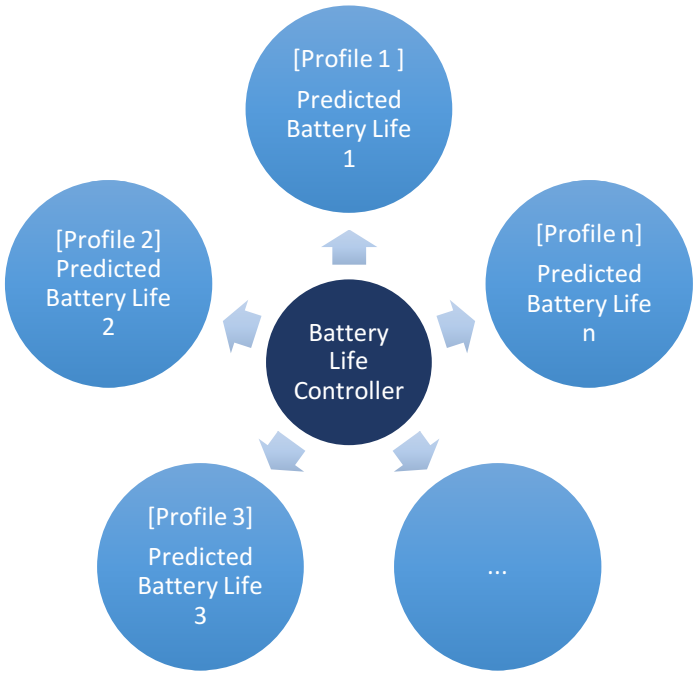


Figure 21. Profiling diagram with the central *Battery Life Controller* and the different usage profiles (with their predicted battery life).

- **Model Context (Epistemic Nature)**: last but not least, let us try to understand the epistemic source of uncertainty related to the model context. Up to this moment, we have not considered possible failures and errors related to the Third Party Service Providers and their computing infrastructures. But it may be the case that they can afflict the overall availability with malfunctioning problems. The situation is similar to the one reported in the epistemic case of model structure, but here we have not information about the computing hardware of the internet service providers. As it is possible to derive, the uncertainty here affects the boundaries of the model not considered in the model itself, therefore we speak of epistemic uncertainty in the model context category. The interesting situations are where the providers become concurrently available or unavailable. This dependence is related to the existence of the considered uncertainty form. Once again, a possible cause of this may be a shared computing infrastructure between the providers. At the end the study over the variation shows exactly this fact: there is a form of

dependence within the providers. The following **[Figure 22]** give us an idea of dependence between providers within the same geographical area (*Asia, Europe, North American, Africa, South America, Australia*). The providers are represented as **nodes** of a **connected graph**, with different colours for the six geographical areas**.** In case of dependence, for example, a failure in a specific geographical area may afflict at the same time all the providers in the same area.

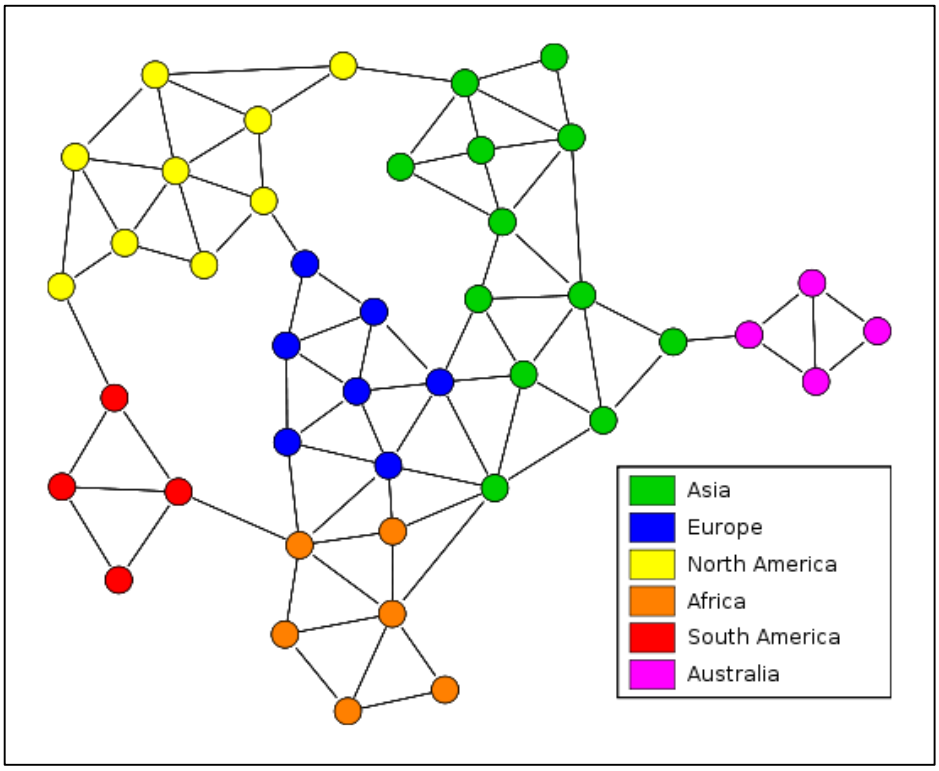


Figure 22. Example of a *dependence graph* with different geographical areas. The providers are represented with nodes, the geographical areas with colours.

Chapter 4 – Availability computation with Fuzzy Logic

4.1 Fuzzy Logic and Uncertainty: the approach

After the conceptualization and the descriptions of the previous chapter let us present a possible alternative and integrated approach to compute the overall availability of the system. The main idea here is to make use of the techniques typical of the Artificial Intelligence (in this specific case with the Fuzzy Logic) in order to establish the overall degree of Availability taking in consideration the different sources of uncertainties (analysed in the previous chapters). The acting strategy will be the same for the three macro-categories of uncertainty (and related availability). In fact, the sources of uncertainty will be opportunely mapped into *Fuzzy Sets*, with their *membership functions*. With a fair application of the fuzzy rules, a final level of availability will be estimated (with an output variable). At the end, the three availability results will be condensed (also here with fuzzy rules) into the computation of the *Overall Availability,* with an **iterative approach**. Considering that we are dealing with sources of uncertainty different by nature and impact over the overall systems, some assumptions and approximations will be made in order to compute the final results. Note that this approach is typical of situations where data/system integration is needed.

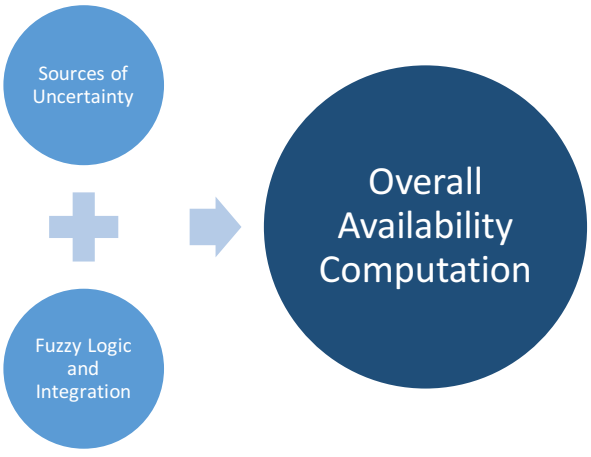


Figure 23. Conceptualization of the integration of the sources of uncertainty through Fuzzy Logic in order to obtain the *Overall Availability* of the system.

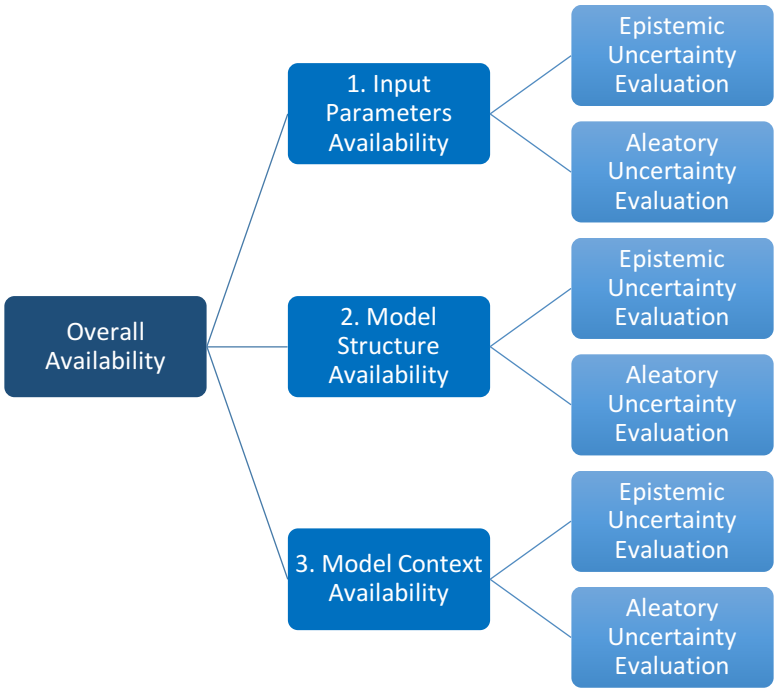


Figure 24. *Divide et Impera* principle applied to the overall availability computation through the usage of Fuzzy Logic.

Let us discuss now some technicalities about the approach that will be used in the following paragraphs. A critical step in the fuzzy logic is the fuzzification process, where we start from the real world domains and values and we go in the direction of fuzzy membership functions, with their specific shapes. There are a lot of possible shapes for the membership functions: such as *trapezoidal, triangular, Gaussian, generalized bell* and so on. Each one has its advantages and disadvantages (and can suit or not a specific factor of uncertainty). Our approach will be systematic and precise: every membership function shape will be justified for the specific problem. Regarding some unity of measures, if needed, a normalization step will be performed as consolidated approach in the field of Fuzzy models. From an operative point of view, all the analyses will be performed using the *Fuzzy Logic Designer* application offered by the **MATLAB toolbox**. There are two main *Fuzzy Inference Systems (FIS)* supported by *MATLAB*. These are the most famous in literature and the used world-wide:

- **Mamdani** Fuzzy Inference System (FIS)
- **Sugeno** (also known as *Takagi-Sugeno-Kang* or *TSK*) Fuzzy Inference System (FIS)

Note that the so-called Sugeno, or Takagi-Sugeno-Kang (or even TSK, as acronym), method of fuzzy inference was introduced in 1985 and it is similar to the Mamdani method in many respects. The first two parts of the fuzzy inference process (that we have seen in **[Chapter 2.3]**), fuzzifying the inputs and applying the fuzzy operator, are exactly the same. The main difference between Mamdani and Sugeno is that the Sugeno output membership functions are either linear or constant. A typical rule in a Sugeno fuzzy model has the form:

IF Input 1 = x AND Input 2 = y, THEN Output IS

$$z = f(x,y) = ax + by + c$$

For a zero-order Sugeno model, the output level z is a constant (a=b=0), therefore z = c. We will not enter into further details of the FISs; it is sufficient to know that we are going to use the first one (*Mamdani*, default FIS in *MATLAB*).

Regarding the mathematical formalisms of Fuzzy Logic, the default settings will be used for the operators. We will use in detail the following settings:

- *And method: Min operator*
- *Or method: Max operator*
- *Implication: Min operator*
- *Aggregation: Max operator*
- *Defuzzification: Centroid operator*

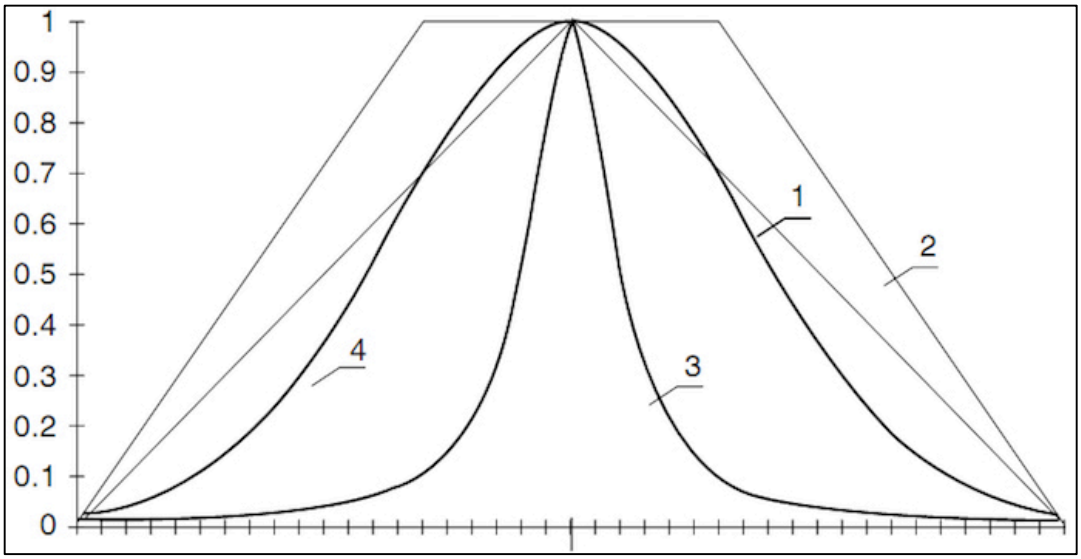


Figure 25. Some classical membership function shapes.
(1) Triangular, (2) Trapezoidal, (3) Gaussian, (4) Generalized Bell.

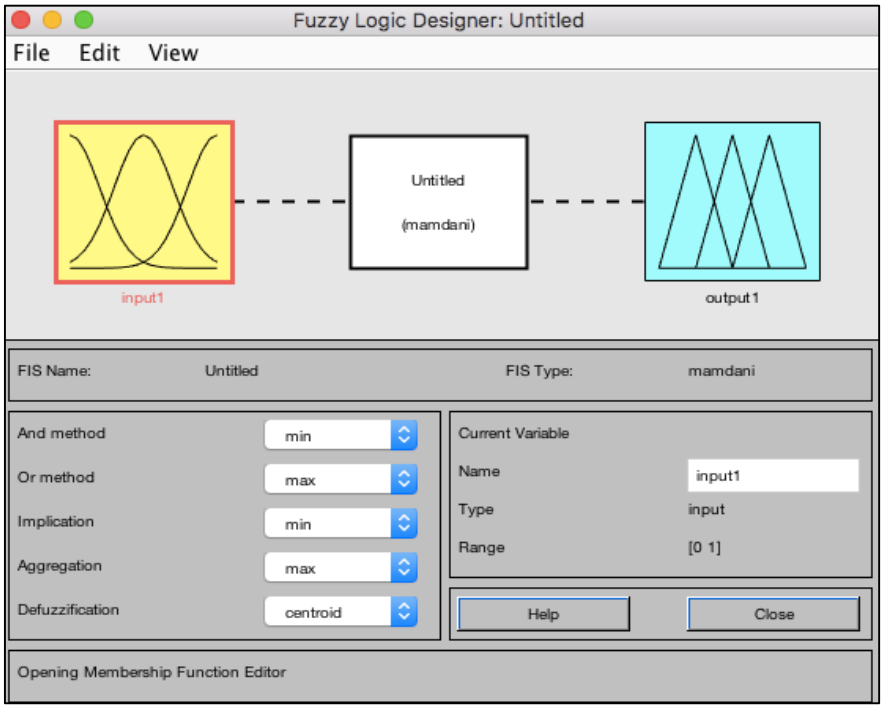


Figure 26. *MATLAB Fuzzy Logic Designer* GUI (Graphical User Interface). On the left the part corresponding to the input variables, on the right the part corresponding to the output variables and the chosen FIS (Mamdani) in the central white box.

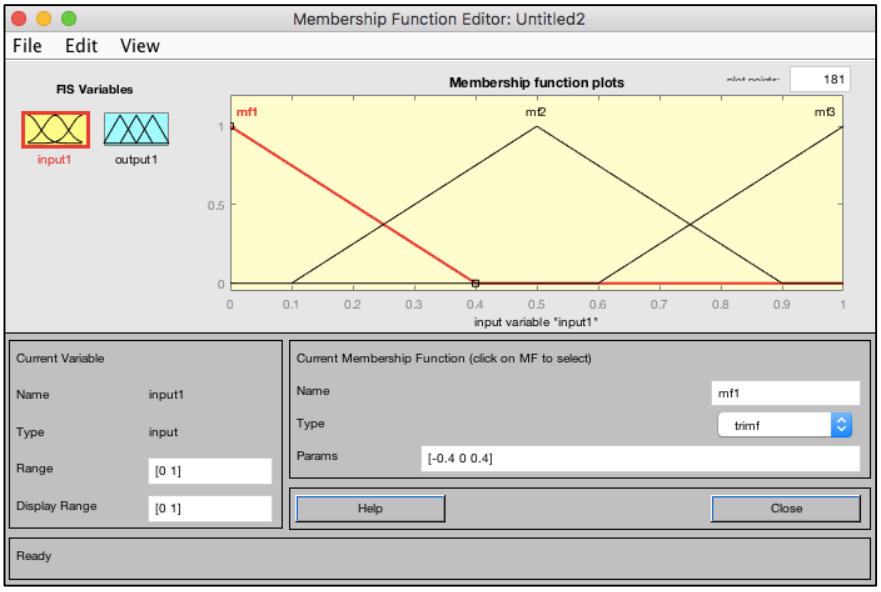


Figure 27. Detail of the *MATLAB Fuzzy Logic Designer Membership Function Editor*, with some sample parameters.

We want to underline that in the Fuzzy Section we will deal with sources of uncertainty related to *noise in sensing, decentralization* and *granularity of models*, according to the taxonomy proposed in **[Figure 7]**. For the real world data, we will refer to the ones presented in [*D. Perez-Palacin, R. Mirandola, 2014*]. In particular for our case study we will use the availability traces extracted from the log file *web-sites.avt* in [*Bakkaloglu et al. 2002*] (and available also in the integrated repository [*Brighten Godfrey, 2010*]). As done in the original reference paper, we will use the **availability traces** that were created from monitoring the servers in *lines 2,3,4,5,7,8* corresponding to servers:

- *asia.cnn.com;*
- *canberra.yourguide.com.au;*
- *digital.library.upenn.edu;*
- *games.yahoo.com;*
- *mail.yahoo.com;*
- *msdn.microsoft.com.*

Let start the discussion on the values to be used in the Fuzzy Analysis. Given that some parameters (for example the battery life) are not known (we have not real world data for it, they are more context specific), in the *Overall Availability* computation **[Chapter 4.5]** we will perform three different iterations of the process with different parameters in order to handle more cases. Now it is important to understand which are the specific considered **values** for the input fuzzy variables related to the aleatory/epistemic sources of uncertainty. Let us present them with reference to real world data in the next sub-chapters.

4.2 Input Parameters Uncertainty and Availability

Let us start the discussion over the influence of the sources of uncertainty related to the input parameters uncertainty and their effect over the availability. There is a best practice in the field of *Fuzzy Modelling*: use a number of membership functions between three and seven for each considered variable (the same number of concepts that a human being can manage at the same time, according to the scientists). We will go in this direction, in order to reflect the general guidelines, here and in the following subchapters.

The first input variable that we are going to consider here is the Availability computed with the classical mathematical approach over the considered system, for each of the six parts of the system (corresponding to the traces introduced in **[Chapter 4.1]**). For each specific application (with series and parallel computations) we can obtain a specific availability value. Of course, this number is strongly application-dependent. An availability value of 99,9% may be very good for a Wi-Fi access point, but can generate disasters in other 'critical' fields (such as a bank or a hospital). We will use on the axis of the abscissa values that are coherent with the considered application, little variations can be made in case of re-modelling need. From the point of view of the membership function shapes, we will use trapezoidal shapes at the edges in order to model a little bit of indifference, while for the central membership functions we will use triangular shapes. In some cases, we will use the Gaussian membership function in order to model smoothness.

Note that our approach is just one of the possible Fuzzy approaches; in fact, for the same problem it is possible to define different input values or different membership functions. Here the important concept is that a considerable degree of freedom in the process of modelling is given to the system designer. Each system designer can capture different aspects of the reality and model them in a different way; this is related to the human sensitivity. The main purpose of this work is to show a possible application of the *Fuzzy Theory* (and the *Fuzzy Logic*) in order to establish the overall degree of availability of the considered real world system. Possible extensions or different versions are always a possible alternative to this work. For the moment, let us try to build and explain a first conceptualization in this chapter and in the next ones.

From an operative point of view we will structure the shapes of the membership functions as follow. The first input variable to consider is the Availability

(registered in the logs, for each of the six modules). Here the values on the abscissa axis can reflect the measured values and the membership functions can reflect the availability behaviour, with a good degree of stringency required (we need a high value for our purposes).

Regarding the second input variable and its influence the situation is a little bit more complex. In fact, it is difficult to measure and understand the precision related to the availability measure. In order to compute this parameter, we will use the **sample variance**, mapping it into the normalized measure precision. We can imagine that with an exhaustive work with the logs and other instruments we have understood the past behaviour of the measures and the error distribution (from a probabilistic point of view). With this consideration, as anticipated, we can map the normalized precision on the abscissa axis and we can select the membership functions as usual. In order to estimate the measure precision (and then compute the fuzzy normalized variable), it is possible to use the *'Divide et Conquer'* approach mentioned in **[Chapter 2.3]** considering the **variance** of the availability measured in different time intervals (formally, we will estimate the **sample variance** form our real world data). Clearly, the more the precision is high, the more the registered value of availability of the system is precise. Regarding the output, we will use a standard with two external triangular shapes and a Gaussian in the middle.

For each module of the system (mobile device, antenna, Wi-Fi and three service providers) we will extract from the logs the **availability** and the **normalized measure precision**. Then these parameters will be fuzzyfied and for each module will be computed a single availability value (which captures also the uncertainty with respect the measure precision). In order to obtain a *crisp value* as estimator for the whole system *Input Parameters Availability*, we will apply the classical series/parallel computation rules on the block diagram for our case study (see **[Figure 16]**), given the output of the six fuzzy analyses with the availability and the normalized measure precision for each component computed in the previous phase. The final output, in conclusion, is an **availability value** whose computation has taken into account the impact of the sources of uncertainties in the input parameters.

After the presentation of the variables, with their figures, we will present the rules that allow us to express the output as a function of the input values, with all the possible combinations related to the membership functions. The original tool has a nice shift-based interface that show us how the output changes according to the inputs (also with little variations, similarly to the sensitivity analysis). As a

final step, we will show here and in the next chapters the *3D output surface* of the output variable. All these components of the fuzzy analysis will be explained in the related figures. First of all, let us present in **[Figure 28]** the block diagram of the **Input Parameters Fuzzy model**. There are on the left the input variables (*MeasuredAvailability* regarding the aleatory source of uncertainty and *NormalizedMeasurePrecision* regarding the epistemic one), on the right the output variable (*InputParametersAvailability*). The white box in the middle represents the *Mamdani*-based rule box.

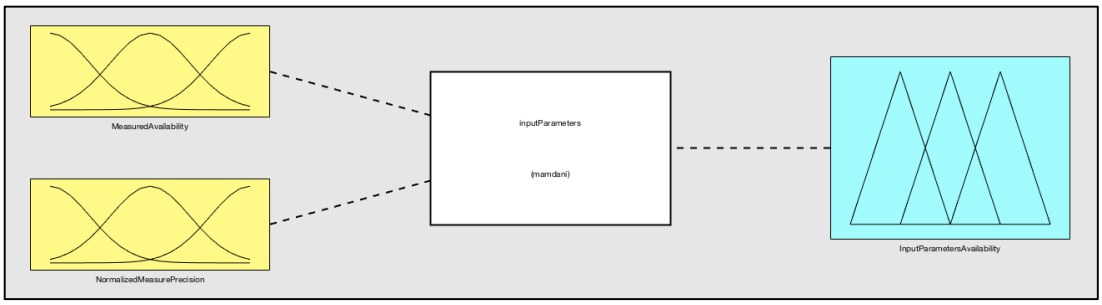


Figure 28. Block diagram of the Input Parameters Fuzzy model.

According to our case study we can measure the 'classical' availability (with respect to input parameters sources of uncertainty) at least in four different ways:

- Online (service) providers' availability ($\mathbf{A_{PROVIDERS}}$);
- Accessing media's (Wi-Fi, Antennas etc..) availability ($\mathbf{A_{ACCESSING\_MEDIA}}$);
- Mobile devices' availability ($\mathbf{A_{MOBILE\_DEVICES}}$);
- Overall case study availability (the combination of the previous availability values, $\mathbf{A_{OVERALL}}$).

We will consider the availability of **each component** (*mobile device, Antenna, Wi-Fi* and the three service providers *sp1, sp2 and sp3*). Note that on the reference paper is presented as final result the overall availability value, which takes into account overall contribution of the mobile devices, the accessing media and the service providers. It can be computed putting together the mobile device's availability, the service providers' availability and the accessing media's availability, using the following series formula:

$$\mathbf{A_{OVERALL}} = \prod \boldsymbol{Ai} = \mathbf{A_{MOBILE\_DEVICES}} \times \mathbf{A_{PROVIDERS}} \times \mathbf{A_{ACCESSING\_MEDIA}}$$

It is clear that the (performance) bottleneck of the system, from the point of view of the availability, will be the one with the lowest value of availability. If the mobile devices are strongly available, the accessing media are robust enough and the online service providers are in a great number (with warranty of continuity) we expect an overall availability value close to one. The key for an ideal functioning of a self-adaptive system should be to change reference accessing media or service provider when the performances are not satisfactory. We can manage this with specific algorithms, *if-then(-else) rules* or (static/dynamic) thresholds in the *Knowledge module.* These actions can be done on the bases of the registered performances and the available data.

In **[Figure 29]** and **[Figure 30]** are presented the shapes for the membership functions of the first input variable (*MeasuredAvailability*). There are 4 membership functions here: *VeryBad* that covers a big range (from 0 to 0.9) and models indifference with the trapezoidal shape, then there are *Bad* (from 0.8 to 0.95) and *Normal* (from 0.9 to 1) with two triangular shapes and in conclusion there is the *Super* trapezoidal shape that covers the range with higher availability (from 0.95 to 1, flat from 0.99 to 1).

Clearly these measures can have little variations with different contexts of application (but also with the same application and different reference datasets); here we underline just the approach (note that we selected likely reliable values according to our specific problem).

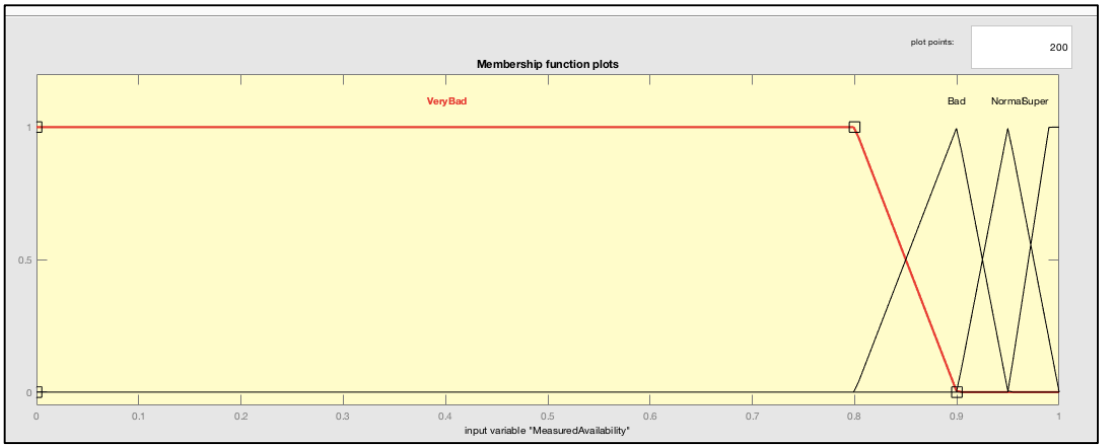


Figure 29. *MeasuredAvailability* (input variable) membership functions.

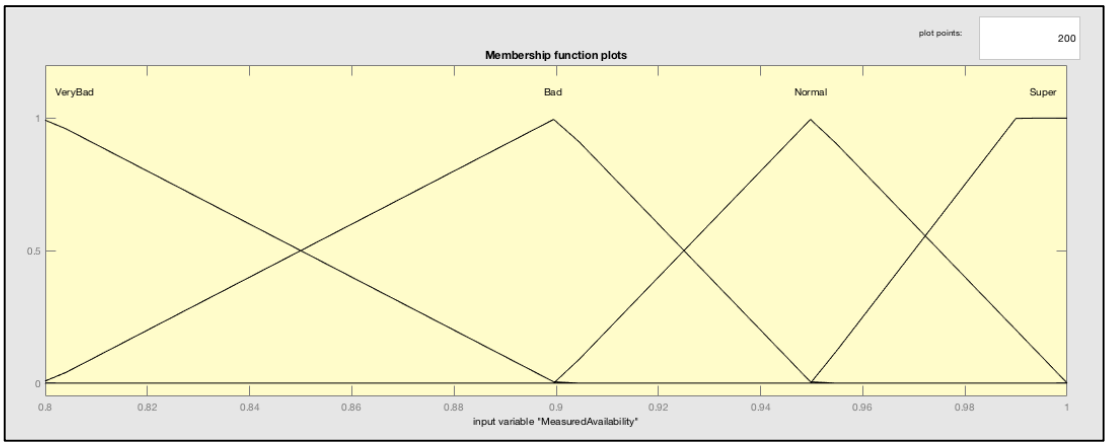


Figure 30. Here is the same input variable presented in the previous figure (*MeasuredAvailability*), with a zoom over the interval [0.8 – 1] in order to see more clearly the shapes for the chosen membership functions.

In **[Figure 31]** are presented the shapes for the membership functions of the second input variable (*NormalizedMeasurePrecision*). There are 3 membership functions here: *NotPrecise* modelled with a trapezoidal shape (from 0 to 0.6) in order to model indifference (a normalized precision under 0.6 is already not precise at all for our purposes). In the middle there is the *Precise* triangular shape ranging from 0.6 to 0.95, with a maximum in 0.9. In conclusion, on the right there is the *VeryPrecise* membership function modelled with a trapezoid (it ranges from 0.8 to 1 and here little indifference is modelled from 0.95 to 1). Also here, according to the specific requirements of the application, little variations over the membership functions and their values are possible. Note that in order to estimate the measure precision (and then compute the aforementioned normalized variable), it is possible to use the *'Divide et Conquer'* approach mentioned in **[Chapter 2.3]** considering the **variance** of the availability measured in the different time intervals. A big value of the variance suggests a low measure precision (also the opposite holds, we expect to see little variance values when the availability measures are robust and therefore meaningful). In this case for the *Fuzzy analysis* we will not consider the cumulative variance ($\boldsymbol{\sigma^2_{OVERALL}}$ or simply $\boldsymbol{\sigma^2}$)[8] of the mobile devices' availability ($\boldsymbol{\sigma^2_{MD}}$), service providers registered availability ($\boldsymbol{\sigma^2_{SP}}$) and the accessing media one ($\boldsymbol{\sigma^2_{AM}}$), but the $\boldsymbol{\sigma^2_i}$ of each module in order to compute the values for the normalized measure precision (through an additional normalization step).

[8] The **Variance** of the measured **Availability** (with the *'Divide and Conquer' approach,* see **[Chapter 2.3]**) is denoted with the symbol $\boldsymbol{\sigma^2}$, as consolidated convention in *Statistics* and *Probability Theory*.

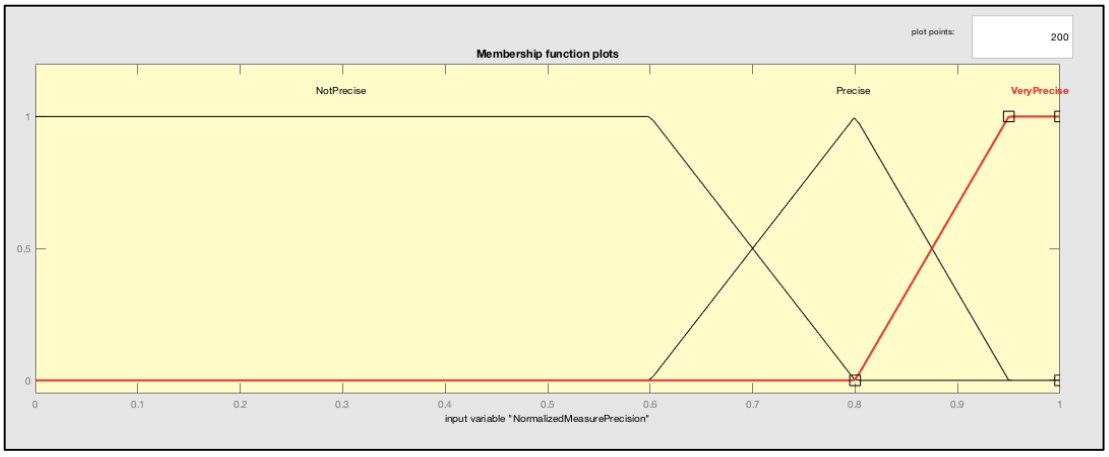


Figure 31. *NormalizedMeasurePrecision* (input variable) membership functions.

In the following **[Figure 32]** we can see the output that we are going to use also for this analysis. There are three membership functions, two triangular shapes for the external membership functions (*Low* and *High inputParameters Availability*) ranging respectively from 0 to 0.25 and from 0.75 to 1) and a Gaussian in the middle with peak in 0.5 for the sake of symmetry. Note that the values on the abscissa don't reflect the proper availability value (eg. 0.5 is not *Availability=0.5*). If we consider instead the proper availability measure, we need a massive shift of the membership functions on the left. Note that we will follow this approach also in the next chapters (it is just a possible approach that we can follow, also here alternatives exist and can give good results).

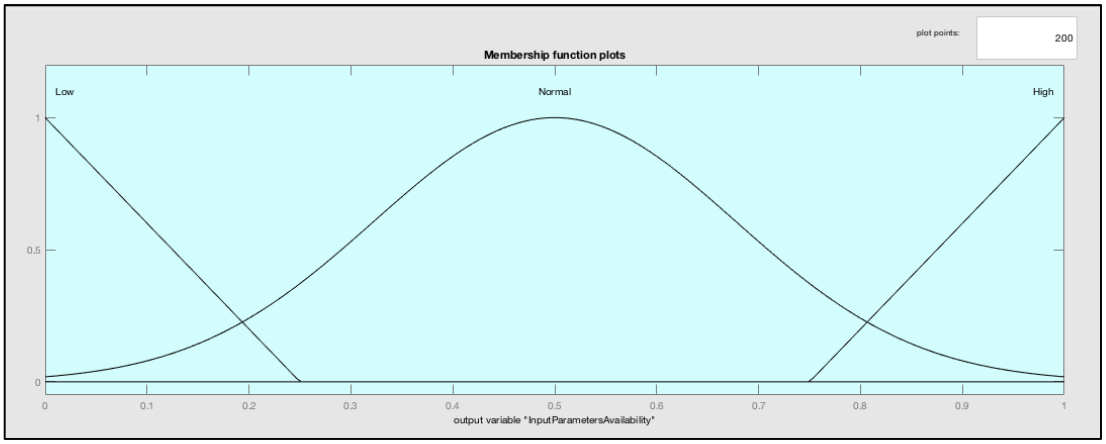


Figure 32. *InputParametersAvailability* (output variable) membership functions.

Let us now open the first **Rule-Box**[9]! In **[Figure 33]** are presented the 12 rules that combine the two input variables in order to give the output variable. Note that the major influence is related to the measured value of the Availability (the second variable, which reflect the uncertainty of the measure, has a minor impact on the final outcome).

1. If (MeasuredAvailability is VeryBad) and (NormalizedMeasurePrecision is NotPrecise) then (InputParametersAvailability is Low) (1)
2. If (MeasuredAvailability is Bad) and (NormalizedMeasurePrecision is NotPrecise) then (InputParametersAvailability is Low) (1)
3. If (MeasuredAvailability is VeryBad) and (NormalizedMeasurePrecision is VeryPrecise) then (InputParametersAvailability is Low) (1)
4. If (MeasuredAvailability is Bad) and (NormalizedMeasurePrecision is NotPrecise) then (InputParametersAvailability is Low) (1)
5. If (MeasuredAvailability is Bad) and (NormalizedMeasurePrecision is Precise) then (InputParametersAvailability is Low) (1)
6. If (MeasuredAvailability is Bad) and (NormalizedMeasurePrecision is VeryPrecise) then (InputParametersAvailability is Low) (1)
7. If (MeasuredAvailability is Super) and (NormalizedMeasurePrecision is NotPrecise) then (InputParametersAvailability is Normal) (1)
8. If (MeasuredAvailability is Super) and (NormalizedMeasurePrecision is Precise) then (InputParametersAvailability is VeryHigh) (1)
9. If (MeasuredAvailability is Super) and (NormalizedMeasurePrecision is VeryPrecise) then (InputParametersAvailability is VeryHigh) (1)
10. If (MeasuredAvailability is Normal) and (NormalizedMeasurePrecision is NotPrecise) then (InputParametersAvailability is Low) (1)
11. If (MeasuredAvailability is Normal) and (NormalizedMeasurePrecision is Precise) then (InputParametersAvailability is Normal) (1)
12. If (MeasuredAvailability is Normal) and (NormalizedMeasurePrecision is VeryPrecise) then (InputParametersAvailability is Normal) (1)

Figure 33. Rule-box concerning *InputParametersAvailability* Fuzzy outcome, starting from the input variables.

In **[Figure 34]** and **[Figure 35]** are presented the output details:

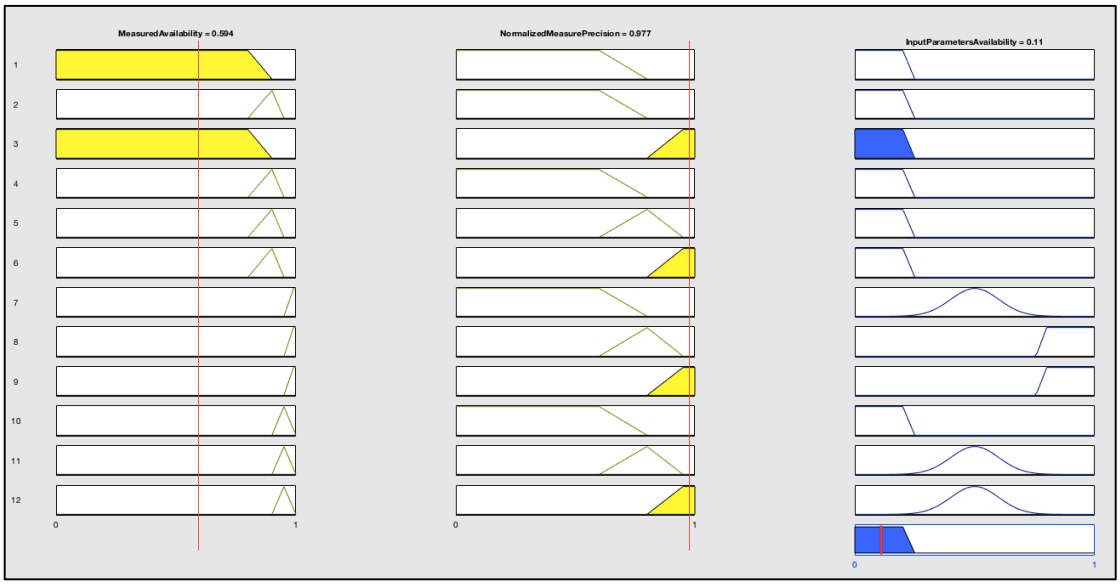


Figure 34. The shift-based interface of *MATLAB* that shows the influence of the input variables over the final output for the considered model (for the Input Parameters Uncertainty and the related Availability).

[9] With the term **Rule-Box** we refer to the module containing the list of *Mamdani rules.* They express the fuzzy output as a function of the fuzzy input variables.

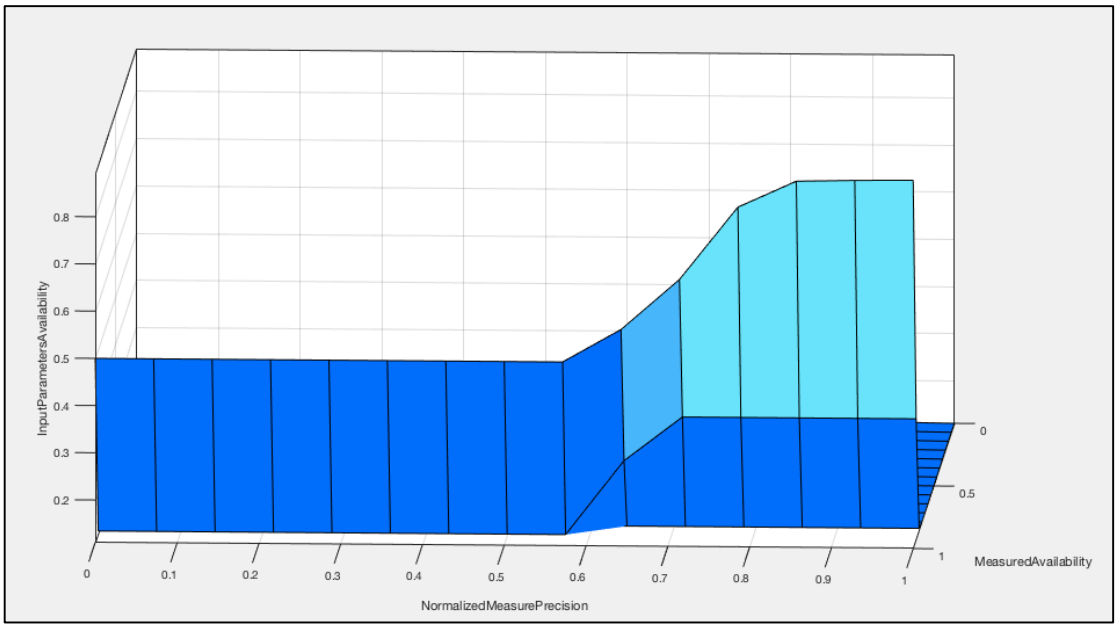


Figure 35. Graphical representation (output surface, with 3D plot) of the *InputParametersAvailability* as a fuzzy function of the two input variables (*MeasuredAvailability* and *NormalizedMeasurePrecision*).

As it is possible to infer from the output surface, we have higher output values when the availability and the normalized measure precision are high. Now let us present **data** and **results** concerning the **Input Parameters Availability** computation. From the analysis of the logs we obtain the availability of each component (**Ai**) with the following formula:

$$\mathbf{Ai} = \frac{\boldsymbol{Time\ interval\ where\ STATUS{=}ON}}{\boldsymbol{Overall\ Observation\ Time}}$$

Clearly, the **Overall Observation Time** includes both the time interval where the *Status = ON* and the time interval when the *Status = OFF*. Note that, coherently with the original paper [*D. Perez-Palacin, R. Mirandola, 2014*], we collected availability from the logs reported in **[Chapter 4.1]**, obtaining:

- *Mobile Device Availability* = **0.9928**
- *Internet Access Point (Antenna) Availability* = **0.988**
- *Internet Access Point (Wi-Fi) Availability* = **0.077**
- *Internet Service Provider (sp1) Availability* = **0.9929**
- *Internet Service Provider (sp2) Availability* = **0.922**
- *Internet Service Provider (sp3) Availability* = **0.988**

Then in order to have a real value (and then fuzzify it) regarding the *NormalizedMeasurePrecision* let us analyse the variance of measured value with the *'Divide and Conquer'* approach, as anticipated in the previous chapters. We will use once again real world data from *web-sites.avt*, for each of the six modules under analysis (one mobile device, two accessing media and three service providers). In this way we can recognize whether the information in the trace represents the steady-state availability behaviour of the considered component or not (influencing or not the registered availability measure). In order to calculate the variance of the measured availability, let us firstly recall the definition of variance for a generic **aleatory variable X**[10]:

$$\sigma^2_X = \mathbf{Var(X) = E[(X - \mu)^2]}$$

Given that we are dealing with real world data, we have not the theoretical values. Therefore, we will **estimate** the expected value of the availability μ_{AV} and the variance σ^2_{AV} (for the service providers, as anticipated). As consolidated routine in statistics, we will estimate these parameters with two famous **estimators**: the **sample mean** for μ_{AV} and the **sample variance** for σ^2_{AV}:

$$\textbf{Sample Mean} = M_{AV} = \frac{1}{Nsamples} \times \sum_{i=1}^{Nsamples} Availability(i)$$

$$\textbf{Sample Variance} = S^2_{AV} = \frac{1}{(Nsamples - 1)} \times \sum_{i=1}^{Nsamples} (Availability(i) - M_{AV})^2$$

In the reference paper the authors considered the service providers' availability log for regarding the 13th line in *web-sites.avt*. We will perform the same type of analysis for each of the six modules. Here with the *'Divide and Conquer'* methodology (introduced in **[Chapter 3.2]**) we will use three different availability measure for each module:

- *Availability regarding the total amount of time (already known);*
- *Availability in the first half of the log;*
- *Availability in the first quarter of the log.*

[10] In the following formula **X** is the generic aleatory variable (in our case the *Availability*). **E[…]** represents the *expected value*, as convention in statistics. Last but not least, μ in the end represents the medium value of the aleatory variable (μ = **E[X]**, in our case the average availability measure).

We will use these **three time intervals** for each sample in order to estimate the statistical parameters, as anticipated before. We want to remark here that if we want more reliable estimations of the statistical parameters, we can proceed performing additional availability samples in different temporal intervals (it is just an extension of what we are going to do here). We will report here the entire workflow it for **sp1**; then the normalized measure precision values for the mobile device, the access points and the other two providers will be computed exactly in the same way. Therefore, let us present the full computation for the first service provider and the final results for the other five modules. Let us start from the **sample mean** computation for **sp1**.

$$\textbf{Sample Mean (sp1)} = \mathbf{M_{AV}} = \frac{\mathbf{1}}{\mathbf{3}} \times \sum_{\boldsymbol{i=1}}^{\mathbf{3}} Availability(i) =$$

$$= \frac{\mathbf{1}}{\mathbf{3}} \times (\mathbf{0.9919 + 0.9927 + 0.9929}) = \mathbf{0.9925}$$

Then we compute the **sample variance**:

$$\textbf{Sample Variance (sp1)} = \mathbf{S^2_{AV}} = \frac{\mathbf{1}}{\mathbf{2}} \times \sum_{\boldsymbol{i=1}}^{\mathbf{3}} (Availability(i) - \mathbf{0.9925})^{\mathbf{2}}$$

$$\frac{1}{2} \times [(0.9919 - 0.9925)^2 + (0.9927 - 0.9925)^2 + (0.9929 - 0.9925)^2] =$$

$$= \mathbf{2.8 \times 10^{-7}}$$

Therefore, we can summarise that for **sp1** $\mathbf{M_{AV} = 0.9925}$ and $\mathbf{S^2_{AV}} = \mathbf{2.8 \times 10^{-7}}$. As we can see, the sample variance has a very low value. We can map this value to the *NormalizedMeasurePrecision* in several ways, in this specific case we can do an **association table** where for each value of the variance is obtained a corresponding value of *NormalizedMeasurePrecision* (**NMP**) in the interval [0;1]. As alternative, it is possible to use a **normalization factor**[11] (η) for the mapping. Here are reported some **reference values** for the association:

$\mathbf{S^2_{AV}}$:	>1	10^{-2}	10^{-4}	10^{-6}	$<10^{-8}$
NMP:	0	0.25	0.5	0.75	1

[11] Note that the *normalization factor* (η) and the *mappings* are **domain specific**; as we can imagine different applications or datasets can have different normalization factors and mappings.

Also in this case, the mapping can be done in several ways and with different associations, if needed (it is strongly application-dependent, similarly to η). Once we have obtained these sample variance values, we have to fuzzify them. Regarding the linguistic labels for **sp1** (with *NMP=0.801*), they will be *Precise = 0.75* and *VeryPrecise = 0.25*. Let us present the other results for the normalized measure precision (approximated to the third decimal cipher):

- *Mobile Device Normalized Measure Precision* = **0.901**
- *Internet Access Point (Antenna) Normalized Measure Precision* = **0.952**
- *Internet Access Point (Wi-Fi) Normalized Measure Precision* = **0.673**
- *Internet Service Provider (sp1) Normalized Measure Precision* = **0.801**
- *Internet Service Provider (sp2) Normalized Measure Precision* = **0.830**
- *Internet Service Provider (sp3) Normalized Measure Precision* = **0.852**

As we have seen, the values for the input parameters availability computation are derived directly from the logs (real world data). Therefore, they will be fixed for the three iterations of the process. Now we can compute the *InputParametersAvailability* (also known as α in this work) directly with the *MATLAB Toolbox*, obtaining both the fuzzy memberships and the defuzzified value. Using the *shift-based interface*[12] (proposed in **[Chapter 4.2]**) with the *MeasuredAvailability* of each component and their *NormalizedMeasurePrecision* computed above it is possible to achieve a fuzzy output result (and a defuzzified one) for each module. In this way, in fact, for each component we obtain its *InputParametersAvailability.* Then the next step of this chapter is to compute the *Overall Input Parameters Availability* of the system (α), considering all the modules together. (once again, this is not the classical availability measure, but an auxiliary value for further analyses). Note that it is also possible to follow another effective street in the *membership design phase,* where for each point of the domain the sum of the fuzzy membership functions is equal to one. Coming back to the *Input Parameters Availability* for each component, using the *MATLAB* tool, we obtain the following *defuzzified values* (with centroid):

- *Mobile Device Input Parameters Availability* = **0.7804**
- *Internet Access Point (Antenna) Input Parameters Availability* = **0.808**
- *Internet Access Point (Wi-Fi) Input Parameters Availability* = **0.067**
- *Internet Service Provider (sp1) Input Parameters Availability* = **0.675**

[12] Note that the *shift-based interface* of *MATLAB* approximates the real value to the **third decimal cipher**.

- *Internet Service Provider (sp2) Input Parameters Availability* = **0.406**
- *Internet Service Provider (sp3) Input Parameters Availability* = **0.606**

In order to obtain the overall *Input Parameters Availability* from the previous crisp values for the modules, let us proceed with the mathematical computation over the block diagram (series):

$$\mathbf{IPA_{OVERALL}} = \prod_i \boldsymbol{IPAi} = \mathbf{IPA_{MD}} \times \mathbf{IPA_{AM}} \times \mathbf{IPA_{SP}} = \mathbf{0.592}$$

Where:

- $\mathbf{IPA_{OVERALL}}$ = *Overall (Whole System) Input Parameters Availability*
- $\mathbf{IPA_{MD}}$ = *Mobile Device Input Parameters Availability*
- $\mathbf{IPA_{AM}}$ = *Accessing Media Input Parameters Availability*
- $\mathbf{IPA_{SP}}$ = *Service Providers Input Parameters Availability*

Regarding the computation of these values we have the crisp value obtained via centroid defuzzification. For the mobile device we have $\mathbf{IPA_{MD} = 0.7804}$, as we have seen before. For the accessing media and for the service providers we have to use the formula for the parallel:

$$\mathbf{IPA_{AM}} = \mathbf{1} - \prod_i^{\boldsymbol{Antenna,Wi-Fi}} (\mathbf{1} - \boldsymbol{IPAAMi}) = \mathbf{0.821}$$

$$\mathbf{IPA_{SP}} = \mathbf{1} - \prod_i^{\mathbf{3}} (\mathbf{1} - \boldsymbol{IPASPi}) = \mathbf{0.9239}$$

With the obtained values, the final value for $\mathbf{IPA_{OVERALL}}$ (or $\boldsymbol{\alpha}$) is equal to **0.592** (with membership *Normal = 0.91*). Clearly this value is quite different if we compare it to the classical availability values (and the one obtained in the reference paper, which was **0.98**). This happens because it is computed considering that the information in the availability model is not trustworthy. Thus, with this new layer of uncertainty, the value we obtain for the system availability is lower. For our purposes we have computed a value which reflects the impact of the uncertainty over the *Input Parameters Availability*. As anticipated, it is possible to calibrate in a different way membership functions and values in order to obtain values with a different scale of comparison or it is possible to map this value on a classical availability scale. In the next chapter we will continue to follow this approach, focusing on the model structure availability and related sources of uncertainty.

4.3 Model Structure Uncertainty and Availability

In this chapter we will explain the components of the Fuzzy analysis related to the model structure. The first input variable regards the **confidence** (between 0 and 1) that a specific service provider is ON, e.g., that it is possible unavailability corresponds to a transient out-of-service period but that it still exists in the world. It is possible to extend this approach to the overall number of servers with a summation of the confidences, in order to estimate the online providers with a certain degree of overall confidence. The second input variable, instead, is used in order to estimate the **dependency** between the *accessing media* (Antenna and Wi-Fi). The more they are dependent, the more the crashes can give reliability limitations. We will adopt this information in a fuzzy variable after a correlation study between the providers. We will adopt here a normalized scale between 0 and 1.

The output style will be the same as in the previous case. Also here all the information will be explained in the specific figures, starting from the variable and concluding with the 3D surface of the Model Structure Availability. As we can imagine, sometimes the figures are more informative and clear with respect to the words: thus let us present everything starting from the block diagram (black-box model).

In **[Figure 36]** is presented the block diagram of the **Model Structure Fuzzy model**. On the left we can notice the input variables (*ProviderIsOnConfidence* and *ProvidersDependencies*), while on the right there is the output variable (*ModelStructureAvailability*). The white box in the middle represents the *Mamdani*-based rule box.

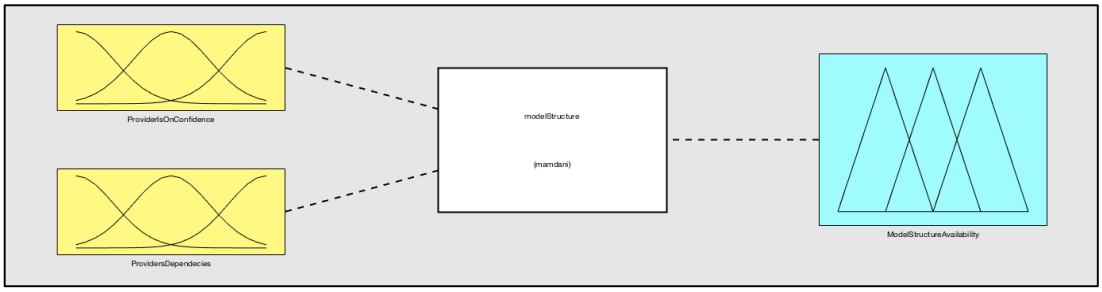


Figure 36. Block diagram of the Model Structure Fuzzy model.

Let us start our discussion in order to model this first model stucture-related fuzzy variable with its membership function. In **[Figure 37]** are presented the shapes for the membership functions of the first input variable (*ProviderIsOnConfidence*). There are 3 membership functions here, the domain is [0,1] as anticipated before (it reflects the confidence from a statistical/ mathematical point of view). The first membership function is called *No-WayIsOn*, it is a triangular shape and rages from the maximum in 0 (when we are sure that the provider is *OFF*) to 0.5. The second membership function is another triangular shape (in this case isosceles and not rectangular). It ranges from 0 to 1, with maximum in 0.5 and it is called *ReluctantlyTrustIsOn*. The third and last membership function represents the higher values of confidence. It ranges from 0.5 to 1 (where 1 means 100% confidence of *Status=ON*) and it is called *SureIsOn* (the more the confidence is high, the more we are sure about the online status of the analysed provider). From a practical point of view, it is possible to investigate on the received last timestamp where *Status = ON*. Clearly, the more it is recent, the higher the confidence of its online status (and *vice versa*).

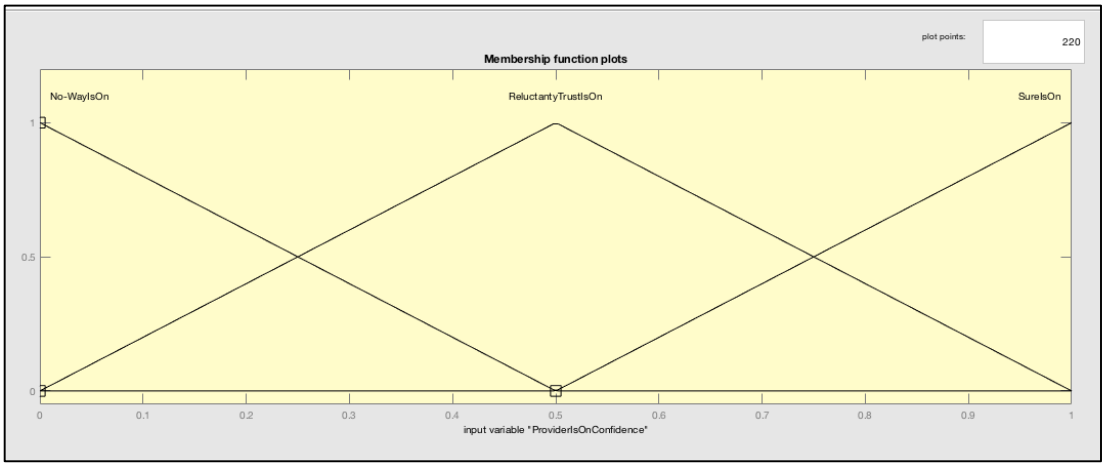


Figure 37. *ProviderIsOnConfidence* (input variable) membership functions.

Now let us start a consideration about the accessibility of the network for our specific case study. As we have seen previously in **[Chapter 3.2]**, regarding the epistemic sources of uncertainty for the Model Context, we can identify several possible problems. We recall here the possibility of DNS errors, broken routing links, non-optimal redirecting routing paths, dependences in providers' crashes and son on. It is evident that all these problematics can have a strong influence on the accessibility of the system. Here we will analyse a particular aspect that afflicts the accessibility of the system: the **dependences in crashes** between the

accessing media. The more the dependency is consistent, the more simultaneous crashes will arise (and the less will be the accessibility of the system).

In **[Figure 38]** are then presented the shapes for the membership functions of the second input variable (*ProvidersDependencies*). There are 4 membership functions here: one singleton and three trapezoids. The initial singleton in zero (*All-Independent*) represent the case where all the providers are completely independent (therefore there is not dependency at all, with value 0 on the abscissa axis). Then as dependence between the providers grows, there is the first trapezoid *Mostly-Independent*. It constantly ranges from 0 to 0.1 modelling indifference, then goes to 0 when the dependency level is 0.3. At this point, there is *Some-Dependencies*, the second trapezoidal shape which ranges from 0.1 to 0.7 with maxima in the interval (0.3 – 0.5). Last but not least there is the trapezoid *Strong-Dependencies* (starts from 0.5 to 0.7 with a linear behaviour, then is steady from 0.7 to 1 in order to model indifference for that interval of dependency values).

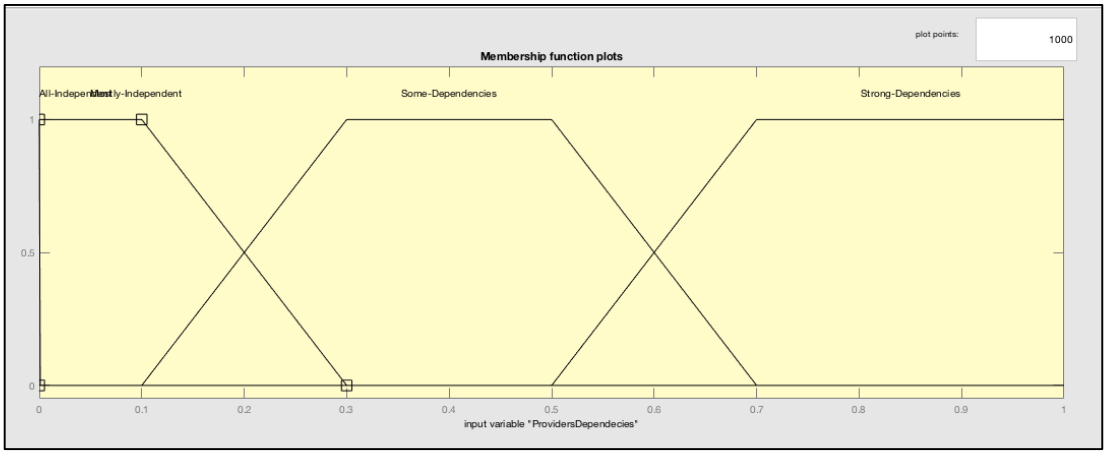


Figure 38. *ProvidersDependencies* (input variable) membership functions.

In **[Figure 39]** we can see the output shapes. There are three membership functions, two triangular shapes for the external membership functions (*Low* and *High ModelStructureAvailability*) ranging respectively from 0 to 0.25 and from 0.75 to 1) and a Gaussian in the middle (*Medium*) with peak in 0.5 for the sake of symmetry. Note that the values on the abscissa do not reflect the proper availability value (eg. 0.5 is not *Availability=0.5*), it is just a normalized measure. Also in this case, if we consider instead the proper availability measure, we need a massive shift of the membership functions on the left.

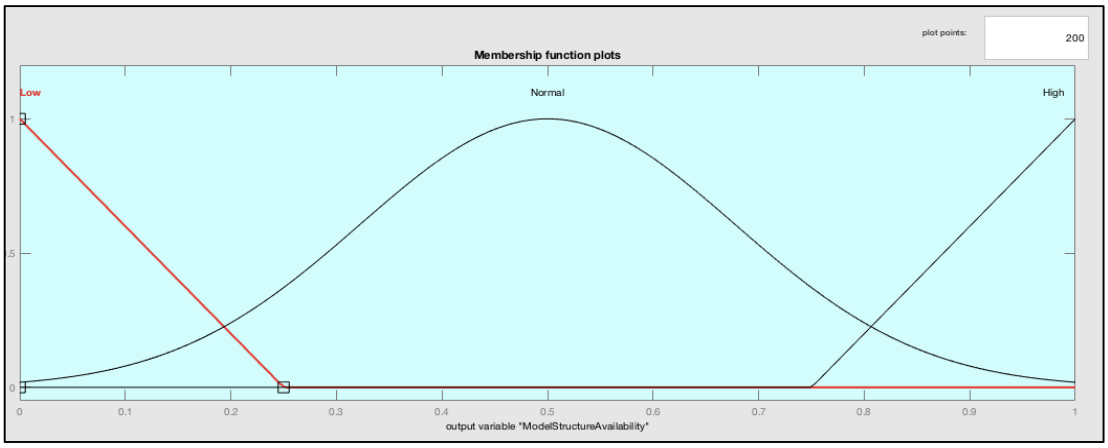


Figure 39. *ModelStructureAvailability* (output variable) membership functions.

Let us open the second **Rule Box**. In **[Figure 40]** are presented the 12 rules that combine the two input variables in order to give the output variable. Here both variables have a strong influence over the final outcome. Intuitively, for an optimal behaviour we desire a high confidence of online status for every provider and a low dependency between them (achievable via server replications in different areas and similar techniques). Note that it is possible to estimate the number of online providers given a specific time instant with the information on their confidence to be online and specific statistical techniques, methodologies and algorithms.

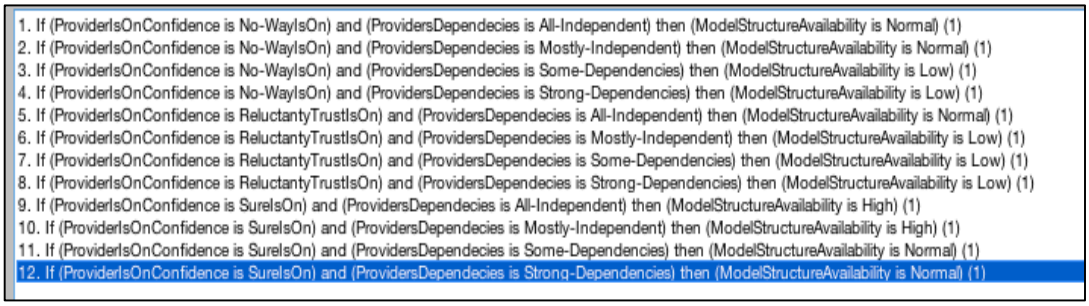


Figure 40. Rule-box concerning *ModelStructureAvailability* Fuzzy outcome, starting from the input variables.

In **[Figure 41]** and **[Figure 42]** are presented the output details for this specific step of Fuzzy Analysis:

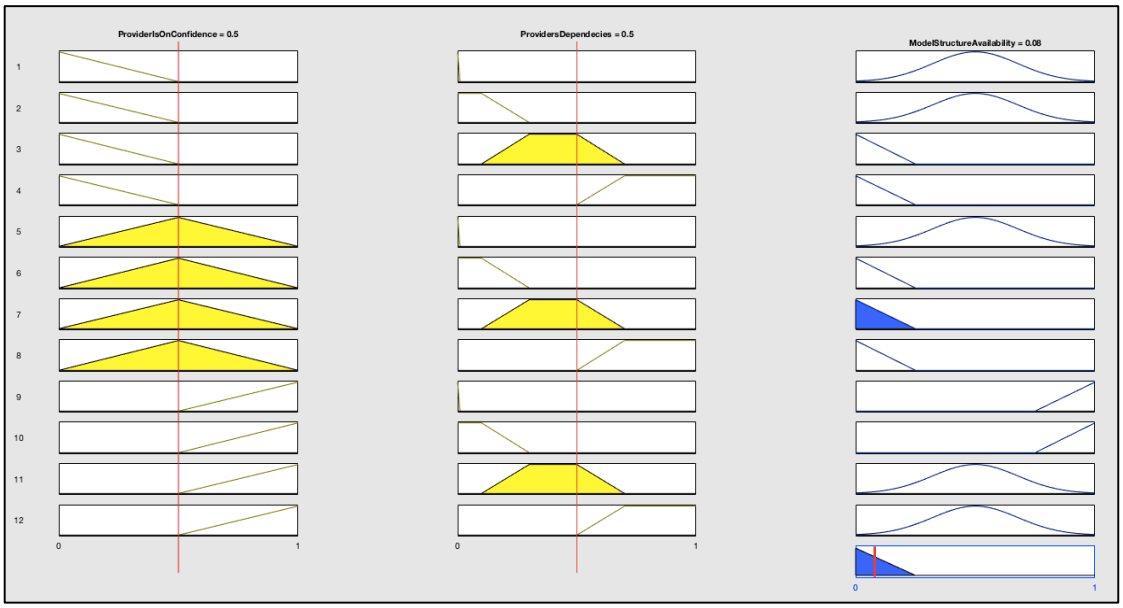


Figure 41. The shift-based interface of *MATLAB* that shows the influence of the input variables over the final output for the considered model (for the Model Structure Uncertainty and the related Availability).

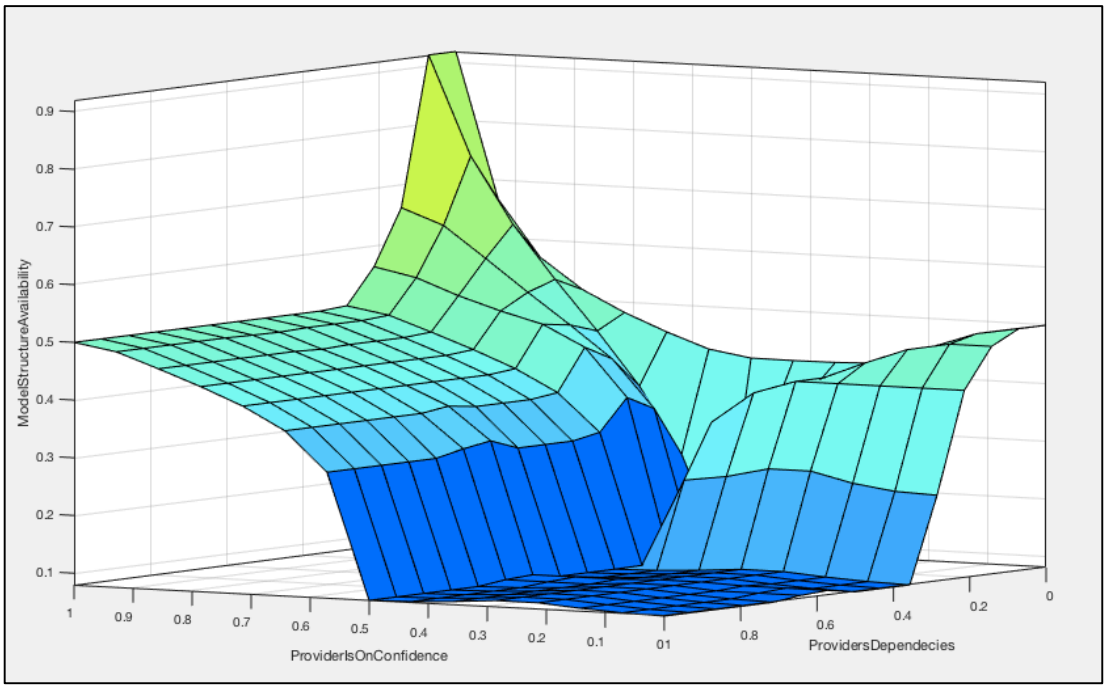


Figure 42. Graphical representation (output surface, with 3D plot) of the *ModelStructureAvailability* as a fuzzy function of the two input variables (*ProviderIsOnConfidence* and *ProvidersDependencies*).

The output surface follows the guidelines of the rules, with higher values when the confidence is high and the dependencies are limited. Let us go now towards the *Model Structure Availability* computation. The ideal estimation for the *ProviderIsOnConfidence* is based on the past behaviour (historical data) of the same provider, or the last information available on its online status (it can send a typical acknowledge message to a managing system when it is ON as happens in telecommunication protocols, for example). Looking at the logs, we can understand that we have different lengths of observation time for the three providers, in fact we have the following observation intervals:

- Service Provider 1: **18,222,329 seconds**;
- Service Provider 2: **18,222,333 seconds**;
- Service Provider 3: **18,221,869 seconds**.

If we select a specific observation timestamp, for example $\mathbf{T_1 = 18{,}000{,}000}$ **seconds** we have the following information about the providers:

- Service Provider 1 *Current Status = ON*, Last Online Appearance Timestamp (after a previous failure): **17,274,901 seconds**;
- Service Provider 2 *Current Status = ON*, Last Online Appearance Timestamp (after a previous failure): **16,986,287 seconds**;
- Service Provider 3 *Current Status = OFF*, Last Online Appearance Timestamp (after a previous failure): **17,954,861 seconds**.

Given these data, if we want to perform an *overall confidence estimation* at the end of the next minute, $\mathbf{T_2 = 18{,}000{,}060}$ **seconds**, we can give the following confidences for the three providers (based on historical data and last ON timestamp):

- Service Provider 1 *Confidence* = **0.99**;
- Service Provider 2 *Confidence* = **0.95**;
- Service Provider 3 *Confidence* = **0.76**.

In our experiments for the estimation of the fuzzy variable we will use an integrated value of **0.9** (the *arithmetic mean* of the confidences reported above), meaning that on average we have **90% of confidence** that a provider is **ON** (in practice different providers have different levels of confidence as we have seen, but we perform a simplification here). Of course, it is possible to select different values and see how changes the output. If we fuzzify this value, we have the

following memberships: *ProviderIsOnConfidence* is *SureIsOn = 0.8* and *ReluctantlyTrustIsOn = 0.2*. If we consider the *ProvidersDependencies* regarding the *access media providers*, the ideal would be to perform a preliminary study similar to the one performed in **[Chapter 5]** with the support of Neural Networks (or we can use other instruments and methodologies). Nowadays in practice, due to several causes (same physical location of servers, dependence in their modules and so on), it is almost impossible to have a full providers' independence. If we consider our data concerning the *access media providers*, we have the following observation intervals:

- Antenna: **1,017,518 seconds**;
- Wi-Fi: **18,221,885 seconds**.

Considering only the time interval in common (*1,017,518 seconds*, because after this timestamp we do not have additional data for the Antenna), we can perform an *overlapping study* between the intervals of concurrent offline state. We have that on the *12,308 seconds* where the Antenna has *Status=OFF,* there are *2,401 seconds* where the Wi-Fi is concurrently offline. This means that on average, the **19,507%** of crashes are simultaneous, therefore the crisp value to be fuzzyfied is *ProvidersDependencies* = **0.19507** on the abscissa axis. We therefore register the following fuzzy memberships: *Mostly-Independent = 0.52* and *Some-Dependencies = 0.48*. Once again, we have all the instruments to compute the membership also with slightly different percentages of dependencies (directly with the reference *MATLAB* interface).

Once we have obtained these values, we can use the *MATLAB* interface in order to estimate the **crisp value** for the Model Structure Availability (given the rule-based system). Given these two fuzzy variables and their memberships, we obtain (through *centroid defuzzification*) *Model Structure Availability* equal to **0.712** (with membership *Normal = 0.49*). Note that this value does not reflect the impact of the input parameters sources of uncertainty (it is an orthogonal approach).

4.4 Model Context Uncertainty and Availability

Let us present the components of the model context uncertainty and the related availability from a fuzzy point of view. The first input variable regards the battery life (if we want to be precise, the predicted battery life) of the smartphone/tablet at a given point of the time. This information is crucial for the considered system; for example, some usage profiles are not possible with a very low level of battery. Here we perform uncertainty handling through *model refinement*, in order to obtain the battery value. Note that formally the battery level is an input parameter, but the external (exogenous) variables influencing the battery belong to the model boundaries (and so they are related to the context).

The second model input variable references to the robustness of the computing infrastructure and its computation is a little bit trickier. How we can calculate (or at least estimate) the robustness of a computing infrastructure (often very wide and complex)? There are several methods in literature, one very interesting (and actual) proposal may be the one of [*Nageswara S. V. Rao, Fei He, Jun Zhuang, Chris Y. T. Ma, David K. Y. Yau, 2012*]. The authors apply the concept of *Nash Equilibrium*, given a Cloud Infrastructure, in order to achieve a measure for the robustness of the system. For our purposes, we will use a measure of robustness as a function of dependences in crashes. We will adopt a percentage ranging from 0% to 100% (note that if we want this value can be once again normalized in the range 0 to 1).

Note that this study is very similar to the **dependency** studied in the previous subsection. However, our model allowed to represent providers and access media, in the previous subsection it was classified a model *Structural Uncertainty*. However, the model used to represent the system does not allow to model the computing infrastructure of providers, and then such **dependency on shared computing infrastructures** belongs to the category of uncertainty in the Model *Context*. Hence, we will explicitly deal with the service providers (*sp1, sp2* and *sp3*) and their dependency for the fuzzification and the analysis performed.

In **[Figure 43]** we can observe the block diagram of the **Model Context Fuzzy model**. On the left we can see the input variables (*PredictedBatteryLife* and *ComputingInfrastructureRobustness*), on the right instead there is the output variable (*ModelContextAvailability*). The white box in the middle represents as usual the *Mamdani*-based rule box.

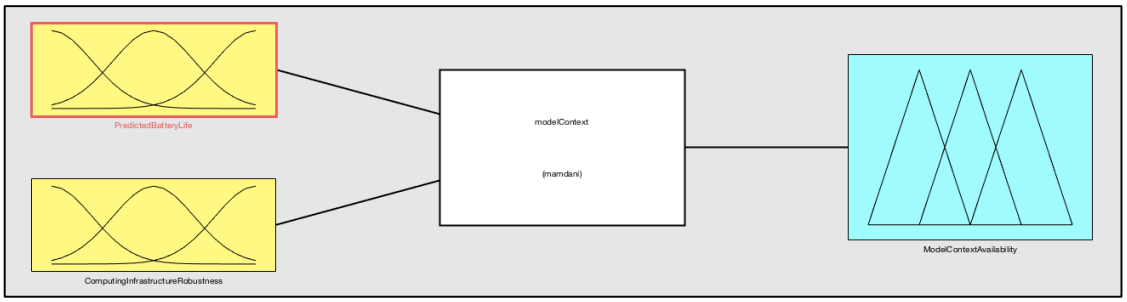


Figure 43. Block diagram of the Model Context Fuzzy model.

Regarding the original problem statement and the considered case study, let us focus now on a stringent and actual problem of the mobile devices: the battery and its lifetime. We want to watch a video streaming on our user side device, but are we going to have enough battery to accomplish the mission? If not, it is possible for the self-adaptive device management to select a different profile in order to solve the problem (for example reducing the display brightness, closing some background applications, disabling the Bluetooth or similar actions).

The answer is not considerable straightforward here. It really depends from the specific device under analysis (tablet/smartphone and its manufacturer), from the current (predicted) battery life and also from other external (exogenous) factors, as our real life experience and commons sense suggests. In order to handle this problem, we will start from a simple idea: encoding the predicted battery life into a Fuzzy variable.

In **[Figure 44]**, in fact, are presented the shapes for the membership functions of the first input variable (*PredictedBatteryLife*). This variable is related to the **mobile device module** of our case study. There are 5 membership functions here with a clear symmetry in their design; they reflect 5 possible situations of the battery: *Critical* (a triangular shape from 0 to 0.2, meaning the 20% of the battery), *Low* (a triangular shape ranging from 0 to 0.4, with maximum in 0.2), *Medium* (a trapezoid ranging from 0.2 to 0.8, with indifference modelled with battery in the range 40%-60%), *High* (a triangular shape from 0.6 to 1, with maximum in 0.8) and *Awesome* (the last triangular shape, ranging from 0.8 to 1, where the battery has the full charge available).

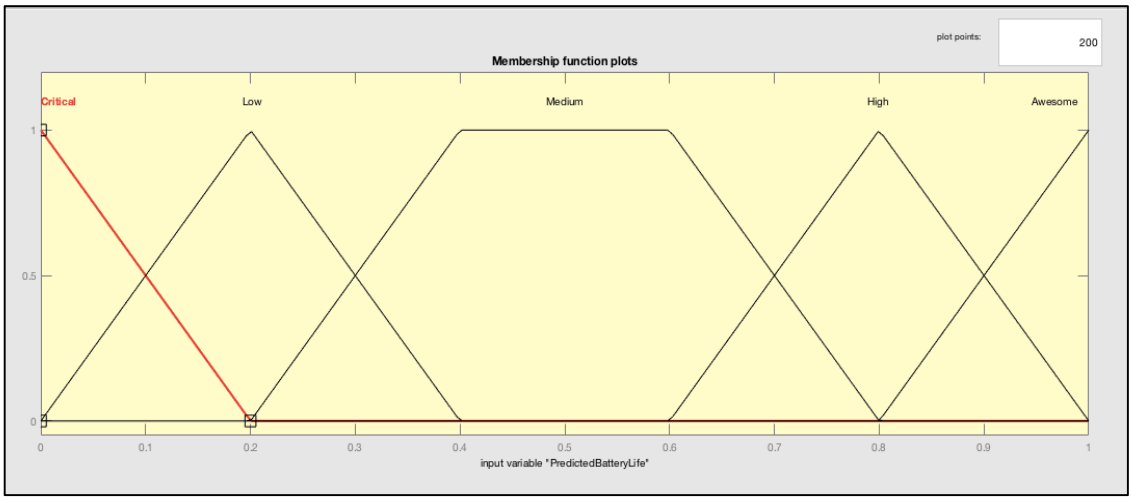


Figure 44. *PredictedBatteryLife* (input variable) membership functions.

In **[Figure 45]** there are possible graphical representation (as already done in a considerable number of smartphones, tablets and other electronical instruments) of the mentioned predicted battery levels and related membership functions (*Critical, Low, Medium, High* and finally *Awesome*). We will consider these levels as reliable, therefore we will not consider 'pathological' cases where this information is untrustworthy (for example with old devices where the sensors are not calibrated/outdated or the batteries are defective).

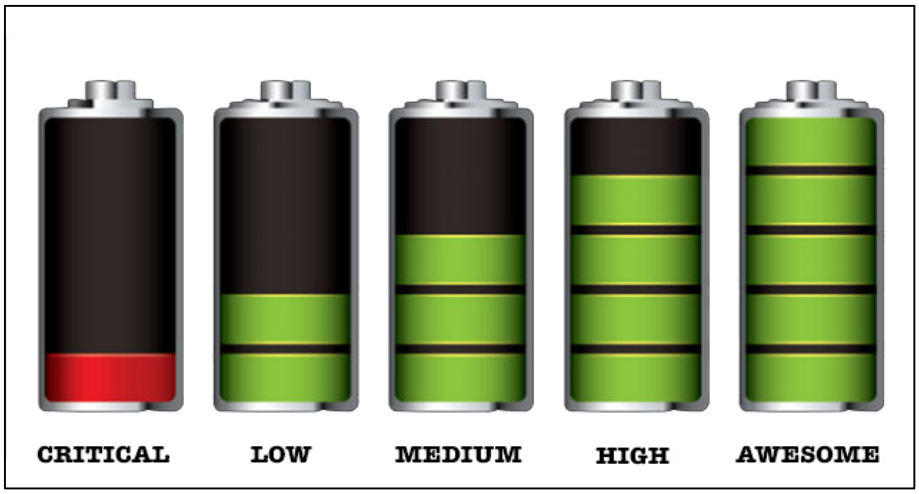


Figure 45. Possible graphical representation of predicted battery levels.

Now let us consider the computing infrastructure of the system under analysis, trying to map its robustness into a fuzzy variable. Also here the (normalized) value of robustness can be computed as a function of different parameters, such

as the presence of backups, replications, availability zones and security policies. We can imagine here a very general formula with possibly *context-aware* specializations with respect to different applications domains and available datasets. We will adopt for simplicity a *frequency-based* approach on simultaneous variations in the states of the providers. The more they are dependent, the more the **percentage of simultaneous failures** will be relevant.

In **[Figure 46]** are presented the shapes for the membership functions of the second input variable (*ComputingInfrastrcutureRobustness*). There are 3 trapezoidal membership functions here. On the abscissa axis is reported the percentage of simultaneous crashes between the third party providers. It is easy to derive that higher the percentage, lower the robustness of the infrastructure. The first trapezoid *GreatRobustness* ranges from 0% to 20% (it is constant from 0% to 10%, then the membership function lineraly descreases to 0). *NormalRobustness* ranges from 10% to 30% growing, then from 30% to 50% stays to one and linearly descreases from 50% to 70%. The last trapezoid *NotRobust* grows linearly from 50% to 70% and then reaches the *plateau,* remaining stable. As usual, the values for the memerbership functions can be adapted to specific domains in different ways (here we present coherent values with respect to the analysed study case and application domain).

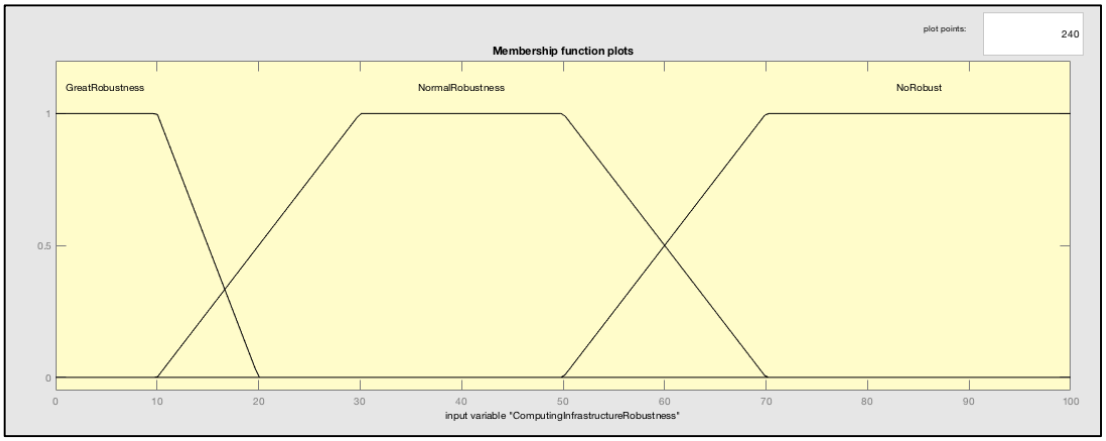


Figure 46. *ComputingInfrastrcutureRobustness* (second input variable) membership functions.

In **[Figure 47]**, as we have seen before for the previous outputs, there are three membership functions, two triangular shapes for the external membership functions (*Low* and *High ModelContextAvailability*) ranging respectively from 0 to 0.25 and from 0.75 to 1) and a Gaussian in the middle (*Medium*) with peak in 0.5

for the sake of symmetry. Note that the values on the abscissa don't reflect the proper availability value (eg. 0.5 is not *Availability=0.5*), it is just a normalized measure.

Note that also in this case, if we consider instead the proper availability measure, we need a massive shift of the membership functions on the left.

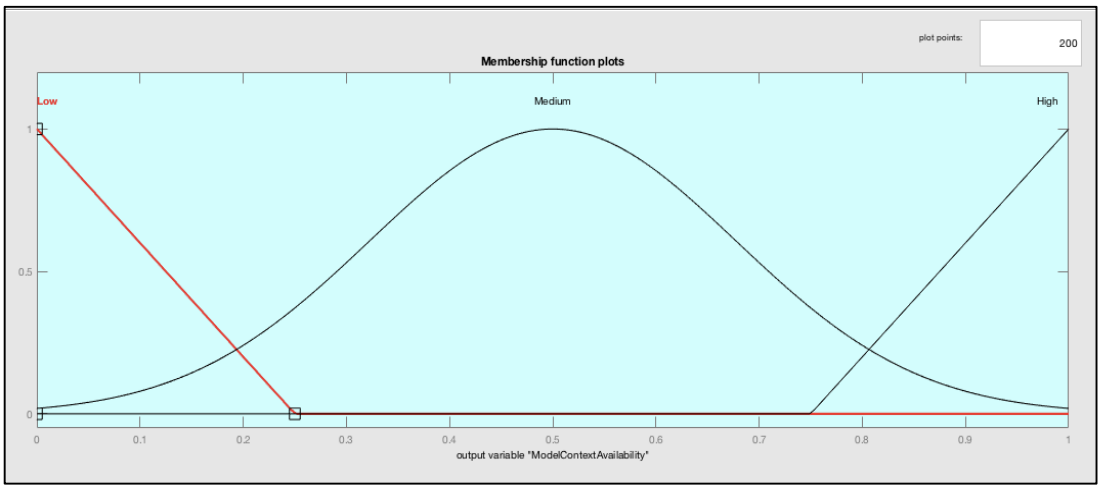


Figure 47. *ModelContextAvailability* (output variable) membership functions.

Let us open the third **Rule Box**. In **[Figure 48]** are presented the 15 rules that combine the two input variables in order to give the output variable. Here both variables have a strong influence over the final outcome. In fact, it is clear that in order to achieve a good Model Context-related Availability we need both a robust computing infrastructure and a sufficient level of predicted battery life (it is almost trivial that without battery, it is not possible to accomplish the original mission).

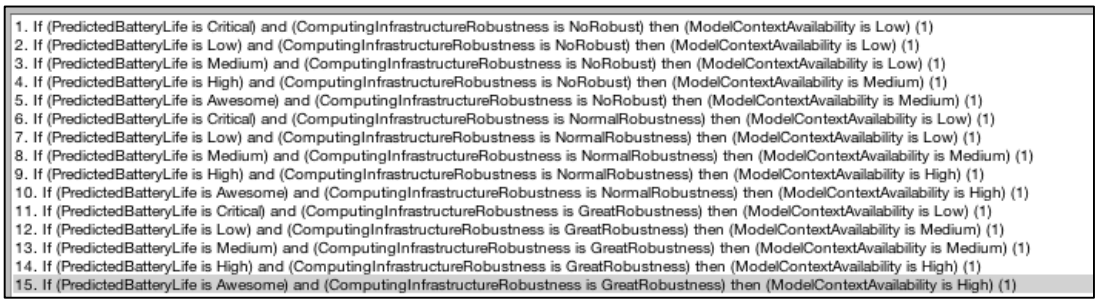


Figure 48. Rule-box concerning *ModelContextAvailability* Fuzzy outcome, starting from the input variables

In **[Figure 49]** and **[Figure 50]**, as consolidated routine, are presented the output details.

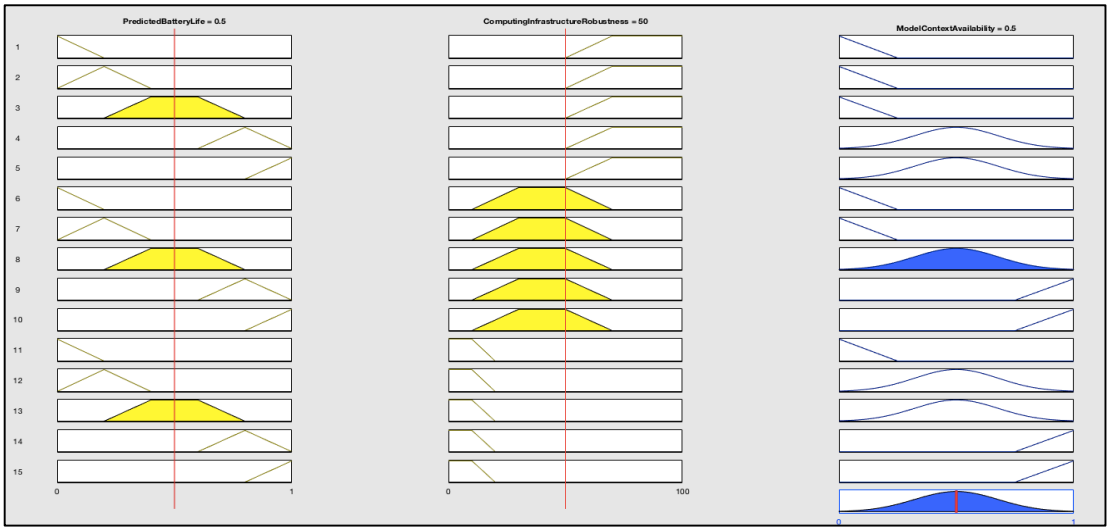


Figure 49. The shift-based interface of *MATLAB* that shows the influence of the input variables over the final output for the considered model (for the Model Context Uncertainty and the related Availability).

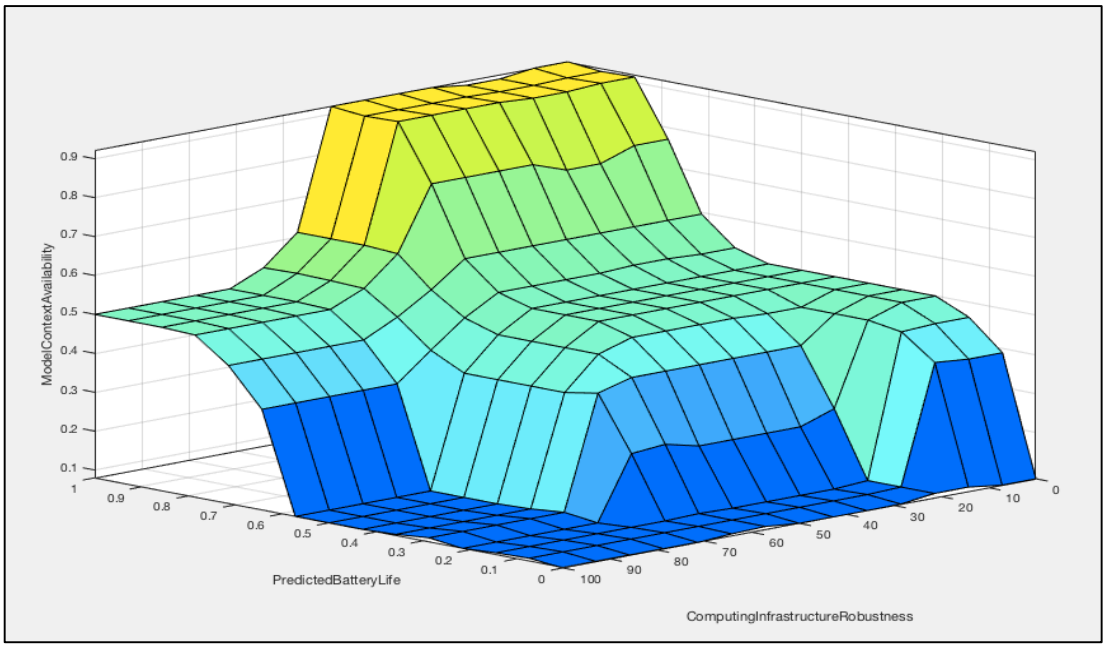


Figure 50. Graphical representation (output surface, with 3D plot) of the *ModelContextAvailability* as a fuzzy function of the two input variables (*PredictedBatteryLife* and *ComputingInfrastructureRobustness*).

After this theoretical introduction, the next step is to compute the *Model Context Availability*, starting from the sources of uncertainty. Regarding the predicted battery life (*PredictedBatteryLife*), we have not real world data from the mobile devices under analysis (we have just its availability, as already seen). We will solve this problem with **three different values** for the three iterations of the analysis. In particular we will adopt the following values (and fuzzy memberships):

- **Predicted Battery Life** (1st Iteration) = **0.1** (or 10%)[13] with membership functions *Critical = 0.5* and *Low = 0.5*;
- **Predicted Battery Life** (2nd Iteration) = **0.5** (or 50%) with membership function *Medium = 1*;
- **Predicted Battery Life** (3rd Iteration) = **0.9** (or 90%) with membership functions *High = 0.5* and *Awesome = 0.5*.

Let us perform another step ahead. From a conceptual point of view we understood that the considered **computing infrastructure robustness** (*ComputingInfrastructureRobustness*) of the system is calculated as function of the **concurrent failures** in the network. This variable is related to the causes already presented and explained before. In our case, considering also the high availability value presented in the reference paper [*D. Perez-Palacin, R. Mirandola, 2014*], the robustness of the entire network is strong. In other words, from a *frequency based study* on the servers (and their correlation), we can estimate the level of concurrent failures between the providers similarly. The proposed approach is equal to what we have already seen in **[Chapter 4.3]** with the access media. For the three providers (*sp1, sp2* and *sp3*), we have the following observation intervals:

- Service Provider 1 (sp1): **18,222,329 seconds**;
- Service Provider 1 (sp2): **18,222,333 seconds**;
- Service Provider 3 (sp3): **18,221,869 seconds**.

The minimum is clearly the third interval (*18,221,869 seconds*). We will use this interval in order to perform the correlation study on the simultaneous crashes. In this case with three providers we can have only one failure (and two providers

[13] With **lower battery levels** (0% - 9%), the *Model Context Availability* rapidly (and asymptotically) goes to **0**, as expected from the real world experience. The battery life, in fact, is essential in order to accomplish the case study mission introduced in **[Chapter 3.1]**.

online), two of the three providers offline (and only one online) or the three providers concurrently offline (worst case for reliability purposes). We have that on the **1,639,968 seconds** where at least on provider is offline, **147,597 seconds** (about the 8,9% of the considered time) there are two providers concurrently offline and **15,579 seconds** (about the 0,94% of the considered time) there are three concurrent failures. Putting together these results, we can estimate the service providers' dependence with the algebraic sum of the previous percentages (there are alternatives, here we perform a simplification). We therefore have a *ComputingInfrastructureRobustness* of **9,84** (or 9,84%,) to be fuzzified. Given the structure of the fuzzy membership functions, with the previous value we obtain a membership *GreatRobustness = 1*. Note that different methodologies can give slightly different results for the robustness estimation, as usual. In addition, also for these two last variables (*PredictedBatteryLife* and *ComputingInfrastructureRobustness*) it is possible to do additional analyses and simulations with different parameters directly with the shift-based interface, if needed.

Using the *MATLAB* interface, considering these variables, their values and membership functions, we obtain (through *centroid defuzzification*) *Model Context Availability* equal respectively to **0.421** (with membership *Normal = 0.9*), **0.461** (with membership *Normal = 0.95*), and **0.5** (with membership *Normal =1*), for the three iterations.

4.5 Overall Availability, final results and considerations

In this section we are going to aggregate the availability results with a *waterfall (iterative) approach* in order to compute the overall availability of the considered system. Therefore, the outputs of the aforementioned availability measures will be the input of the *Fuzzy System* which will compute the **Overall Availability measure**. Knowing the analysis performed so far, it is the time to put the intermediate results together and see what happens from a definitive (and integrated) point of view. Let us do a **brief recap**: we have our case study with the user devices, the accessing media, the service providers and the whole computing infrastructure: can we estimate the overall availability of the considered system with this fuzzy logic-based approach? Yes, in fact we will see how it is possible to perform this estimation with the same instruments already presented and used in the previous sub-chapters. We will adopt here the approach summarized in **[Figure 51]**.

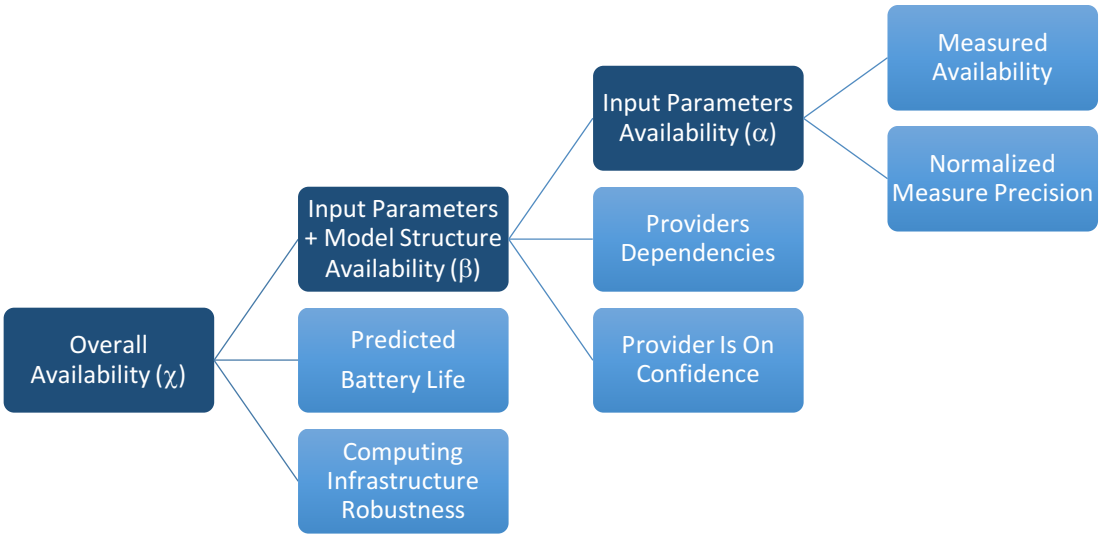


Figure 51. Block diagram of the iterative *Overall Availability* (χ) computation, read the process from right to left (similarly to the writing style of Leonardo da Vinci[14]).

First of all, we will use the *Input Parameters Availability* (α) already computed in **[Chapter 4.2]** with its two fuzzy variables (*Measured Availability* and

[14] **Leonardo da Vinci** wrote most of his personal notes in *mirror writing* (from right to left), only using standard writing if he intended his texts to be read by others. Here we are doing a similarity with the reading direction of our process.

NormalizedMeasurePrecision). Then the *Input Parameters Availability* (α) output becomes one of the input for the *Input Parameters + Model Structure Availability* (β) computation. This fuzzy variable takes into account both the contribution of the *Input Parameters Availability* and the *Model Structure Availability* (with the fuzzy variables *ProvidersDependencies* and *ProviderIsOnConfidence*). After this action, remains the third and final step of the process. The previous output variable becomes one of three fuzzy inputs of this step (together with *PredictedBatteryLife* and *ComputingInfrastructureRobustness*) in order to estimate the value for the *Overall Availability* (χ) of the considered system (in practice it considers the influence of *Input Parameters Availability + Model Structure Availability + Model Context Availability*).

Therefore, starting from the variables related to the *Input Parameters* and the related availability (α), we will converge at a second variable containing the availability intermediate result of *Input Parameters + Model Structure* (β) and then to the *Overall Availability* (χ), who considers all the sources of uncertainty analyzed in the previous sections. The final results will be *de-fuzzyfied* with the canonical **centroid** operator, as we will explain later on. Note that the output surfaces for (β) will be slightly different from the one presented in **[Chapter 4.3]**, because this version takes into account also the contribution of the input parameters (with their fuzzyfied sources of uncertainty). Obviously, we will present all the details with graphs, schemata and descriptions. The same observation holds also for the *Overall Availability* (χ), because it takes into account the two fuzzy variables of *Model Context* (as seen in **[Chapter 4.4]**) plus the aforementioned β. But we will see these details in the remaining part of this sub-chapter.

In figures **[Figure 52]** and **[Figure 53]** we can see the block diagrams for the computation of *Input Parameters + Model Structure Availability* (β) and the *Overall Availability* (χ), with the input variables and the output variables involved in the computation. Regarding the first of the two figures, we have on the left the input variables (*Input Parameters Availability* (α), *ProvidersDependencies* and *ProviderIsOnConfidence*). In the middle there is the *Mamdani*-based rule box as usual and on the right there is the output variable (*Input Parameters + Model Structure* (β)). Note that we can compute (α) as already seen in **[Chapter 4.2]**.

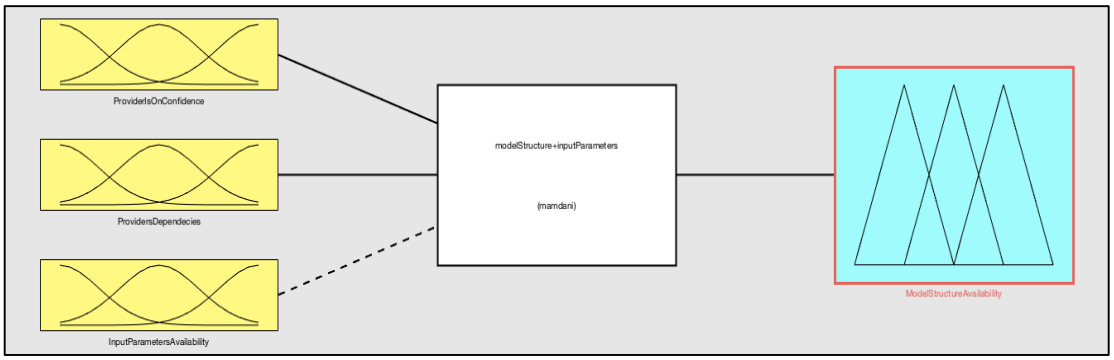


Figure 52. Block diagram of the *Input Parameters + Model Structure Availability* (**β**) Fuzzy model.

Here there is a classical *representation problem*: the output surface is a function of three fuzzy variables, in fact:

$$\beta = f(ProviderIsOnConfidence, ProvidersDependencies, \alpha)$$

We know that is not possible to graphically visualize a function of three (or more) variables. Therefore, there is no chance to report only one graphical output surface here. Nevertheless, it is possible to see the three *partial output surfaces*[15] directly on the *MATLAB* visualization tool. We will not report additional graphs and details for **β** here (also regarding the *Rule Box*, which has $3\times4\times3$[16] = **36 rules** in this case and the shift-based interface). All the material is available directly on the *GitHub repository* (further details directly in **[Chapter 7]**).

---

[15] The *partial output surfaces* show the output plot as function of only two input variables at a time. Therefore, we will have in detail:
$\beta_1 = f(ProviderIsOnConfidence, \alpha)$;
$\beta_2 = f(ProviderIsOnConfidence, ProvidersDependencies)$;
$\beta_3 = f(ProvidersDependencies, \alpha)$.

[16] In order to compute all the possible **rules**, we have to do the *product of sequences* between the input variables' membership function cardinalities (N_i).
In formula we have: **Number of Rules** = $N_{Rules} = \prod_{i=1}^{Number\ of\ Fuzzy\ Variables} Ni$.
In this specific case *ProviderIsOnConfidence and* α have both 3 membership functions, while *ProvidersDependencies* has 4 membership functions. The product is 36, as expected and previously computed.

Let now perform an additional step towards the conclusion. Here we have to evaluate the contribution of the two *Model Context* sources of uncertainty plus the β computed in the previous iteration. In **[Figure 53]** we can see the block diagram of the **Overall Availability Fuzzy model**. On the left the three input variables (*PredictedBatteryLife, ComputinginfrastructureRobusteness* and β). On the right instead we can see the output variable (*OverallAvailability*). The white box in the middle represents the *Mamdani*-based rule box. In this case we have the maximum number of rules seen so far (in fact we have a total of 5×3×3 = **45 rules**). Also in this case arises the representation problem, given that once again we have an output variable as function of three input variables. Here in particular we have:

$$\chi = f(PredictedBatteryLife, ComputinginfrastructureRobusteness, \beta)$$

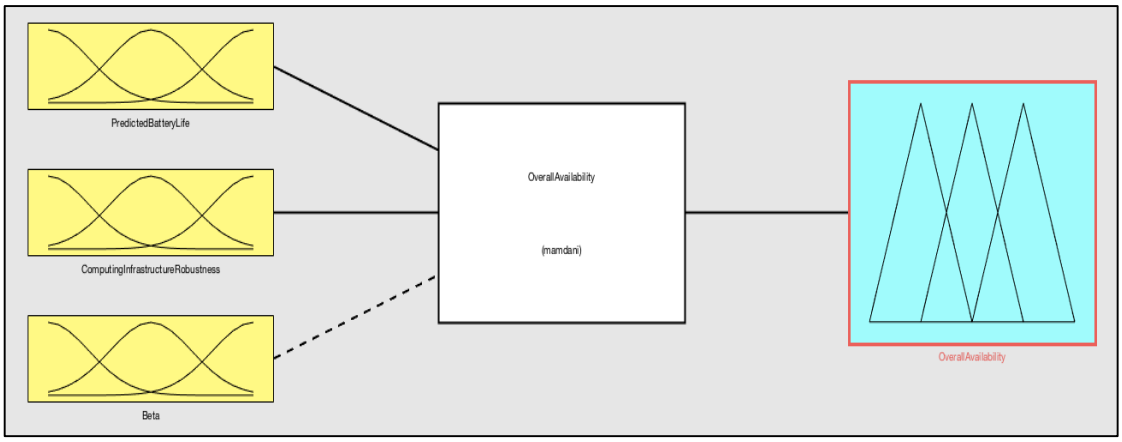


Figure 53. Block diagram of the *Overall Availability* (χ) Fuzzy model.

In **[Figure 54]** in a similar way of what we have seen before for the previous outputs, there are three membership functions, two triangular shapes for the external membership functions (*Low* and *High OverallAvailability*) ranging respectively from 0 to 0.25 and from 0.75 to 1) and a Gaussian in the middle (*Medium*) with peak in 0.5 for the sake of symmetry. Note that the values on the abscissa don't reflect the proper availability value (eg. 0.5 is not *Availability=0.5*), it is just a normalized measure.

Later on in this section, we will perform a reasoning about normalizations and mappings for the obtained values. The main purpose remains to estimate a concrete overall availability value for the real world system under analysis, considering the sources of uncertainty detected and already explained.

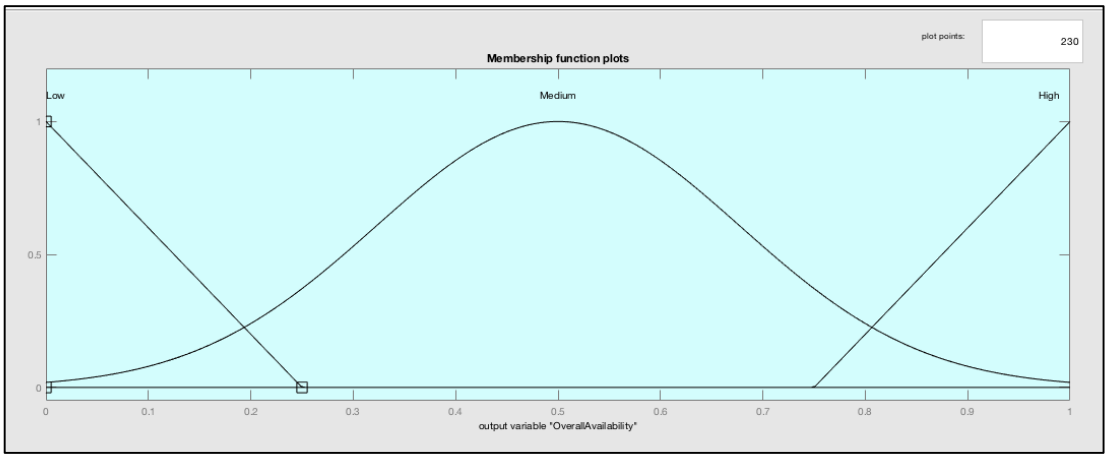


Figure 54. *OverallAvailability* (final output variable) membership functions.

We want to highlight here that the fuzzy approach is always the same, we derived all the **45 rules** that allow us to present the fuzzy output and the partial output surfaces. Due to the size of the *Rule-Box*, we will not enumerate all the rules here (but they are available in the specific *Fuzzy folder*, see the *GitHub Repository*). It is also possible to solicit and experiment the final fuzzy model with different parameters and uncertainty weights, if needed or desired.

Let us come back to the formalisms of our case study, presenting now the three graphical representation concerning the *partial output surfaces* for *the Overall Availability.* In a similar way of what we have previously seen with the *Input Parameters + Model Structure Availability* (β), we will have:

- $\chi_1 = f($*PredictedBatteryLife, ComputingInfrastructureRobustness*$)$, see **[Figure 55]**.
- $\chi_2 = f($*β, PredictedBatteryLife*$)$, see **[Figure 56]**.
- $\chi_3 = f($*β, ComputingInfrastructureRobustness*$)$, see **[Figure 57]**.

Note that they express only partially the meaning of the *Overall Availability* (χ). We will compute χ merging together these contents (as it is possible in practice with the shift-based interface) and real world data in the next chapter. We want to highlight here that χ_1 indicates the output surface as function of *PredictedBatteryLife* and *ComputingInfrastructureRobustness,* χ_2 indicates the output surface as function of *β* and *PredictedBatteryLife* and χ_3 indicates the output surface as function of *β* and *ComputingInfrastructureRobustness.*

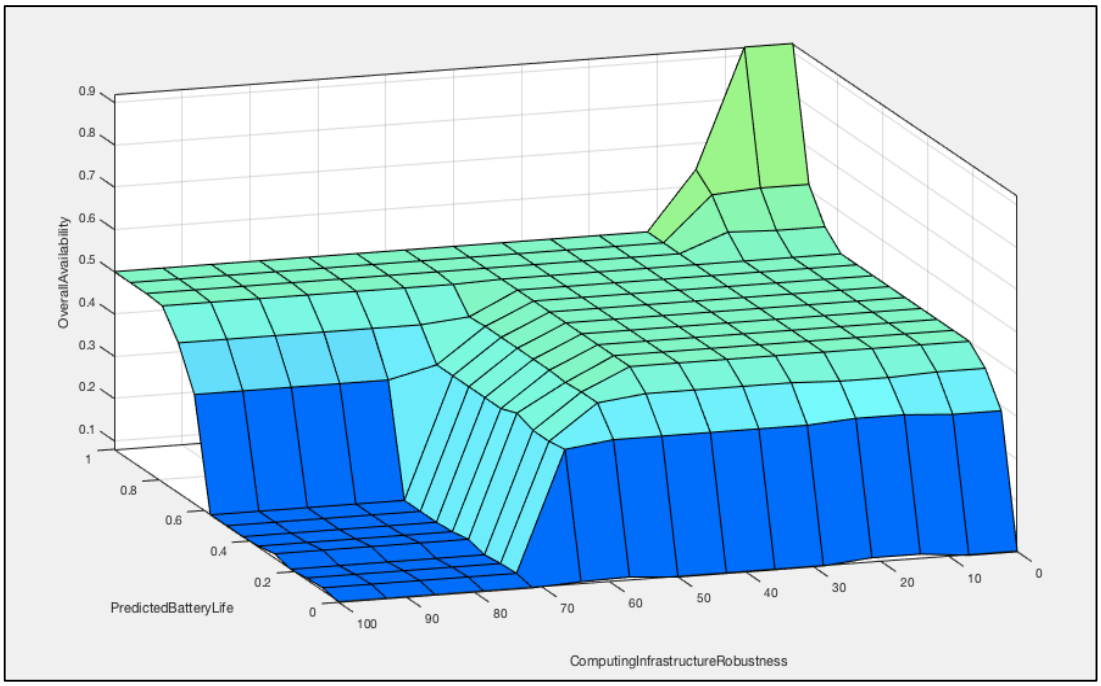


Figure 55. Fist *partial output surface* (χ_1) for the *Overall Availability* (as function of *PredictedBatteryLife* and *ComputingInfrastructureRobustness*).

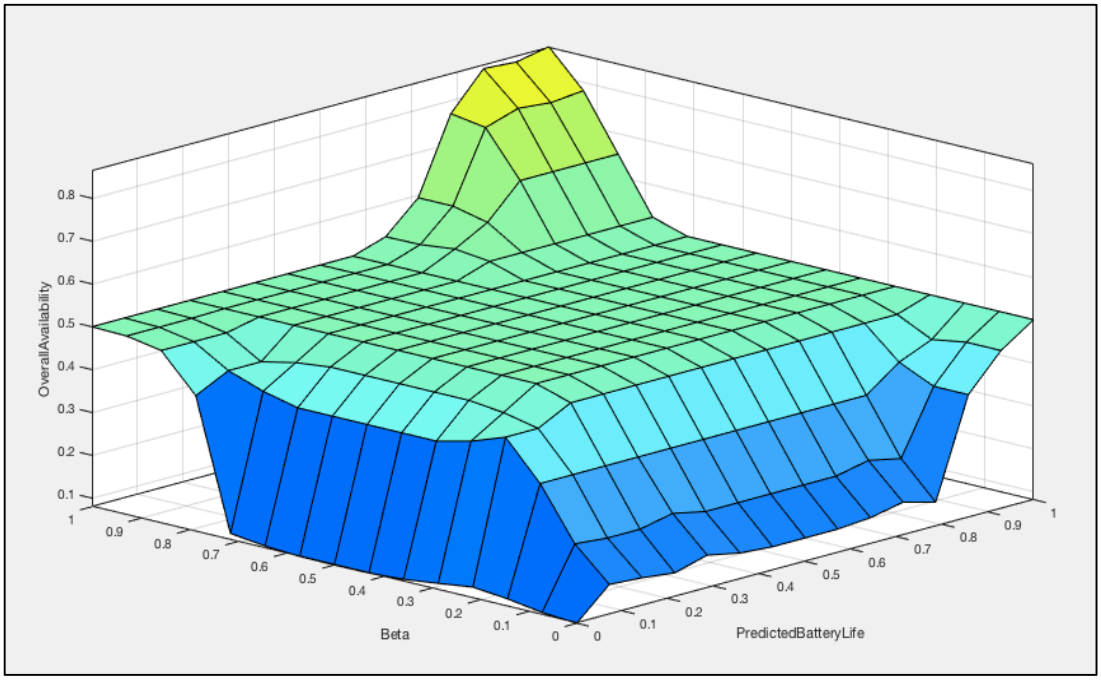


Figure 56. Second *partial output surface* (χ_2) for the *Overall Availability* (as function of β and *PredictedBatteryLife*).

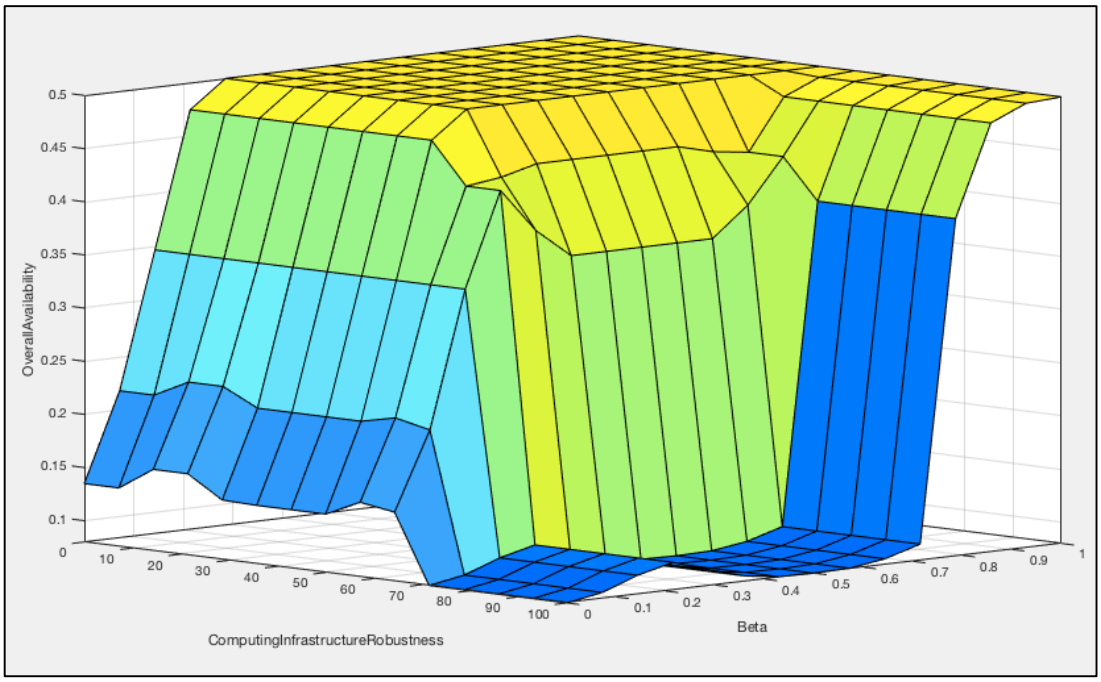


Figure 57. Third *partial output surface* (χ_3) for the *Overall Availability* (as function of β and *ComputingInfrastructureRobustness*).

After this theoretical introduction on the integration that we are going to perform, let us introduce the values (and the results) for the specific variables. The acting strategy is to collect the results from the previous sub-chapters and apply the *waterfall integration*, as anticipated before (see **[Figure 51]**). Regarding the *Input Parameters Availability* (α), we have seen in **[Chapter 4.2]** that the obtained value is **0. 592**. We will use this value for our process iterations, because it is independent (as also β) from the battery level. Now we can compute β with the information collected in **[Chapter 4.3]** about *Model Structure Availability* and the previous value of α (**0.592**). Once again, we initialize the values with the specific *MATLAB* interface, obtaining a value of β = **0.5** (for the integrated *Input Parameters* + *Model Structure* availability). The fuzzy membership for β is only one: *Normal = 1*.

With the available fuzzy variables and their values (for the *Model Context* see **[Chapter 4.4]**), we obtained the following values for the **Overall Availability** (χ) performing the three iterations of the process:

- χ_{1st} (*1st Iteration*) = **0.421** (with membership *Normal = 0.87*);
- χ_{2nd} (*2nd Iteration*) = **0.5** with membership *Normal = 1*);
- χ_{3rd} (*3rd Iteration*) = **0.579** (with membership *Normal = 0.87*).

An important remark about **defuzzification**. In our specific application we have used the **Centroid** as *defuzzification operator* (suggested and supported by *MATLAB*, but also reasonable in practice). Technically speaking, the *Centroid defuzzification* returns the **centre of area under the curve**. If we think of the area as a plate of equal density, the centroid is the point along the x axis about which this shape would balance (as the *centre of mass*, in physics). It returns a defuzzified value from a fuzzy variable, such as α, β an χ in our specific application. All the results (final [χ] and intermediate [α, β], with real world and simulated data) are summarized in the next table, **[Figure 58]**.

	Fuzzy Variable	**Value**	**Fuzzy Membership 1**	**Fuzzy Membership 2**
1st Iteration	α	**0.592**	**Normal = 0.91**	**-**
	β	**0.5**	**Normal = 1**	**-**
	Predicted Battery Life	10	Critical = 0.5	Low = 0.5
	Computing Infrastructure Robustness	9.84	Great Robustness = 1	-
	χ_{1st} ***(Overall Availability)***	**0.421**	**Normal = 0.87**	**-**
	Fuzzy Variable	**Value**	**Fuzzy Membership 1**	**Fuzzy Membership 2**
2nd Iteration	α	**0.592**	**Normal = 0.91**	**-**
	β	**0.5**	**Normal = 1**	**-**

	Predicted Battery Life	0.5	Medium = 1	-
	Computing Infrastructure Robustness	9.84	Great Robustness = 1	-
	χ_{2nd} (Overall Availability)	**0.5**	**Normal = 1**	**-**

	Fuzzy Variable	**Value**	**Fuzzy Membership 1**	**Fuzzy Membership 2**
3rd Iteration	**α**	**0.592**	**Normal = 0.91**	**-**
	β	**0.5**	**Normal = 1**	**-**
	Predicted Battery Life	0.9	High = 0.5	Awesome = 0.5
	Computing Infrastructure Robustness	9.84	Great Robustness = 1	-
	χ_{3rd} (Overall Availability)	**0.579**	**Normal = 0.87**	**-**

Figure 58. Table containing data and results for the three experimentations (iterations) performed with different predicted battery levels.

An important remark: do not make confusion between the *partial output surfaces* of χ, indicated before as χ_1, χ_2 and χ_3 and the *overall availability value* obtained performing the three iterations with the different battery level, indicated as χ_{1st}, χ_{2nd} and χ_{3rd}. Note that if we want to obtain a *de-normalized value* for the overall availability, we can use a **mapping function** that for each value of the **normalized value** χ returns a **standard availability value A**, where **A = $f(\chi)$**. There are several possibilities here for the *mapping function* (as the *n-th root of* χ), each one with its advantages and disadvantages. Note that, adding sources of

uncertainty in the availability computation, the obtained value of availability shows a decreasing behaviour. This is reasonable, because considering more sources of uncertainty typically means have more problems (and a lower value) for the availability of the system. A last remark before the end of this chapter: we want to underline that if we perform a brief comparison with respect to the work of [*D. Perez-Palacin, R. Mirandola, 2014*] there are several differences in the operating approach and the goals pursued. First of all, our approach is a full immersion in the field of *Fuzzy Logic* and *Fuzzy Sets*, while the other work uses different (and specific) techniques in order to deal with the different aleatory/epistemic sources of uncertainty, as we have also investigated in the preliminary analysis of the problem **[Chapter 3.2]**. From one hand, our *Fuzzy 'modus operandi'* is an integrative approach; from the other hand the problems are solved from a higher level of abstraction. In fact, the [*D. Perez-Palacin, R. Mirandola, 2014*] approach is more detailed and case (uncertainty) specific.

Even more: the goals are a little bit different. In our proposal, we try to estimate the overall availability of the real world system under analysis, considering the contribution of input parameters, model context and model structure aleatory/epistemic uncertainty. In this original work every source of uncertainty has the same impact (and weight) on the final outcome, but is not difficult to imagine a more detailed extension with different weights. Some sources of uncertainty and related problems can be more damaging in a real world context. This is not just common sense; this is what we see every day with our eyes (in various fields). In the reference paper the main purpose is to mitigate the effect of uncertainty sources with specific methodologies and techniques. See once again **[Chapter 3.2]** for a first idea and the original paper for further details. We want to stress here that *parameters estimation* and *fuzzification* are often a complex design problem. We tried to follow a coherent methodology, but of course it is possible to redefine some variables, parameters or fuzzy memberships in order to achieve different results. The *MATLAB* code is available on a public *GitHub* repository for possible improvements and extensions.

In the next **[Chapter 5]** there is another goal, related to the prediction of future events (such as the online providers in the next time units), dealing with the uncertainty in the nature of the prediction, called *future parameters value*. Predictions regarding future values, in fact, have always a certain degree of uncertainty. But with the help of curve fitting and neural networks tools, we will try to do our best in order to handle this problem.

Chapter 5 – Neural Networks and online providers' prediction

5.1 A brief introduction to Neural Networks

In this section we will deal with the source of uncertainty called *future parameters value* already presented in **[Figure 7]**. In fact, we will try to build a model able to predict the future behaviour of third party providers and explain their dependence in crashes. In order to accomplish this mission, we will adopt the paradigm of Neural Networks. From a theoretical point of view, Neural Networks (or **ANN**, Artificial Neural Networks) are a computational metaphor inspired by studies of the brain and nervous system in biological organisms (in particular, the human beings). They are highly idealized mathematical models of how we understand and abstract the essence of these nervous systems. Such systems are *learning-based*: in fact, they progressively improve their performances to do specific tasks (for example classification). The main features of the Neural Networks are the following:

- It consists of many simple processing units, called *neurons*, that perform a local computation on their input to produce a certain output. They are represented graphically with circles with possibly multiple incoming arrows for the inputs and an outgoing arrow for the output. Typically, neurons are structured in *layers*, where each layer may perform different types of transformation over the given inputs.
- The network has a *learning algorithm* that lets it automatically develop its internal representation. The '*de facto*' standard learning algorithm is called *Backpropagation*; clearly in literature there exist several alternatives and variants of it.
- Many *weighted neuron interconnections*, which encode the knowledge of the neural network. The weights are updated with the iterations of the learning algorithm (called *epochs*).

One of the mostly widely used (and accepted) processing-unit models based on the **logistic function**, which has the *resulting transfer function* is given by:

$$\text{Output} = \frac{1}{1+ e^{-\sum(w_i \times x_i)}} = \frac{1}{1+ e^{-SUM}}$$

Where *SUM* is the aggregate of the weighted inputs, in formula: $\sum(w_i \times x_i)$ = SUM.

In **[Figure 59]** we can see the graphical representation of an artificial neuron. Starting from the right we have the inputs $\mathbf{x_i}$, their weights $\mathbf{w_i}$, the activation function (for example the logistic function, often indicated as **Σ** or ϕ) and the outputs $\mathbf{a_i}$. The bias is just a *'control input'* used in order to properly handle the activation function.

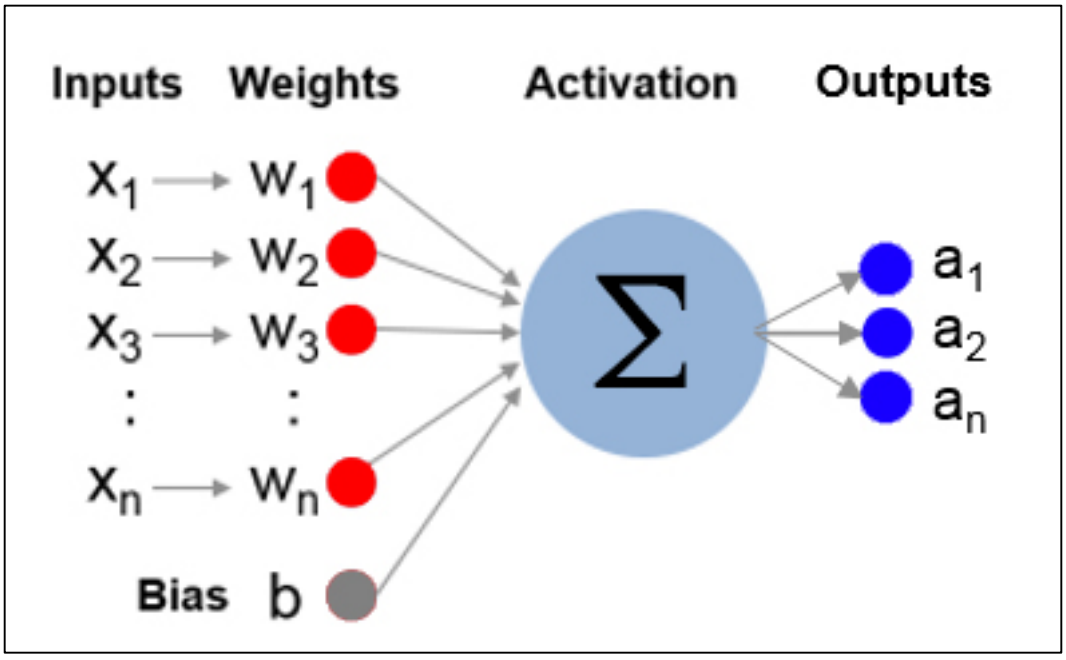


Figure 59. Graphical representation of an artificial neuron.

Richard Lippman describes many neural network models and learning procedure in his work [*Richard Lippman, 1987*]. There are two main well-known classes suitable for prediction application: *Feed-Forward Neural Networks* and *Recurrent Neural Networks*. Most recently there are some applications based on *Convolutional Neural Networks* (CNN*)* that aim to solve problems related to the robustness of analysis. In the last years the tendency is to go towards *Deep Learning* and *Deep Neural Networks*. A deep neural network (DNN) is an ANN with multiple hidden layers (middle layers) between the input and output layers. In a similar way with respect to the classical ANNs, DNNs can model complex non-linear relationships. The extra (hidden) layers enable composition of features

from lower layers, potentially modelling complex data with fewer units than a similarly performing shallow network. Each hidden layer has its own level of abstraction, concerning the features of the network. Deep architectures include many variants of a few basic approaches. Each architecture has found success in specific domains of application (from *Image Analysis* to *Recommender Systems*). It is not always possible to compare the performance of multiple architectures, unless they have been evaluated on the same data sets (sometimes it is also difficult to find 'fair' evaluation criteria). DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back (there is not recursion). *Recurrent neural networks* (RNNs), in which data can flow in any direction, are used for applications such as language modelling. A particular memory model called *Long Short-Term Memory* (LSTM) [*Sepp Hochreiter and Jürgen Schmidhuberm, 1997*] may be particularly effective for this use in general; but in our work we will use a **Nonlinear Autoregressive (NAR)** prediction model with *Focused Time-Delay Neural Network* (FTDNN) without LSTM (see **[Chapter 5.4]**). The following **[Figure 60]** shows the actual *Input/Output* response of the activation unit of neuron, where, as anticipated, *SUM* is computed as weighted sum of inputs. As we can understand from the mathematical expression and also from the graph, the function **Σ(SUM)**, or **ϕ(SUM),** is nonlinear and continuous (its domain is $\Re$, without points of discontinuity). In addition, we can visualize the *S-Shape* (sigmoid curve) in the range 0 to 1, used as activation function for the artificial neuron. We want to recall that it is a function of the interconnections weights $\mathbf{w_i}$ and the inputs $\mathbf{x_i}$ of the network.

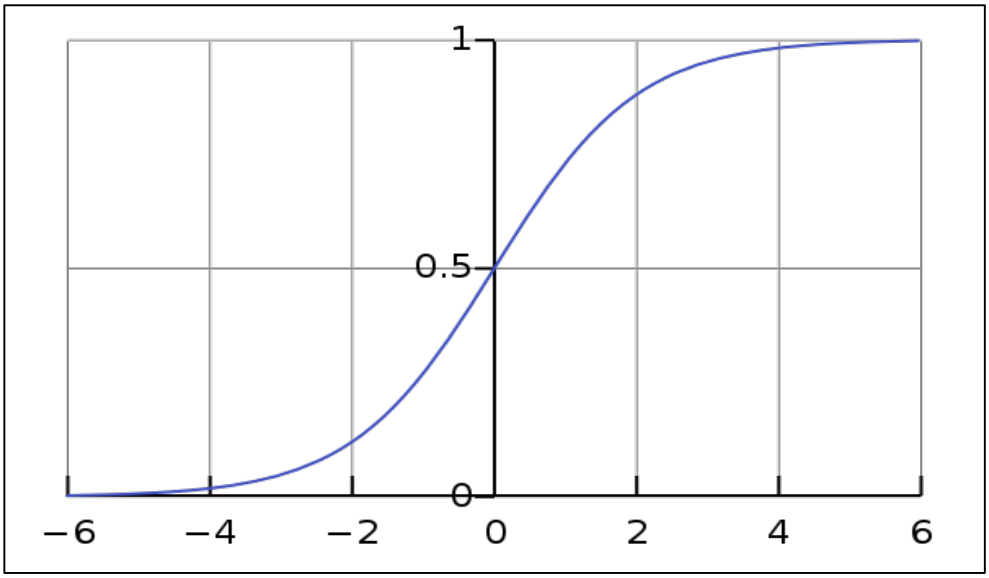


Figure 60. Logistic function (or logistic curve) graphical representation.

5.2 Neural Networks and providers: the approach

Here we start from real world data collected by the logs. In particular, we are working with data extracted from the file *skype.evt* available with [*Brighten Godfrey, 2010*]. In the original file, the information is not enough human readable and comprehensible, therefore the first step of the approach is a classical '*data cleaning*'. It is possible to identify three different fields in the log file:

1. The *Time unit*, where the providers become available (or not available);
2. The *Identifier* of the provider (an alphanumeric string);
3. A *Flag* for the specific provider action (*Up* or *Down*).

In order to understand the number of the providers available given a specific time unit, we performed a computation with simple *R scripting* on the original data: with a for cycle we added or removed providers to the total count depending on the specific flag (*ON/OFF*). Then we structured data in order to be able to import them into the *MATLAB* environment for further analyses There are two interesting factors here: the **time units** (or **timestamps**, with *granularity = 45 seconds*) and the corresponding **number of available providers**, that we can plot in order to visualize in a clear way what we are going to analyse later on:

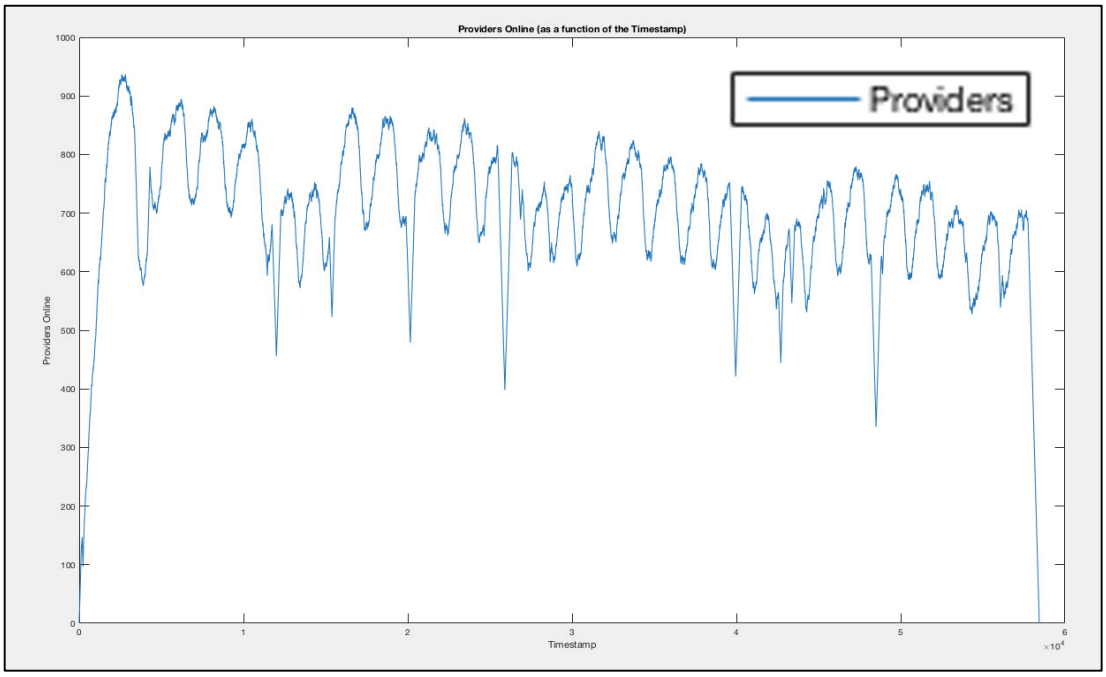


Figure 61. Number of online providers as a function of the time unit.

Now let us try to understand our goals. It is clear that our first goal is to understand if it possible to model in a mathematical way the behaviour of the providers as a function of the time unit. We can notice that there are two transients, at the beginning and at the end, that can create problems in the modelling phase (as we will see also later). In fact, we will perform two different analyses with the neural networks, one for the case with the transients and another one without them (with a proper subset of the data) and we will see the differences also from an applicative point of view. Another important goal is to investigate over the dependences of providers, because it can give a consistent explanation regarding their simultaneous crashes and can be used in order to improve the future predictions. It is clear, also from the **[Figure 61]** (and is highlighted in **[Figure 62]** with the red crosses) that in certain moments there are simultaneously crashes regarding a consistent number of providers. This can arise for several causes, as we have also explained in the previous chapters; but also here we are searching to model the behaviour with the support of the neural network paradigm. As a summary, the main objectives concerning the providers are the following two:

- Understand their number as a function of the time and try to predict their number in the future (*Prediction Problem*).
- Understand their dependence in crashes, trying to predict the next time where they are going to become unavailable at the same time (*Prediction Problem* and *Correlation*[17] *Problem*).

Now let us try to analyse the crashes from an intuitive and graphical point of view. In **[Figure 62]** the red crosses highlight the most evident points of local/global minimum of the function. They always follow a simultaneous crash of online providers (after the points of local maximum of online providers).

We have **58435 samples**, with an **overall observation time** of **31 days**. Therefore, we can compute that we have a flat number of **1885 observations per day**, which means **1 observation every 45 seconds** (the reference **time unit**). It is easy to deduce that there is on average a local minimum each day: this is likely related to the **day-night alternation** (with little variance in the weekends, as expected) of online providers, following the users' *demand curve*.

---

[17] Correlation is usually measured with a **correlation coefficient**, a number that quantifies a type of correlation and dependence, meaning statistical relationships between two or more values in fundamental statistics. The most common is the *Pearson product-moment correlation coefficient*, also known as *r*, *R*.

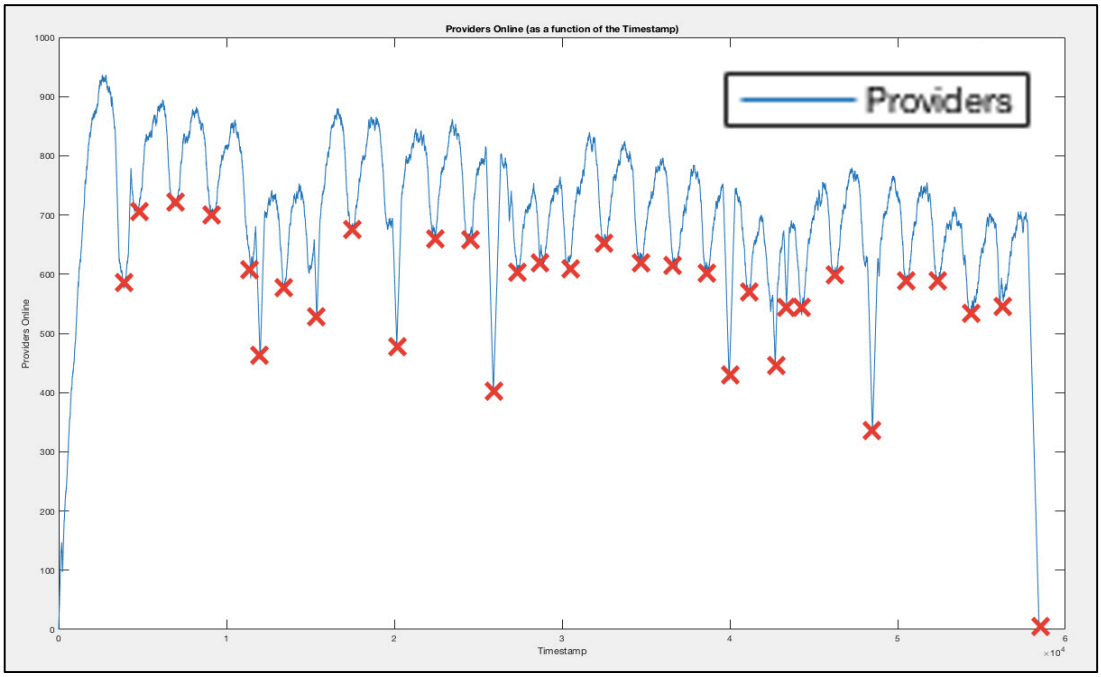


Figure 62. Online Providers, with the red crosses highlighting the most evident points of local/global minimum of the function.

Now, before entering in the details of the NN approach, let us see how the traditional methods (for example *Curve Fitting*) can perform in order to approximate the future behaviour of the third party providers. Then, after this first attempt, we will see how Neural Networks can give their contribution in order to solve this problem.

5.3 Preliminary Curve Fitting Analysis (with and without transient)

Let us present some interesting results obtained with the *MATLAB Curve Fitting Toolbox*. This approach can give us an idea about possible curve fittings and what to expect using traditional methods. The tool is full of possible fitting functions, but we are going to see only two of them: *Linear Fitting* and *Fourier Fitting* (with 1 term and 8 terms). First of all, let us show the fitting curves with transient (both for the beginning and for the end), then we will see the second case (without the transient).

In **[Figure 63]** we can see the online providers as a function of the time in the Curve Fitting Toolbox and a **Linear Fitting**. It is clear that here there's *Underfitting*: a linear function is not able to capture the oscillations of the real world curve.

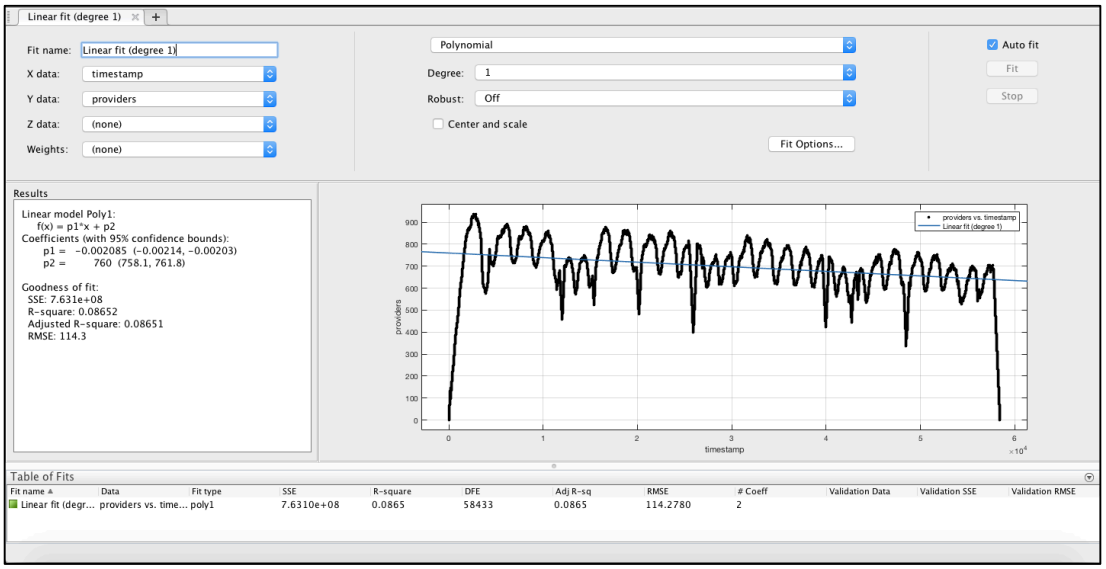


Figure 63. Online providers as a function of the time in the Curve Fitting Toolbox and a *Linear Fitting*.

In **[Figure 64]** there are the online providers as a function of the time in the Curve Fitting Toolbox and a **Fourier Fitting** (with 1 term). The fitting here works mediocrely (there is an undulatory behaviour, but with a lower frequency with respect to the target curve). As additional drawback, it does not capture the effect of the transients.

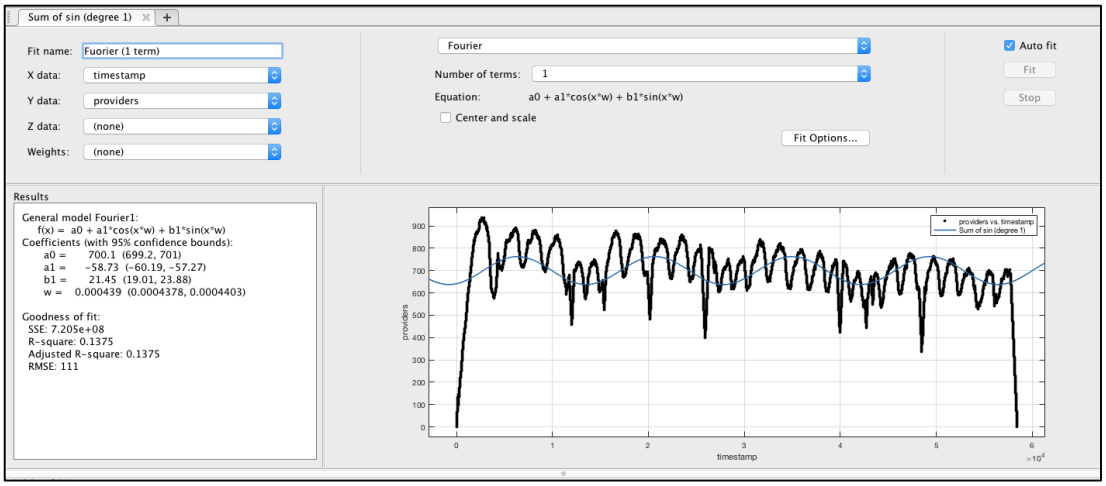


Figure 64. Online providers as a function of the time in the Curve Fitting Toolbox and a *Fourier Fitting* (with 1 term).

In **[Figure 65]** instead, we can notice the online providers as a function of the time in the *Curve Fitting* Toolbox and a **Fourier Fitting** (with 8 terms). As also the intuition suggests, this is the best fitting of the three. It captures both the effect of the initial and the final transient, it does not overfit the log data and on average fits the right value. But still the fitting is not enough satisfactory.

As additional remark, note that *Fourier Fitting* with more terms captures the effect of both initial and the final transient intervals, but its maximums, minimums and variation frequencies do not match the ones in the trace.

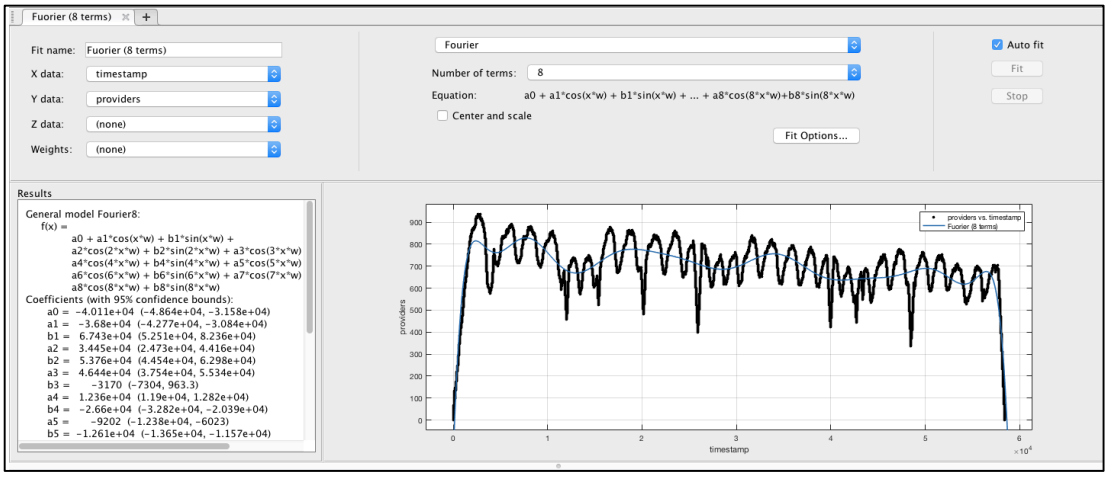


Figure 65. Online providers as a function of the time in the Curve Fitting Toolbox and a *Fourier Fitting* (with 8 terms).

Now let us present the differences with respect the version without the transient. We decided to cut the initial values (and the final ones) under the number of 600 online providers, performing a subset on the original data. From the original *58435 measures* we have now a subset of *56631 samples*[18]. Let us understand how the curve fitting changes in this second case (that we are also going to recall later on with the Neural Network approach).

In **[Figure 66]** is presented the new version of the graph, where we have the number of online providers as a function of the time unit (version with cut transients).

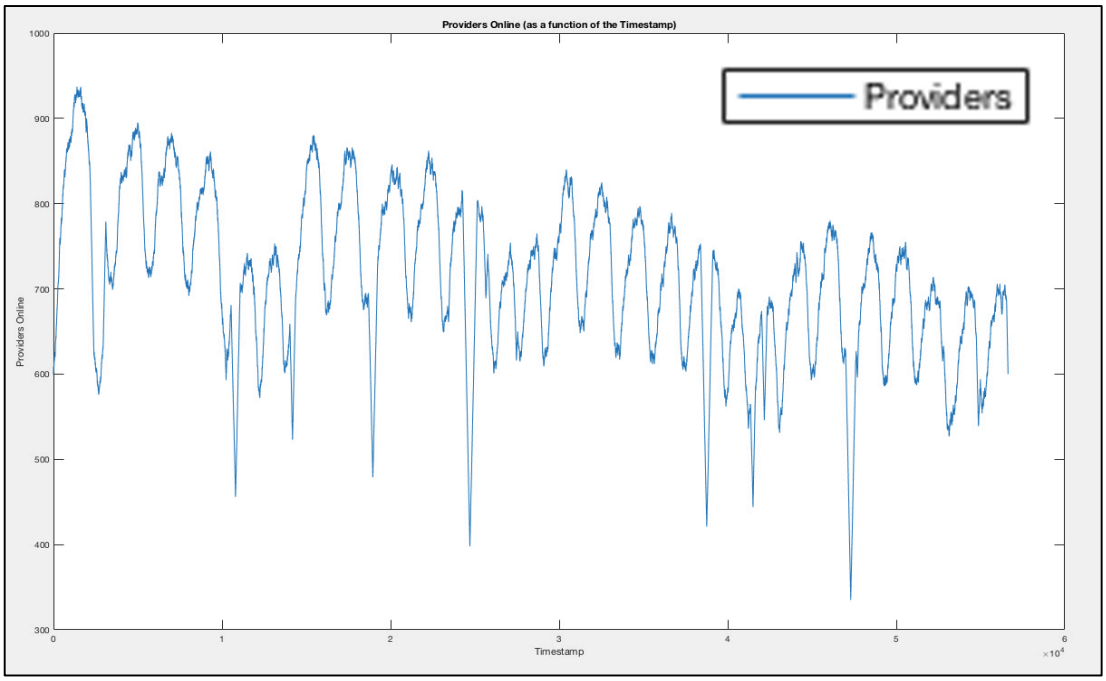


Figure 66. Number of online providers as a function of the time unit (version with cut transients).

In **[Figure 67]** we have the online providers as a function of the time in the Curve Fitting Toolbox and a **Linear Fitting** (without transient). Also here, it is clear that here there's *Underfitting*: a linear function is not able to capture the oscillations of the real world curve.

[18] We therefore cut a total of **1804 samples** (1204 from the initial transient and 600 from the final transient).

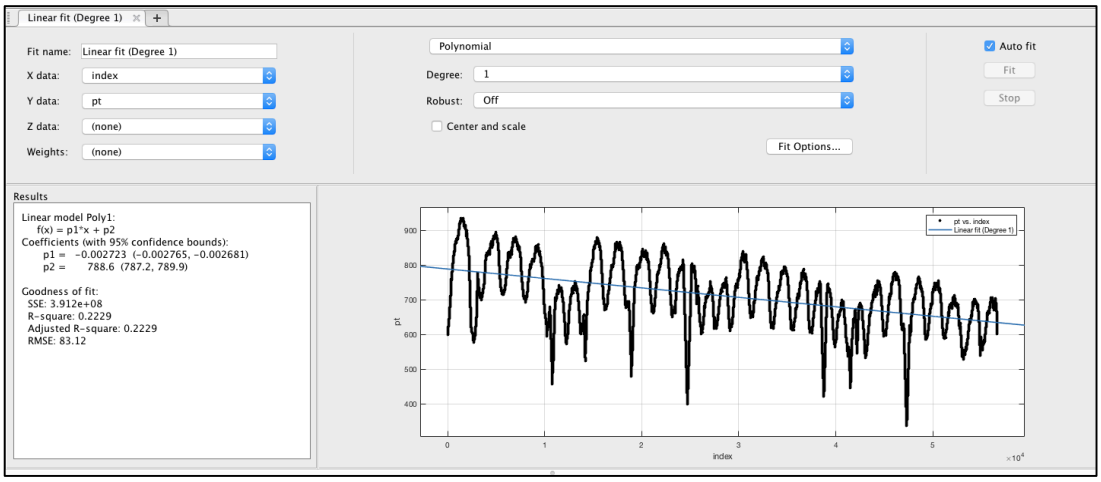


Figure 67. Online providers as a function of the time in the Curve Fitting Toolbox and a *Linear Fitting* (without transient).

In **[Figure 68]** there are the online providers as a function of the time in the Curve Fitting Toolbox and a **Fourier Fitting** (with 1 term). As we can see here the coefficients for the sinusoidal and cosinusoidal factors have a very low impact in the curve fitting, therefore the behaviour of the best fitting is very similar to the previous (*linear*) case, treated in the previous **[Figure 67]**. Even more, here the fitting is worse than the *Fourier Fitting* (1 term) with the entire trace **[Figure 64]**.

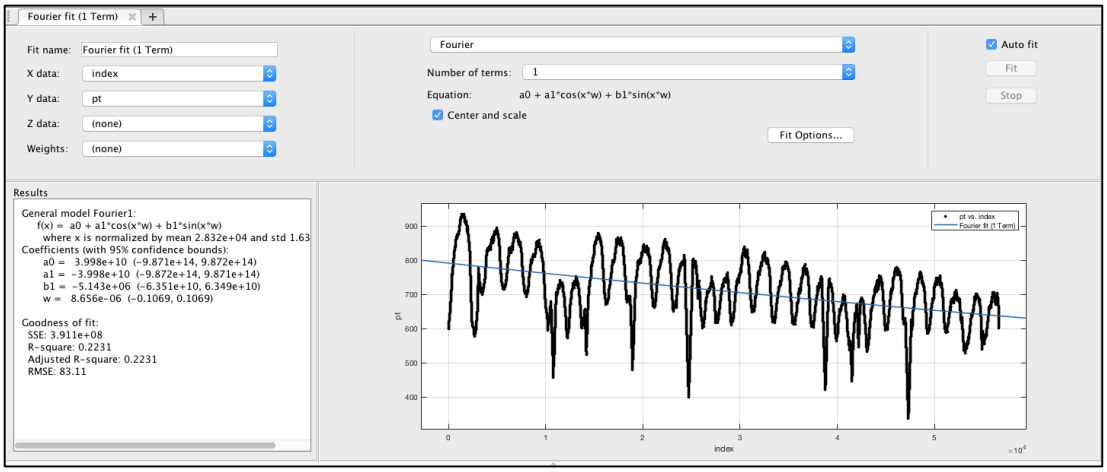


Figure 68. Online providers as a function of the time with *Fourier Fitting* (1 term).

In **[Figure 69]** we can see the online providers as a function of the time in the *Curve Fitting Toolbox* and a **Fourier Fitting** (with 8 terms). It still captures both the effect of the initial and the final transient (also after the elimination of it, due to the automated code of the Toolbox). A translation towards the abscissa axis

can be helpful in order to fit without this effect and obtain an efficient curve fitting (for the effective values then we have just to undo the translation).

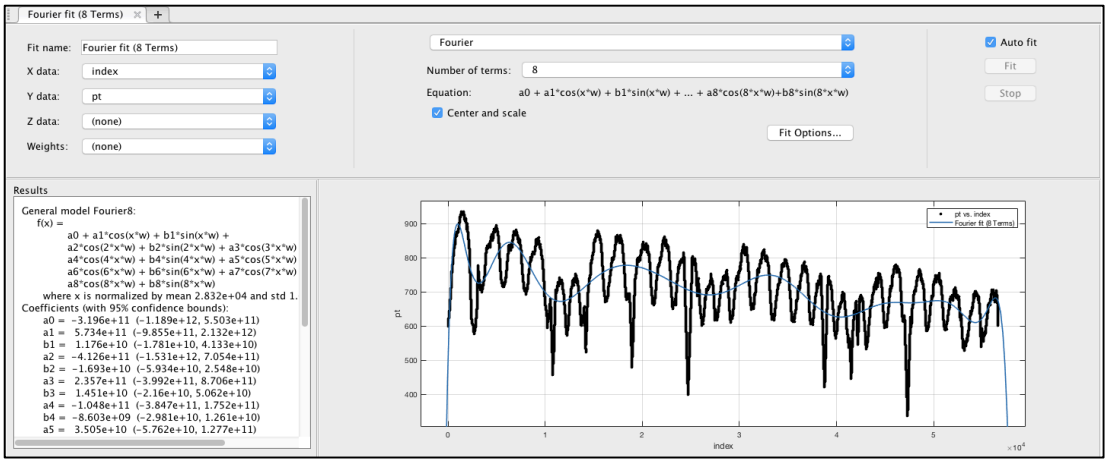


Figure 69. Online providers as a function of the time with *Fourier Fitting* (8 terms).

5.4 Neural Networks and providers (with transient): procedure, results and considerations

After the *Curve Fitting* introduction, and having noticed its limitations for prediction, it is now the time to explore the paradigm of **Neural Networks**, and see what contribution they can give us in order to achieve our goals. Also in this case, we will work with *MATLAB* instruments: in particular we will use the *Neural Network Toolbox* for **Time Series Analysis**. In fact, we can see the number of providers as discrete time series, where each sample is a function of the time unit. There are three possible prediction models available in the tool:

- **Nonlinear Autoregressive with External (Exogenous) Input (NARX),** which predicts series **y(t)** given **d** past values of **y(t)** and another exogenous series **x(t)**.
- **Nonlinear Autoregressive (NAR)**, which predicts series **y(t)** given **d** past values of **y(t)**, without the contribution of external variables. See the block diagram directly in **[Figure 70]**.
- **Nonlinear Input-Output,** which predicts series **y(t)** given **d** past values of **x(t)**, when the values of **y(t)** are not available during the deployment.

We will use the second model, that suits perfectly our situation. We have the real world observation **y(t)** available (without **x(t)**, the influence of external series) and we want to build a model for the prediction of the future values, based on the Neural Network. We have anticipated some features about the Neural Networks previously **[Chapter 5.1]**. Note that we are going to add information and explanations during their direct usage here and in the following chapter.

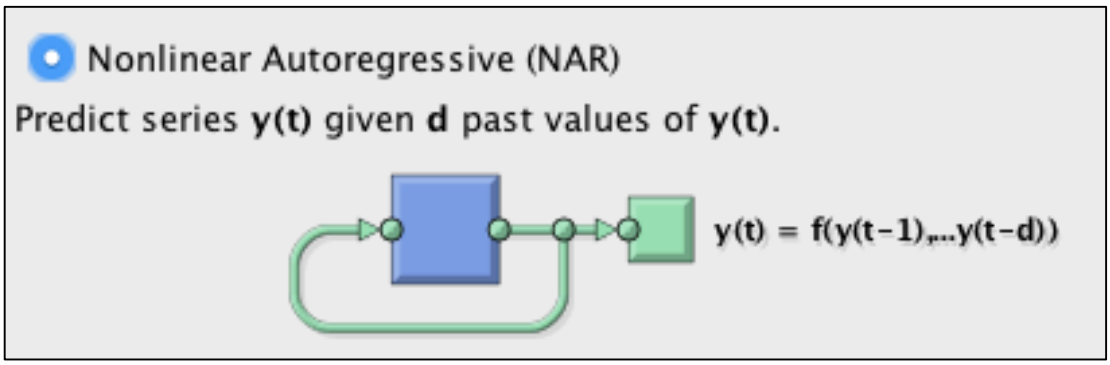


Figure 70. *Nonlinear Autoregressive (NAR)* block diagram model.

After the selection of the data for the *Time Series Analysis* (in our case the succession of providers' number), it is the time to perform a subdivision between *Training Data, Test Data* and *Validation Data* for the Neural Network. We will choose a classical *70%-15%-15%* random subdivision (default settings, coherently with respect to the subdivision proposed in [*Dobbin and Simon, 2011*], see **[Figure 71]**). Further experiments made us note that, for the case study used, different subdivisions give little differences in the final results (as also the re-training of the network with different random samples subsets or a different training algorithm).

Recalling the **Neural Network Theory** [*Kurt Hornik et al. 1989*] *training data* are the ones used in order to calibrate the weights during the training phase (according to their error), *validation data* are used in order to measure network generalization (avoiding overfitting, the validation halts training when generalization stops improving) and *testing data* are used in order to measure network performance during and after training (we will use the *MSE, Mean Squared Error*, as quality criterion).

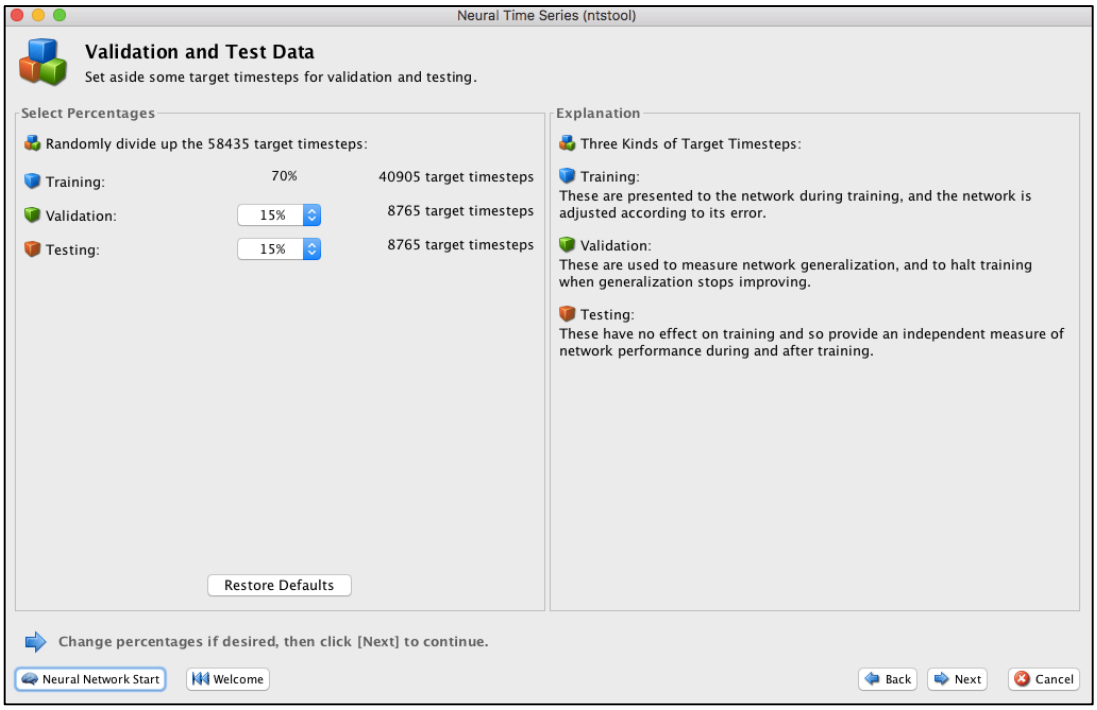


Figure 71. MATLB graphical interface that shows the target data subdivision between *Training* (40905 targets, 70% of data), *Validation* (8765 targets, 15% of data) and *Testing* (8765 targets, 15% of data).

The next step is the one related to the Neural Network parameters. Beginning with the most straightforward dynamic network, which consists of a feedforward network with a tapped delay line at the input. This is called the focused time-delay neural network (*FTDNN*). This is part of a general class of dynamic networks, called *focused networks*, in which the dynamics appear only at the input layer of a static multilayer feedforward network. Knowing that we are dealing with a non-trivial network for *Time Series Prediction*, we have to calibrate the number of neurons for the hidden layer $\mathbf{N_{HL}}$ [*G. Brightwell et al. 1996*] and the numbers of delays **d** [*Bert de Vries et al. 1990*].

The network will be trained in *Open Loop* (without retroaction of **y(t)** values, see **[Figure 72]**) for efficiency purposes, but then the prediction of the future values will be performed in *Closed Loop* (with retroaction of **y(t)** values). We will use for the training *10 hidden neurons* and *10 hidden delays*, knowing that we need the right compromise between complexity and generalization capabilities (in order to avoid both *Underfitting* and *Overfitting*).

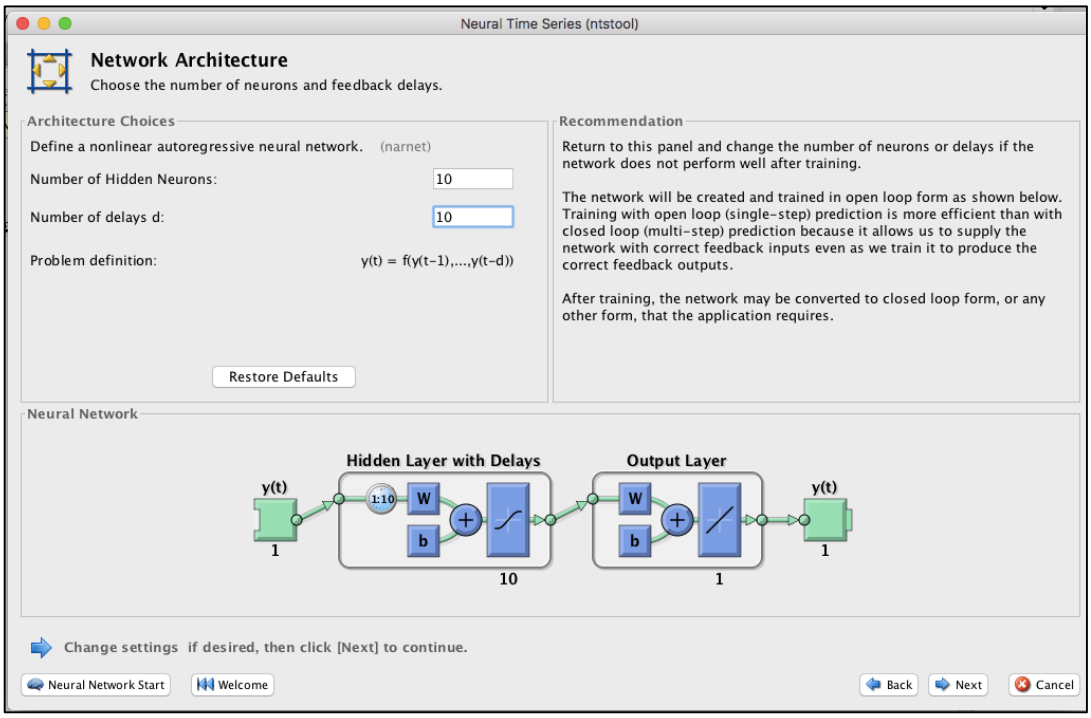


Figure 72. MATLAB graphical interface that shows the selected number of *hidden neurons* ($\mathbf{N_{HL}}$ = 10) and *delays* (**d** = 10), with the diagram of the network that we are going to use in the training phase of the Neural Network.

There are three neural network training algorithms available in the *MATLAB Toolbox*, we will use the *trainlm* (**Levenberg-Marquardt backpropagation**) which is very fast and allow us to reach good results [*S.Sapna et al. 2012*]. The other two algorithms are more '*memory efficient*', but request more time or have less performances. We are interested in understanding how the neural network outputs differ from the known targets; a high degree of *similarity* is a concrete feedback of a successful *training phase*. After this initial setting-up phase, we launched the first simulation with this first network.

In **[Figure 73]** and the following ones **[Figures 74-79]** we can see some details and results of the training phase.

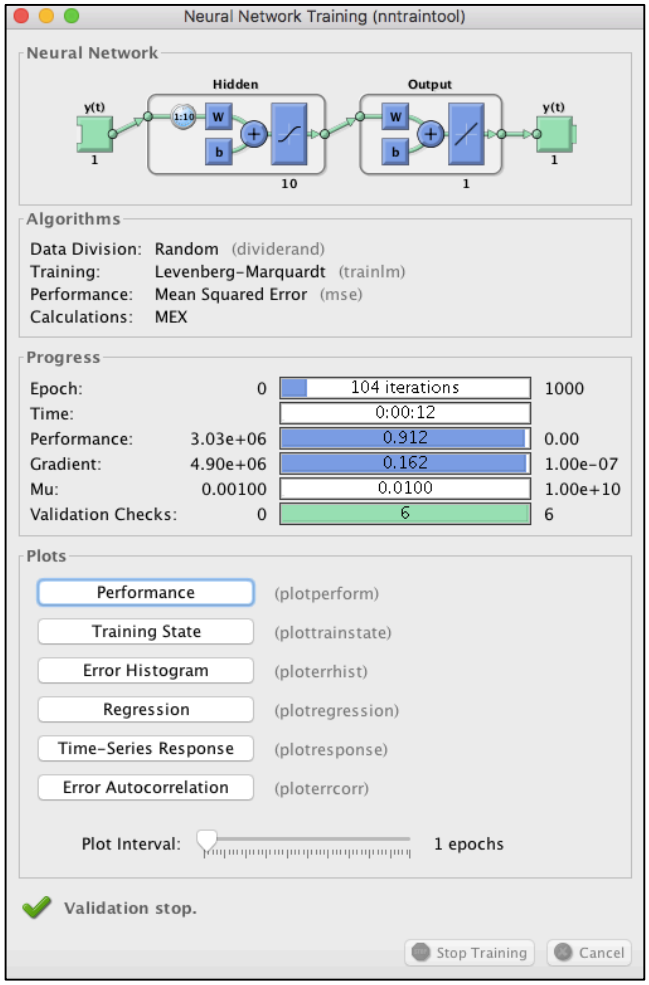


Figure 73. Details of the first neural network trained. The *Levenberg-Marquardt* backpropagation algorithm in this case took 104 iterations (epochs) in order to achieve the best performances and generalization, according to the Validation Checks.

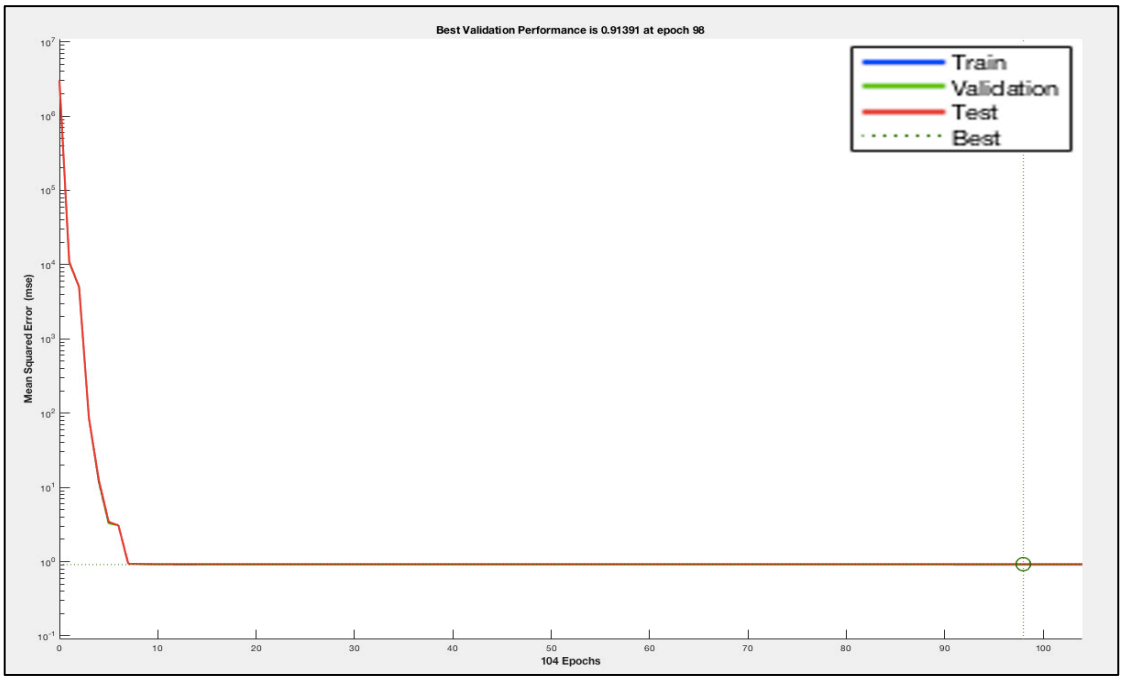


Figure 74. Graph of the MSE (*Mean Squared Error*) concerning Training, Validation and Test Data as a function of the backpropagation epochs. According to the computations, the best Validation performance is $MSE_{VAL} = 0.91391$ at epoch 96.

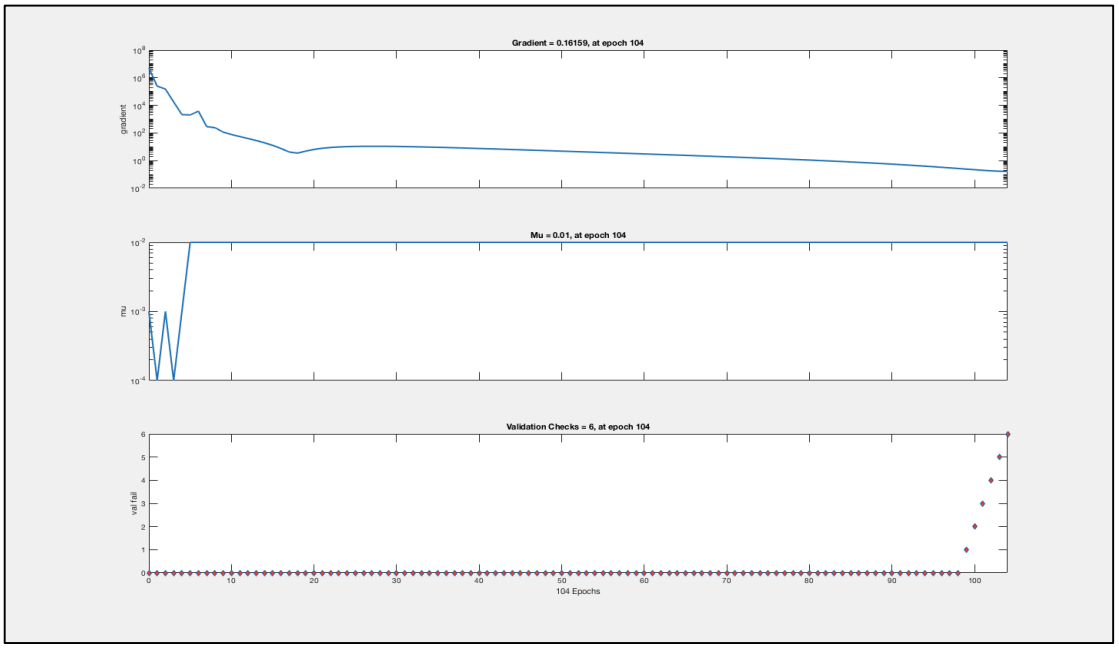


Figure 75. Three graphs representing respectively the *Gradient* (for the backpropagation), *Mu* (μ, control parameter for the backpropagation convergence) and the *Validation Checks* (halting and generalization) as a function of the training epochs.

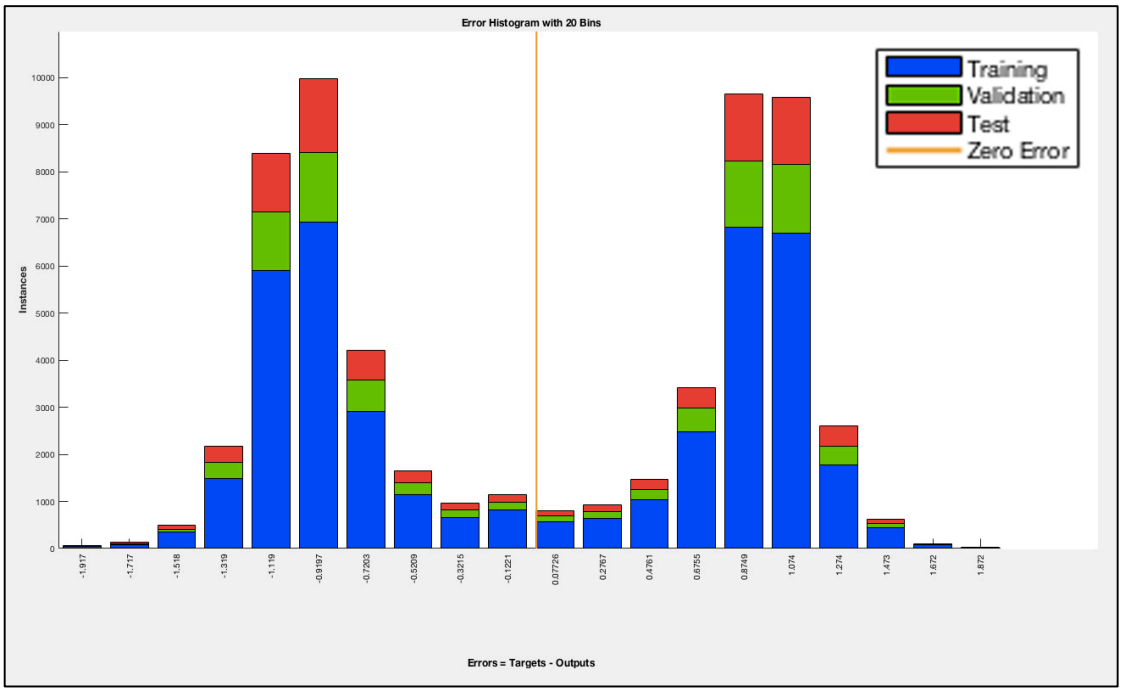


Figure 76. Distribution of the error (Output-Target) concerning Training (in blue), Validation (in green) and Test (in red) Data. As we can observe, it has its maxima around ±1.

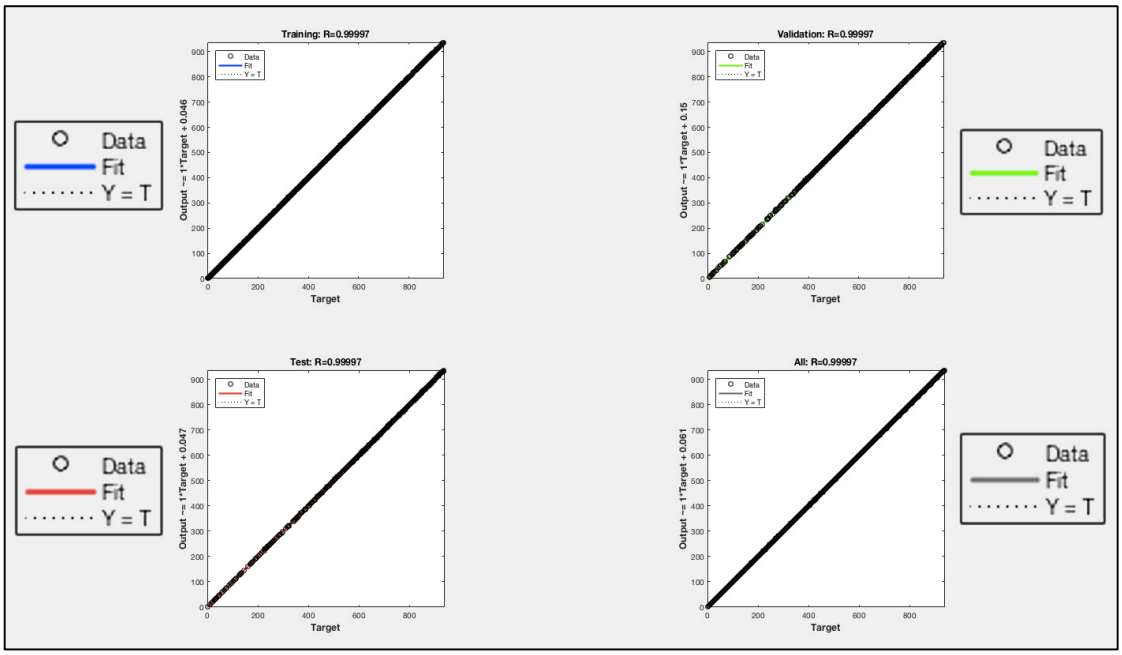
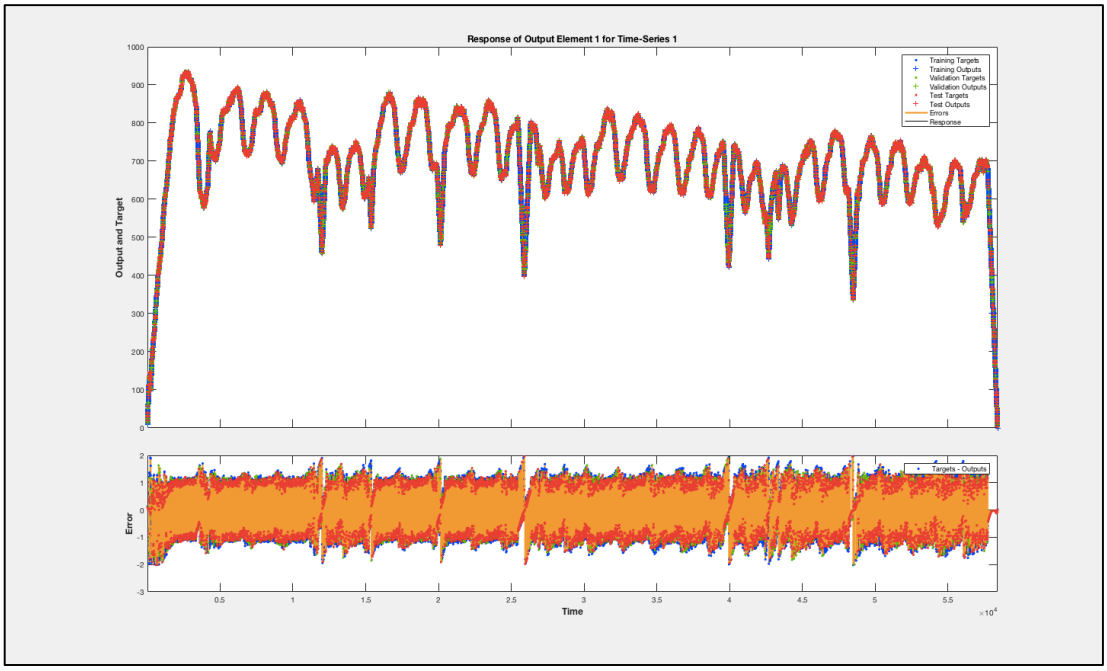


Figure 77. Output-Target correlation is very high (Regression Value R= 0.99997).



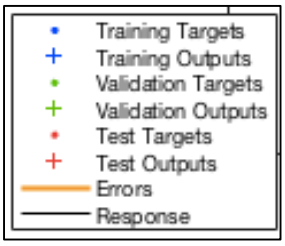


Figure 78. Neural network response (output and target) with highlighted error for the considered time series. This graph is full of information; for a better comprehension we have the legend on the left which explains the different components (Training / Validation / Test Targets and Outputs, Errors and Response).

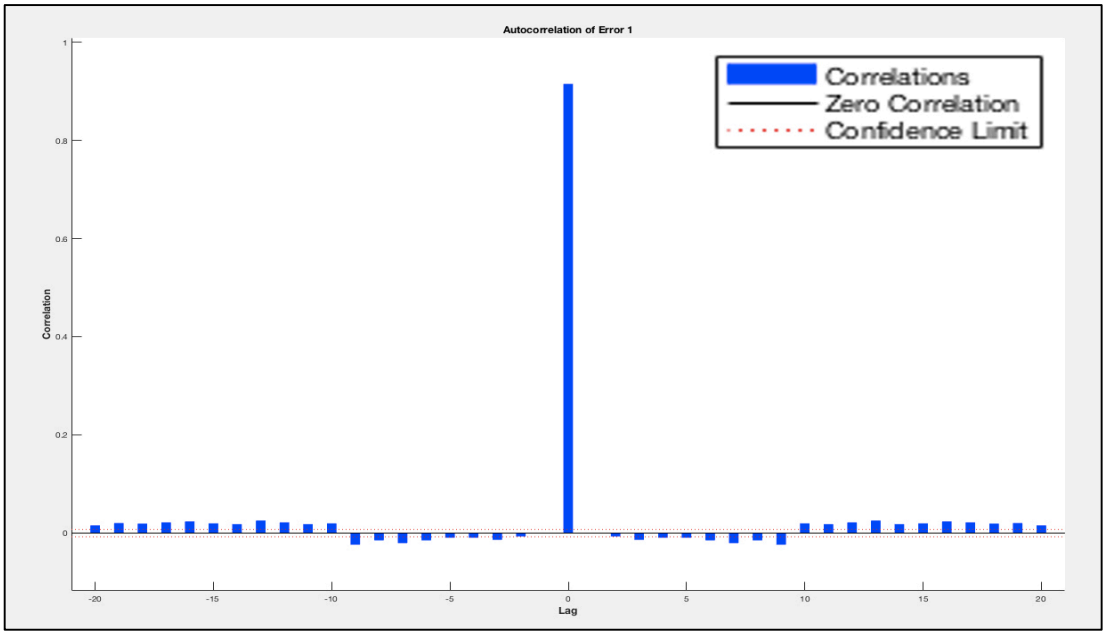


Figure 79. Autocorrelation plot for the error (with very low values, with Lag ≠ 0).

Let us present some analyses and considerations on this first training phase, looking at the training results and the previous figures. Regarding the *training duration*, we can say that the network spent **104 iterations** (epochs). The registered *execution time* lasted exactly **12 seconds**. The *halting criterion* of this training phase was the **Validation Stop**[19] (instead of the *training epochs' limit*, set to 1000 in *MATLAB*). See **[Figure 73]** for further details.

Regarding our performance criterion, the **MSE** (*Mean Squared Error*), we obtain its best validation performance $\mathbf{MSE_{VAL}}$ = **0.91391** at **epoch 96** (full behaviour directly in **[Figure 74]**). Regarding the technicalities of the *Backpropagation* algorithm, it is possible to observe the behaviours of the *gradient curve* (how the training improves epoch after epoch), the *control parameter* μ (which is the *learning rate* in the backpropagation algorithm) and the *validation checks* (in order to test the level of *generalization* of the network) as function of the epochs directly in **[Figure 75]**.

Another interesting graph is the one presented in **[Figure 76]**, where it is plotted the distribution of error (*Output vs Target*) for each of the three data sets (*Training, Test* and *Validation*). It is clear that the maximum error is performed in the training phase, because the network is still learning how to predict new data (in fact the blue histogram of *Testing Data* is always the more consistent). Another interesting observation here is that the prediction error has it maxima around **±1**. This means that the prediction for a given time unit is often '*wrong*' just for plus or minus one provider, as it is also reasonable to think.

Furthermore, the correlation between output and target is very high as ideally expected, with a regression coefficient **R = 0.99997**; see **[Figure 77]** for additional details. We recall here that **R** is the **Pearson's coefficient**, a measure of the strength and direction of the linear relationship between two variables that is defined as the (sample) covariance of the variables divided by the product of their (sample) standard deviations. The more the value is close to one, the more the variables are correlated. The obtained value suggests a high correlation between outputs and targets as expected (in the optimal case they should

[19] **Validation Stopping** is used to express that even though the *training set error* is decreasing, the *non-training validation set error* (and possibly, the *non-training test set error)* is increasing. Since we are designing a network in order to predict new (unseen) data, it does not make sense to continue training (therefore the training phase stops).

represent exactly the same information; in this situation we are very close to this case).

One of the most informative graph is probably the one presented in **[Figure 78]**, where are plotted the *Training, Test* and *Validation* samples both for Targets and Outputs, with the highlighted error above for the whole time units' interval. It is almost constantly distributed (as often happens for *periodic* or *almost-periodic*[20] functions, see [*Shilder et al. 2005*] for additional details).

In the final **[Figure 79]** is summarised the *autocorrelation* of the samples. We want to recall that the autocorrelation, also known as *serial correlation*, is the correlation of a signal (or a time series) with a delayed copy of itself as a function of delay. Informally, it is the similarity between observations as a function of the time lag between them. We have little values of autocorrelation in the graph (for **Lag ≠ 0**), because it covers a little interval of observation (only **40 samples**, while the observed providers' behaviour seems to be a *quasiperiodic function* with *quasi-period* **T = 1885 samples**, or equivalently, **T = 24 hours**[21]).

After the training phase we are now trying to see the results of the neural network regarding the prediction of online providers in the future. Just before the initial considerations and results, let us present three schemata of the neural network: the first one **[Figure 80]** is the one used during the training phase, the second one **[Figure 81]** is the one used for the *N-steps ahead closed loop prediction*, while the third one **[Figure 82]** is very similar to the first one, but one delay is removed from the network and the prediction is *y(t+1)* instead of *y(t).*

---

[20] In mathematics, an **almost periodic function** (or *quasi-periodic function*) is, loosely speaking, a function of a real number that is periodic to within any desired level of accuracy, given suitably long, well-distributed *'almost-periods'* (or *'quasi-periods'*).

[21] We already introduced the information concerning the samples and the overall observation time (with additional details) in **[Chapter 5.2]**.

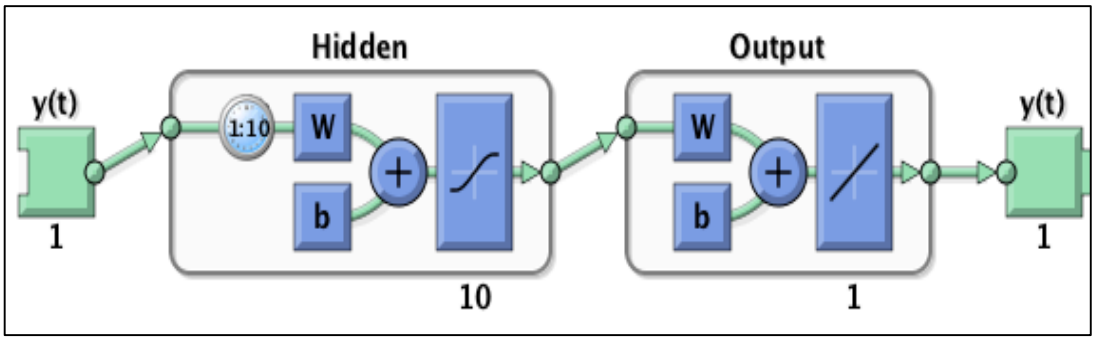


Figure 80. Open Loop Neural Network schema used in the training phase.

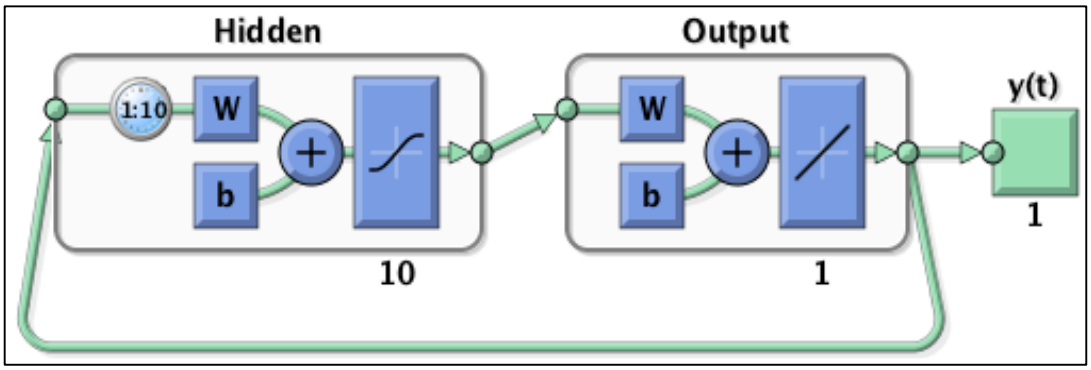


Figure 81. Closed Loop Neural Network schema used for multi-step prediction. It computes new values of *y(t)* given the previous values of *y(t)*.

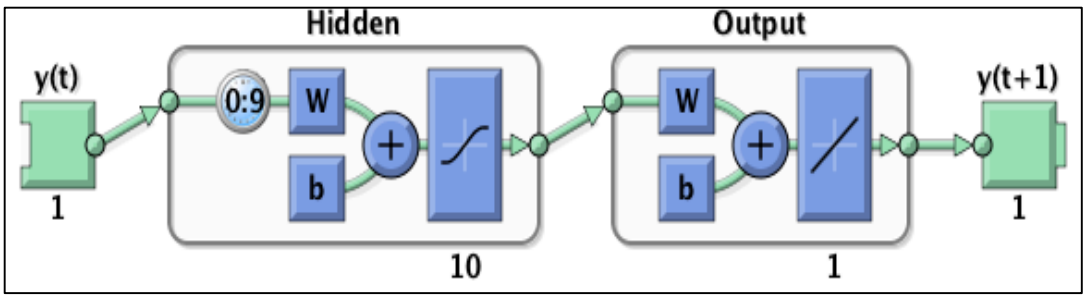


Figure 82. One-step ahead alternative for the Open Loop Neural Network schema.

Now let us try to understand how this first network performs in the predictions of new online providers' values. In **[Figure 83]** are plotted the values for the first 50 time units' previsions according to the network.

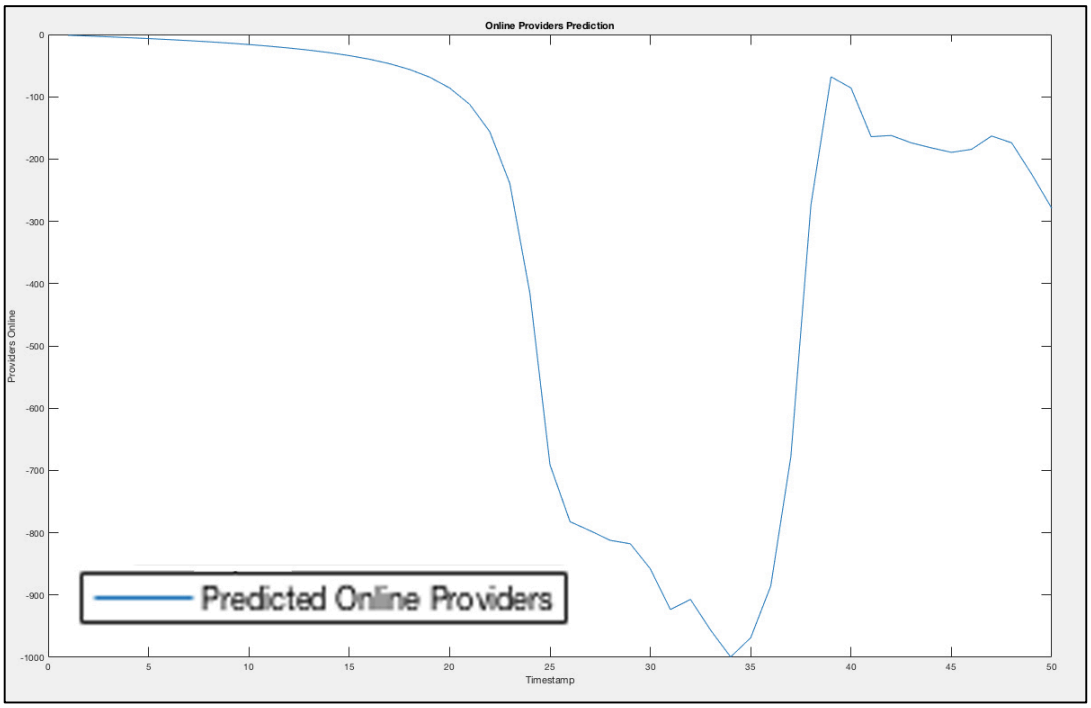


Figure 83. Neural Network (negative) predictions for the next 50 time units.

Here it is clear that with the considered data we have a big problem regarding future predictions. The *Closed Loop Performance* is absolutely bad, and this was emphasized in the simulation phase, for the prediction of new providers' data. In fact, we obtain unreasonable negative values (also very big numbers, with a maximum in -1000 providers).

The main problem here was the steep final transient, that suggests to the network a monotonic descendent behaviour (at least for a certain amount of time). This has not a sense in the practical application field (a negative number of online providers in meaningless), but would capture interesting behaviours in other types of applications, with similar functions.

In next chapter we will cut off the initial and the final transients of the trace log and we will re-execute the process in order to visualize the quality of the prediction on a log that shows continuous operation of the system.

5.5 Neural Networks and providers (without transient): procedure, results and considerations

The main problem for the accuracy of the prediction in the last chapter was the influence of the transient (especially the final transient, which gave a steep decrease from a big number of online providers to zero in the context of a little time interval). This had a considerable drawback regarding the prediction, which suggests to the network to continue to predict the descendent behaviour (at least for a certain time interval), with unrealistic negative values as final outcomes. In a suitable, reliable and robust real wold application, where a minimal functioning of the service is guaranteed, there can be simultaneous crashes of course, but without complete malfunctioning (all servers down). Therefore, it is reasonable to consider the same dataset without the influence of transients. After the previous (problematic) experiment, let us try to consider now the dataset without transient data.

As anticipated previously, with the curve fitting analysis in **[Chapter 5.3]**, we decided to cut the initial values (and the final ones) under the number of 600 online providers, performing a subset on the original data. We recall here that from the original *58435 samples dataset* we have now a subset of *56631 samples*. See **[Figure 66]** in order to recall also the shape of the *online providers' curve* as a function of the time unit. The parameters for the neural network training will be the same of the previous chapter. In the following **[Figures 84-95]** we can see the obtained results of this new training phase:

Results

	Target Values	MSE	R
Training:	39641	9.27404e-1	9.99947e-1
Validation:	8495	9.31334e-1	9.99947e-1
Testing:	8495	9.27377e-1	9.99948e-1

Figure 84. Training results with Target values, MSE and correlation value R for *Training* (39.641 samples), *Validation* (8.495 samples) and *Testing* (8.495 samples) Data.

After the training we will see the predictions both for the next 500 and 10.000 time units and we will start to visualize a first positive and meaningful attempt to predict reasonable values of providers online. This will confirm our hypothesis for the problems related to the final transient for the closed loop prediction.

In **[Figure 85]** there are the details of the second neural network trained (a subset of the first one, but without transients). The *Levenberg-Marquardt backpropagation algorithm* in this case took 47 iterations (epochs) in order to achieve the best performances and generalization, according to the Validation Checks. Note that the training was significantly faster with respect the previous case (in terms of epochs and execution time), due to less data to consider and the absence of particular behaviours such as the transient effect.

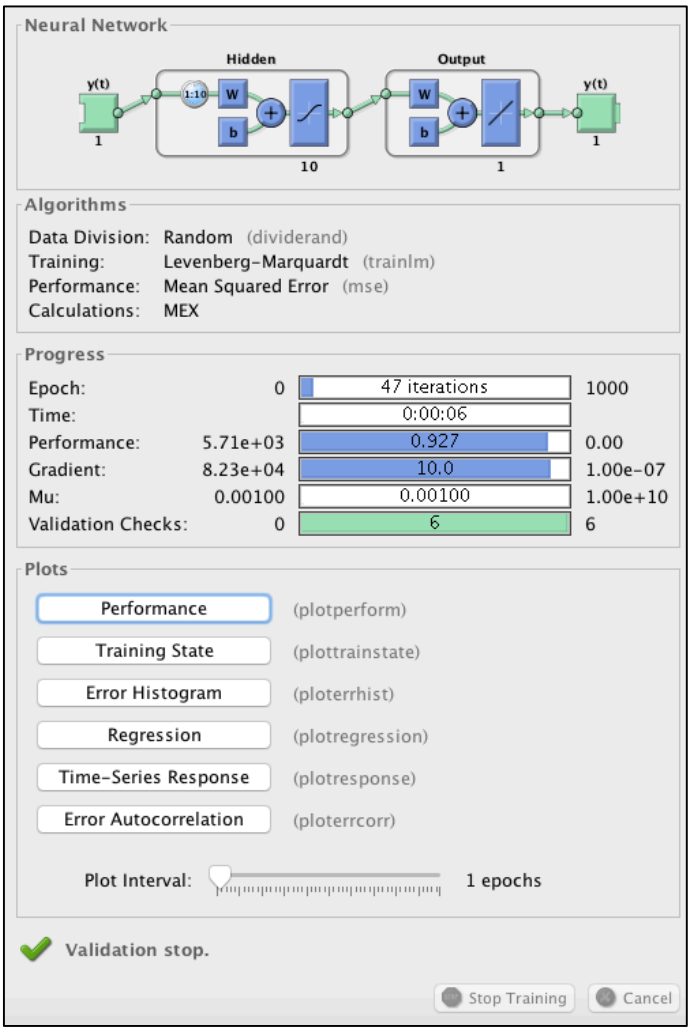


Figure 85. Details of the second neural network trained (using a subset of the input that was provided to the first one, removing the transients).

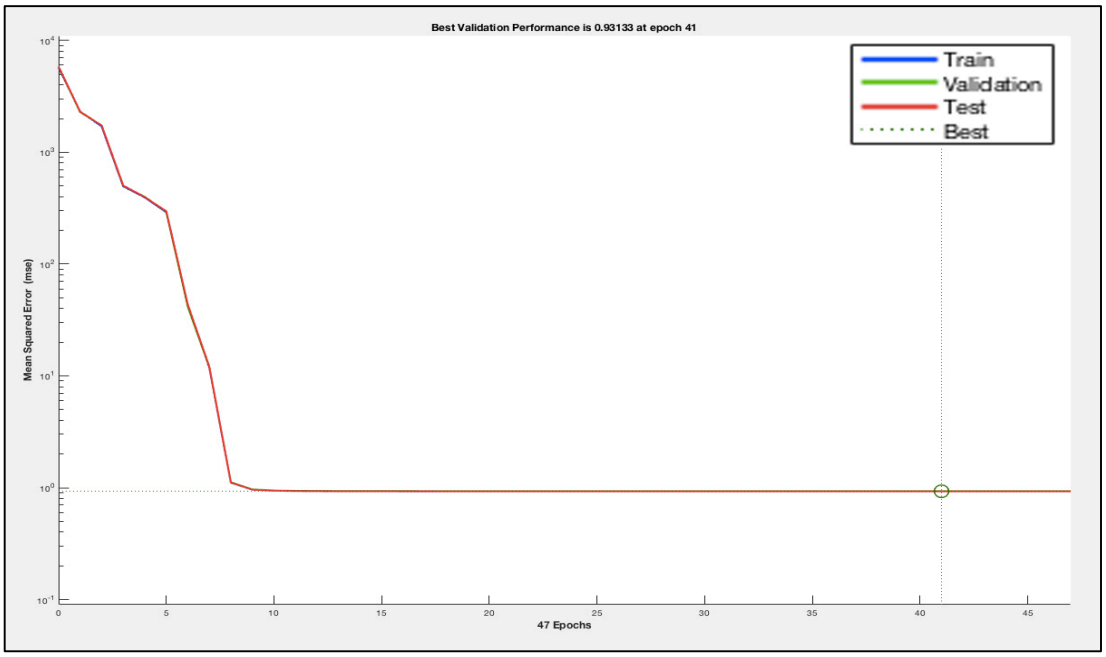


Figure 86. Graph of the MSE (*Mean Squared Error*) concerning Training, Validation and Test Data as a function of the backpropagation epochs. According to the computations, the best Validation performance is $MSE_{VAL} = 0.93133$ at epoch 41.

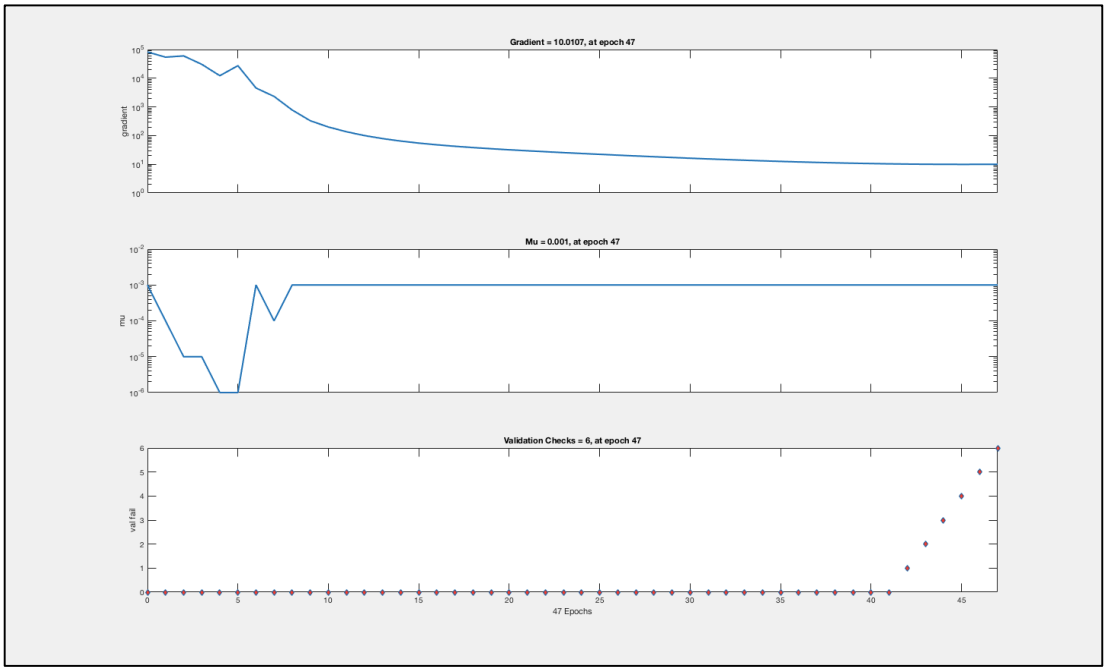


Figure 87. Three graphs representing respectively the *Gradient* (for the backpropagation), *Mu* (μ, control parameter for the backpropagation convergence) and the *Validation Checks* (halting and generalization) as a function of the training epochs.

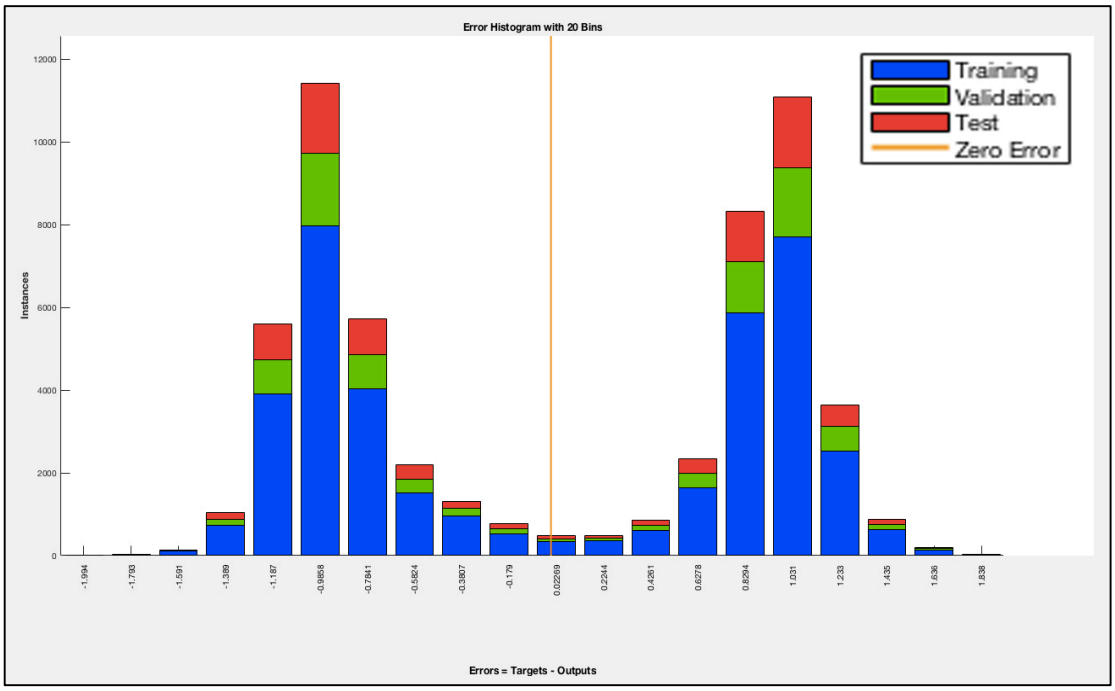


Figure 88. Distribution of the error (Output-Target) concerning Training (in blue), Validation (in green) and Test (in red) Data. As we can observe, it has its maxima around ±1 as in the case of the previous dataset.

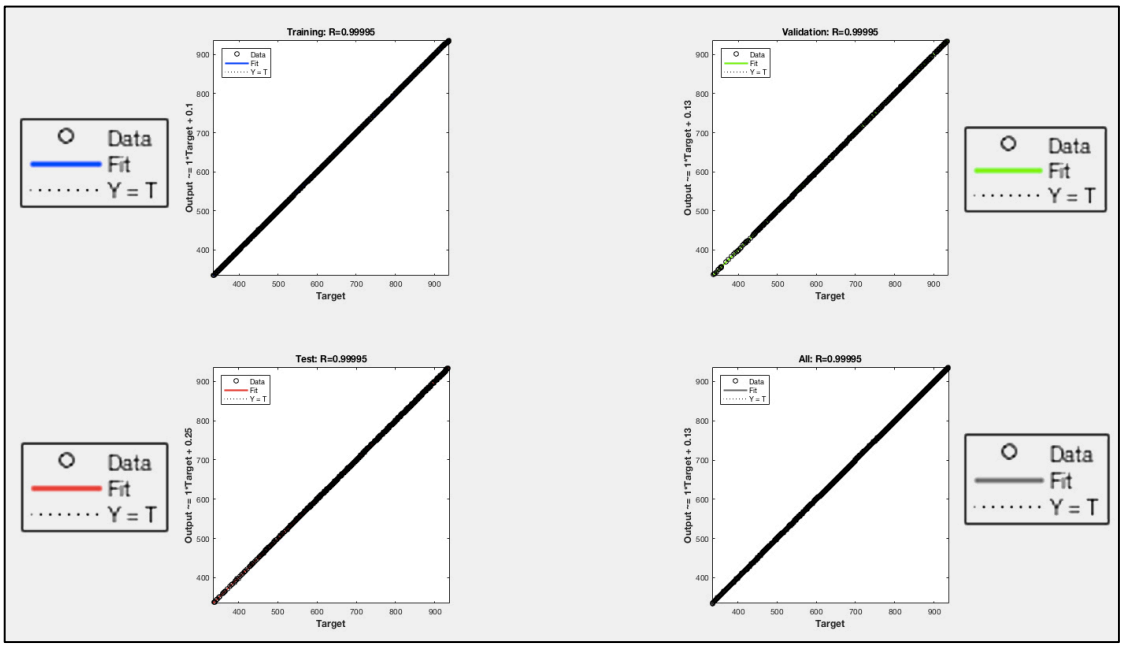
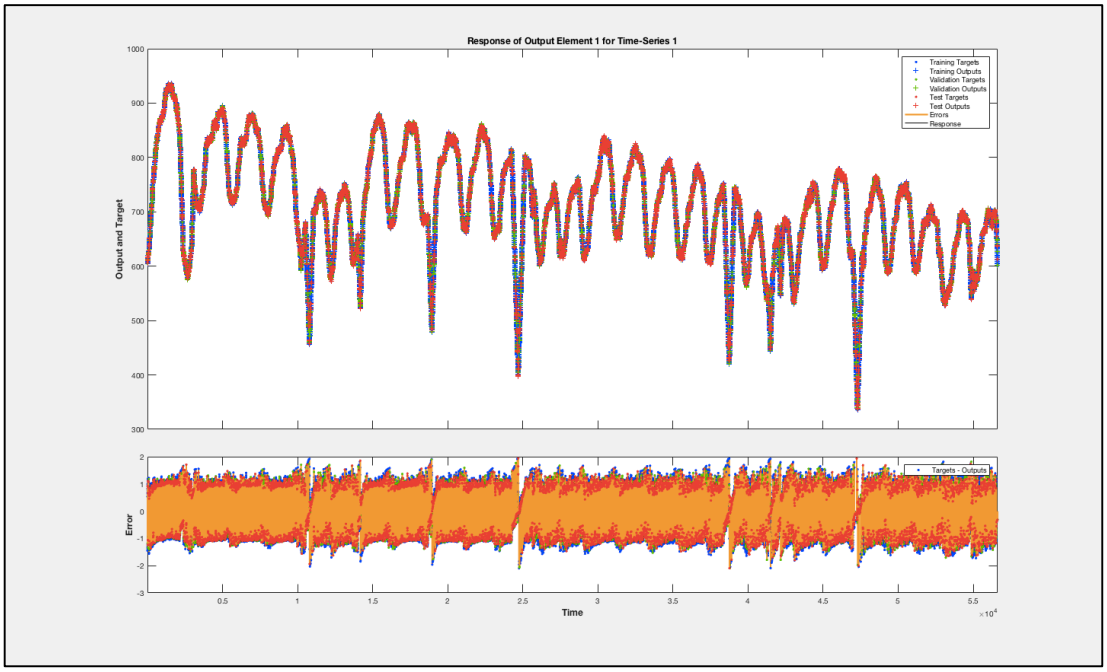


Figure 89. Output-Target correlation is very high (Regression Value R= 0.99995).



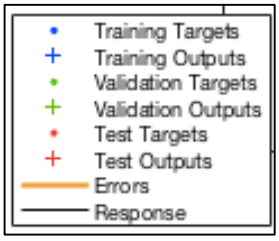


Figure 90. Neural network response (output and target) with highlighted error for the considered time series. This graph is full of information; also here for a better comprehension we have the legend on the left which explains the different components (Training / Validation / Test Targets and Outputs, Errors and Response).

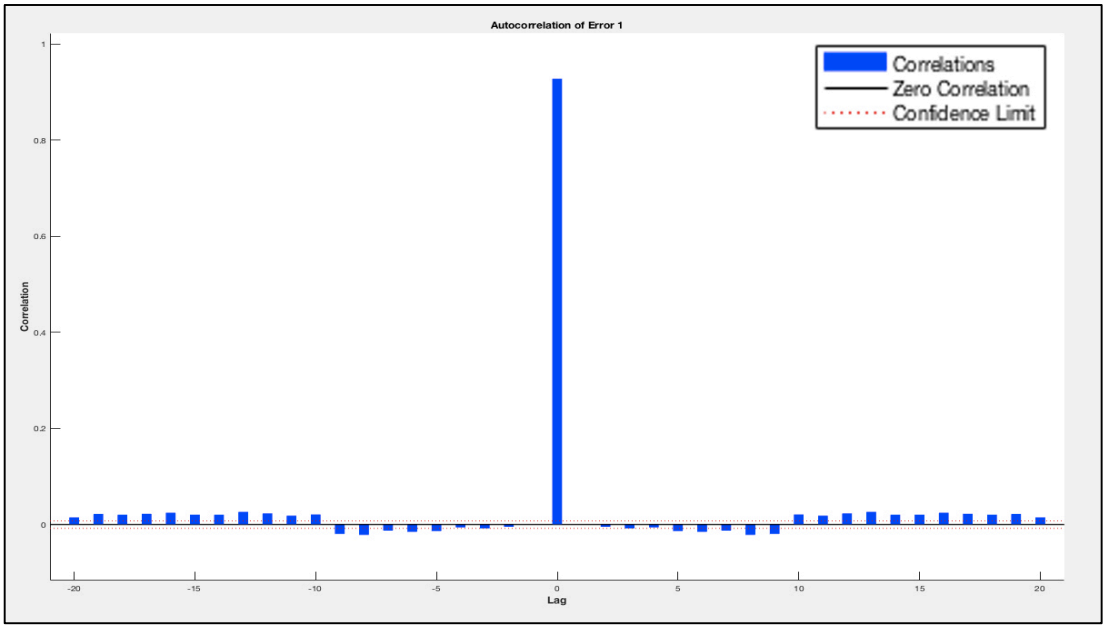


Figure 91. Autocorrelation plot for the error (with very low values, with Lag ≠ 0).

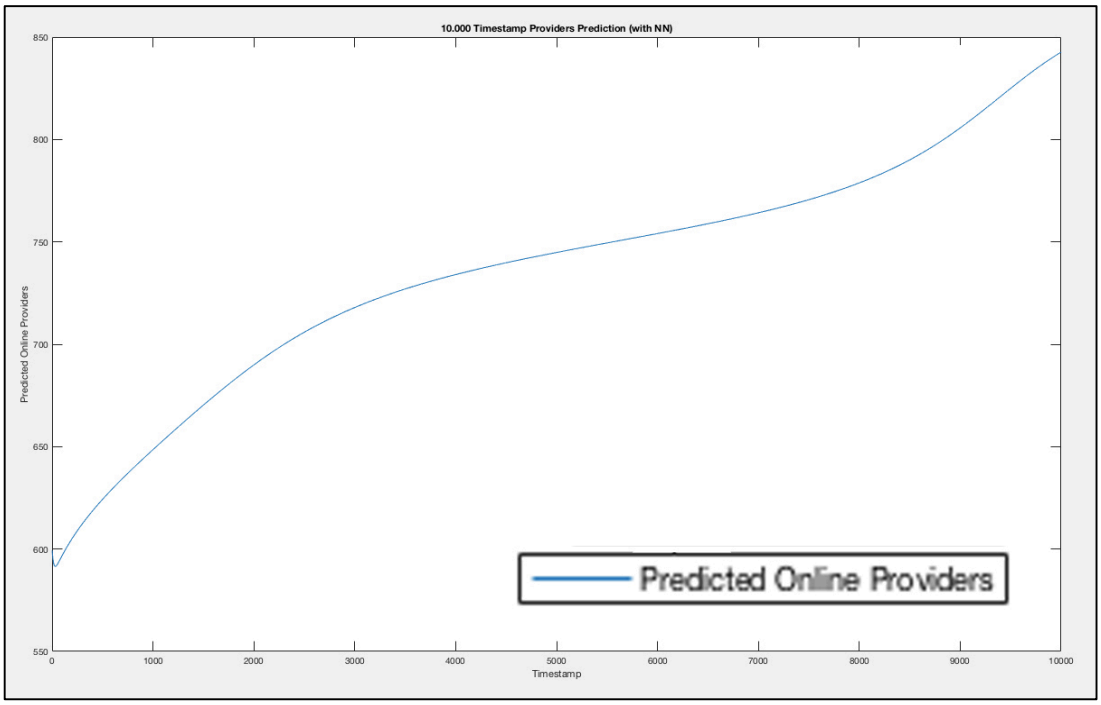


Figure 92. Online providers' prediction for the next 500 time units. Note that according to *Prediction Theory* and common sense, the farther we predict the more the prediction will be unlikely to be effective and precise.

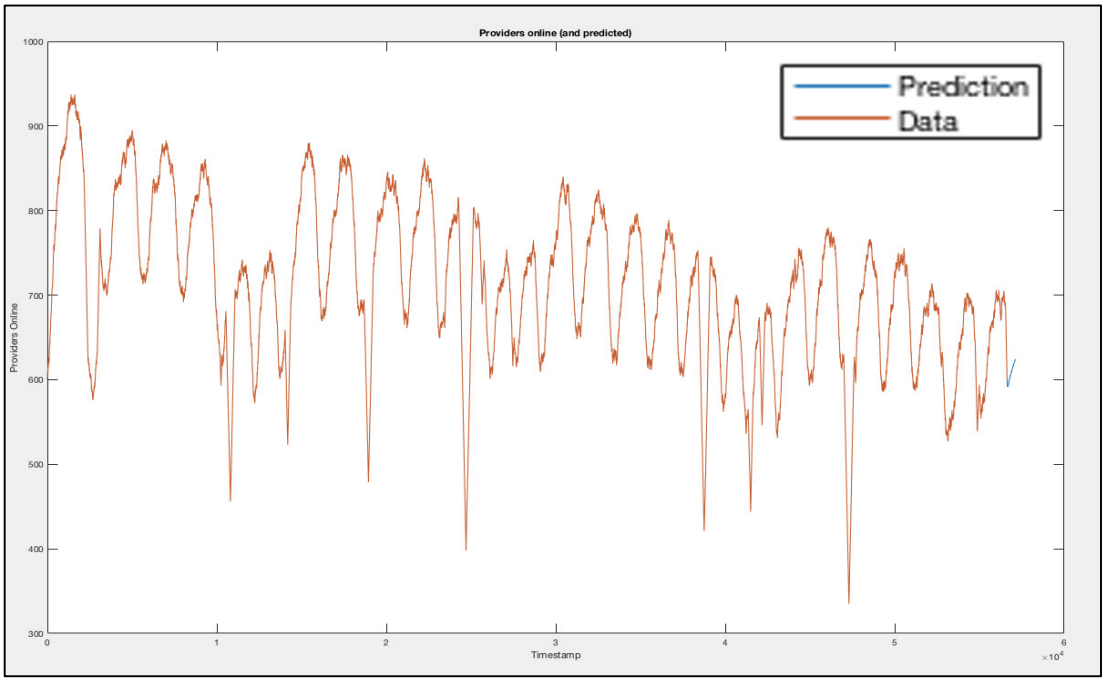


Figure 93. Online providers' real data (in orange) and predicted data for the future 500 time units (in blue). We can notice a growing behavior for the predictions.

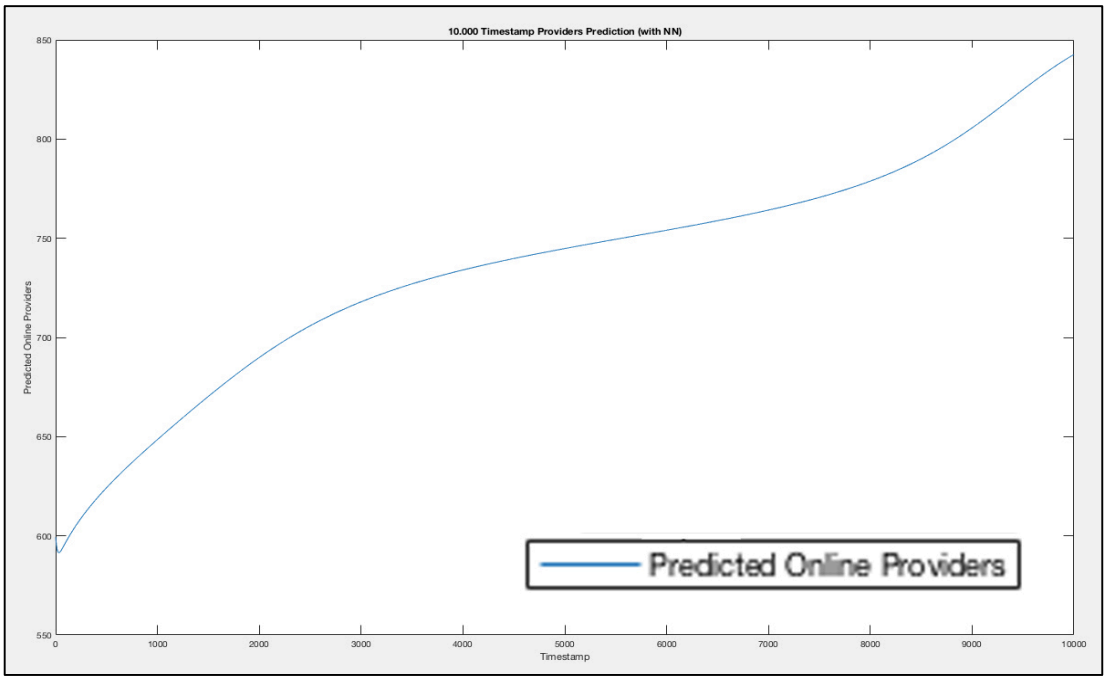


Figure 94. Online providers' prediction for the next 10.000 time units. Note that according to *Prediction Theory* and common sense, the farther we predict the more the prediction will be unlikely to be effective and precise.

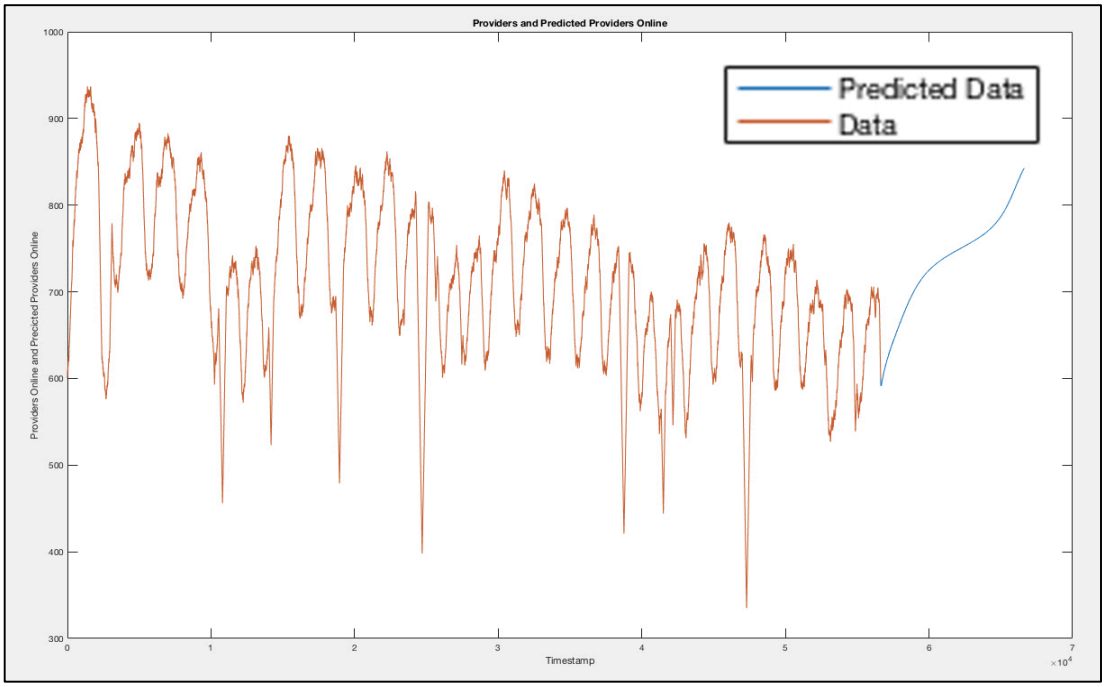


Figure 95. Online providers' real data (in orange) and predicted data for the future 10.000 time units (in blue). We can notice a growing behavior for the predictions.

Let us perform a brief discussion on the aforementioned **results** of this new Neural Network trained without initial and final data. In this case we do not have the effect of transients and their drawbacks on the *predictive application*, as previously seen and discussed in **[Chapter 5.4]**).

It is clear that the results are reasonable (and feasible) from a mathematical point of view in this case. We do not have the problem of negative values and this is a first advantage of the new proposed approach. Nevertheless, the predicted curve for new time stamps is still not very accurate yet, because it does not capture the effects of the continuous oscillations. The situations recall us something similar to what we have already seen in the *Curve Fitting* section, **[Chapter 5.3]**. In fact, the macroscopic growing behaviour of the curve is captured both from the *500 time units' prediction* and from the wider *10.000 time units' prediction*. But we are also interested in the details of the prediction with a finest level of granularity here. And it is clear that the microscopic behaviour, with the high-frequency of oscillations (and the overall undulatory nature), is not captured in the proper way.

As we have previously discussed in the *Curve Fitting* section, also here there is an *Underfitting* problem. The available data for training, test and validation are enough, but the chosen network has not enough expressive power for a deployable predictive application. Therefore, while the first encountered problem was related to the influence of transients in the prediction (with unreasonable negative values), now the second problem is related to an **underestimation** of the right network size and expressive power.

Let us present what we are going to do now and how we are going to solve the *Underfitting* problem. Once we have understood which are the main problems and that we are on the right road towards a *functional predictive model*, we will add more expressive power to the network (with delays and hidden neurons, for example) in order to observe if it is possible to capture the oscillations (performing the predictions more accurately, as desired). Clearly, the parallel challenge would be limit at possible the influence of *Overfitting*. We are not interested in a predictor who performs perfectly on past data and has low precision on new predictions, of course. Therefore, the main purpose of next chapter will be to search for the right network size and see what kind of performances it is able to show.

5.6 Neural Networks and providers (without transient): procedure, results and considerations with more complex Neural Networks

The previous results encouraged us to persist in the analysis of the dataset without transient. It is clear that the obtained prediction now has a meaning, but it still does not capture the oscillations of the providers' number, as it happens in the real world. We have a sort of *Underfitting* problem: the network on average predicts well the (possible) future values, but we want to extend its efficiency with more modelling power. The idea here is to add power to the network with more hidden neurons and delay units, in order to estimate in a better way new values and providers' fluctuations. In order to make these new analyses, we will use two more complex neural network for the training phase, called respectively **NN2** and **NN3**: the first one with 15 delay units and 50 hidden neurons, the second one with 40 delay units and 60 hidden neurons (see **[Figure 96]**).

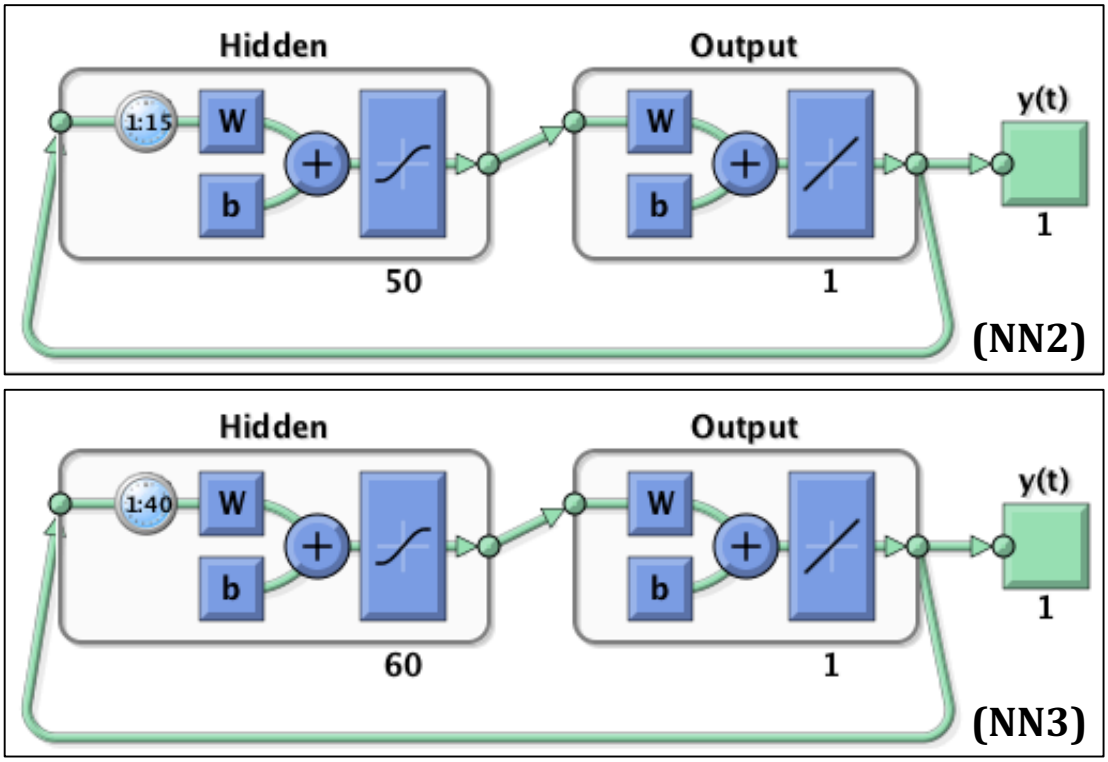


Figure 96. Closed Loop representation of *NN2* and *NN3*, used in order to predict the future 10.000 values of the time series (online providers).

We will not report all the steps of the training phase in order to avoid repetitions, but only the interesting results. In fact, they capture better and even better the oscillations of the functions (also with a little bit of *Overfitting* in the second case).

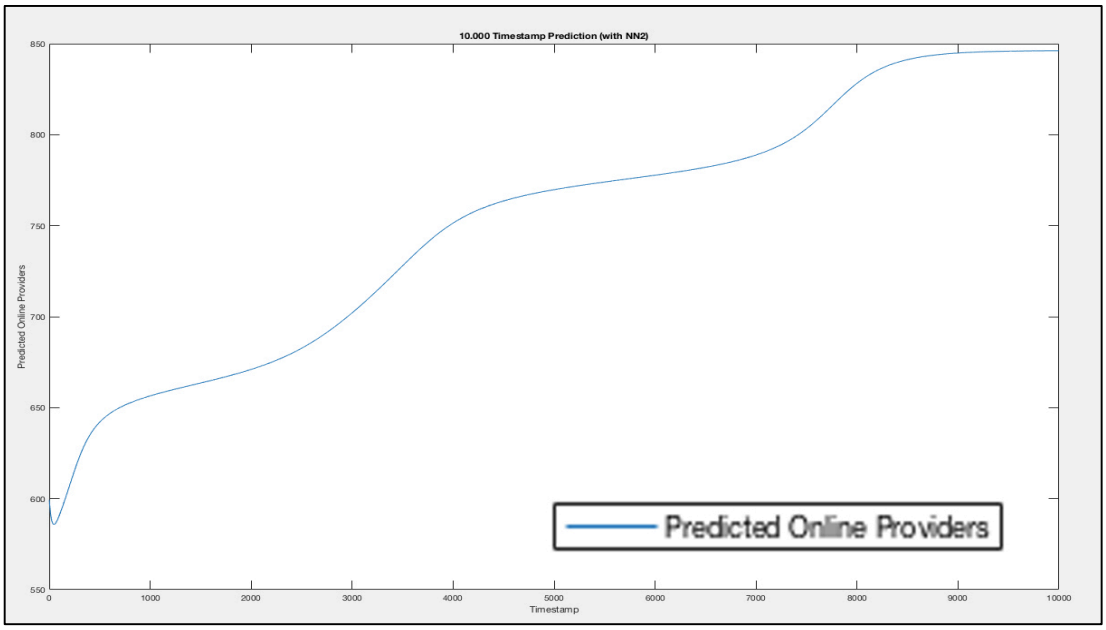


Figure 97. Online providers' prediction for the next 10.000 time units with *NN2*.

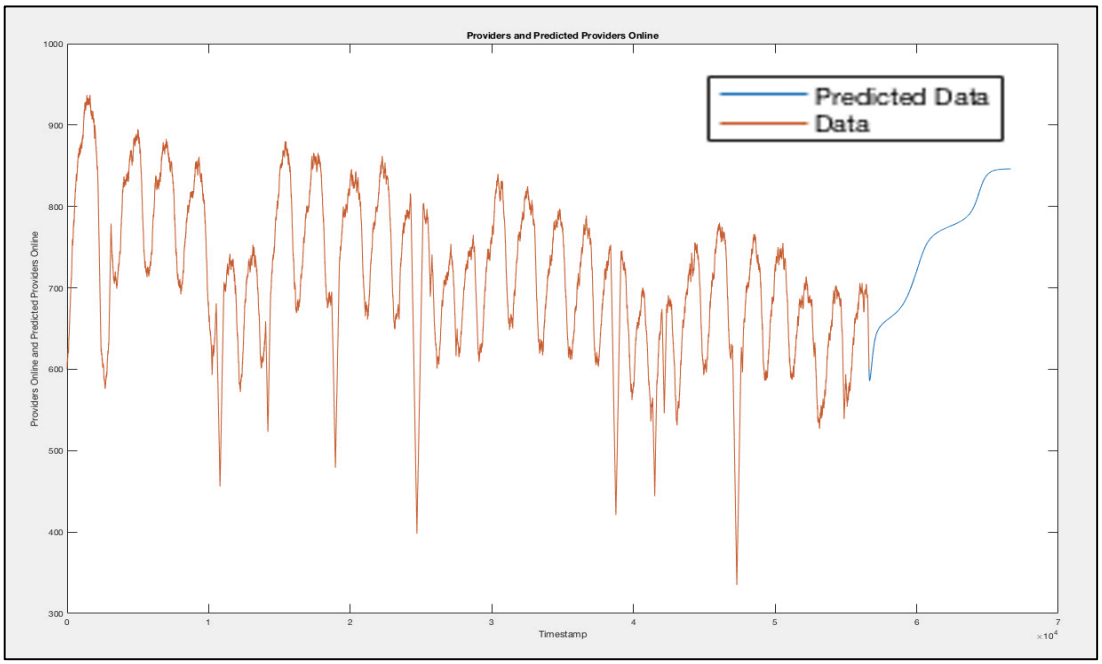


Figure 98. Online providers' real data (in orange) and predicted data for the future 10.000 time units (in blue) with *NN2*.

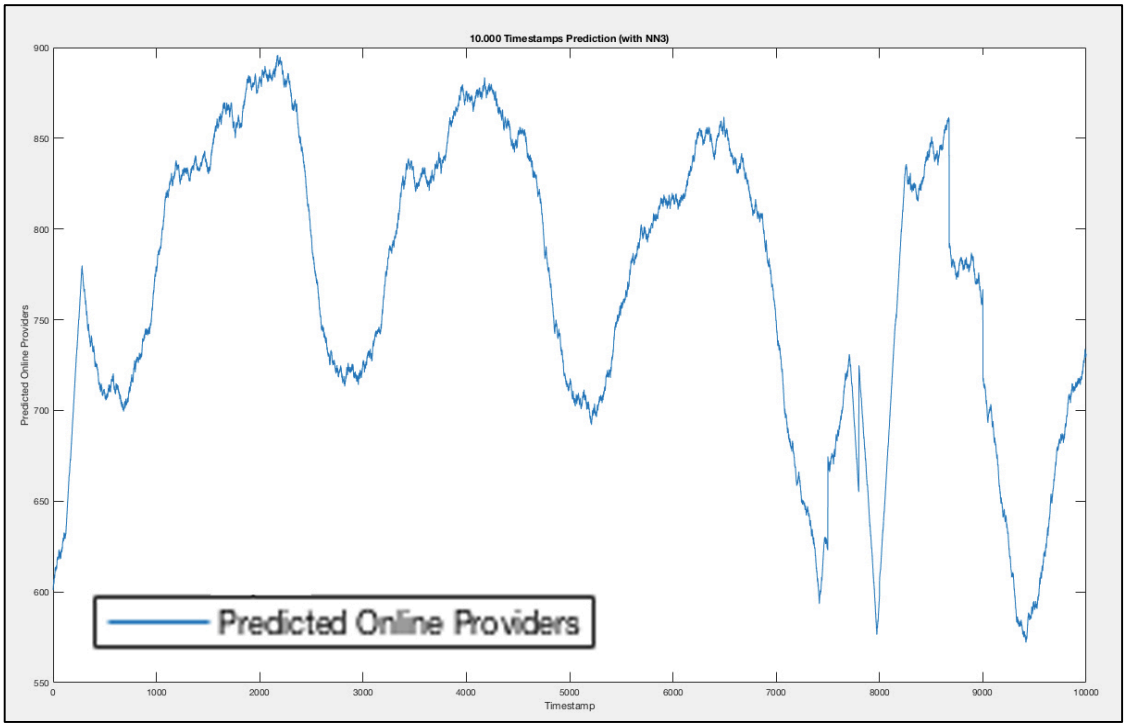


Figure 99. Online providers' prediction for the next 10.000 time units with *NN3*.

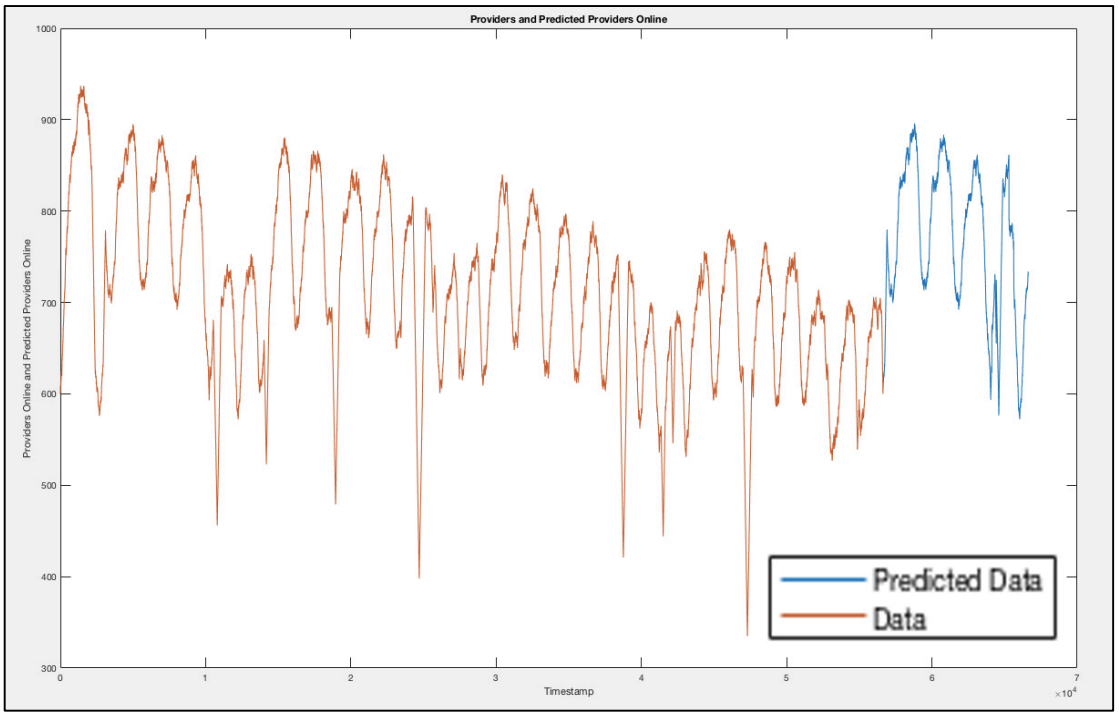


Figure 100. Online providers' real data (in orange) and predicted data for the future 10.000 time units (in blue) with *NN3*. This is clearly the most realistic prediction.

Let us now consider and discuss the obtained results with these new networks. First of all, it is evident that we have improved the past results of **[Chapter 5.5]** both with **NN2** and with **NN3**. Hence, the first statement that follows this approach is that we had a good intuition in the final considerations of the previous chapter. In fact, we performed the hypothesis of *Underfitting* as problem for the predicted values. Given that with these two more complex network we obtained better results, our reasoning is then demonstrated by the evidence.

Let us try to be more precise and detailed here, highlighting advantages and disadvantages of both the analyses (NN2 and NN3). If we compare the predictions of **NN2** with the ones performed by the network in **[Chapter 5.5]** we can see that there is a clear improvement in the approximations of the undulatory nature of the providers' curve. As a drawback, the obtained previsions are still too much optimistic, because the simultaneous crashes are not modelled as happens with the real world data. Here we have just some points with a low value for the *first derivative* of the prediction curve instead of local minima, with an overall *monotonically increasing* behaviour. This is still a high level representation of the real world phenomenon; we want to be more precise in prediction, with a finest level of detail. In other words, the performances of NN2 are still not satisfactory for an efficient predictive application.

Recalling the lesson learned in the previous chapter, we decided to add even more expressive power to the network (as anticipated at the beginning of this chapter). Also in this case with **NN3** we obtained an overall improvement in the predictions. In fact, we have a network with the capability to predict *high-frequency fluctuations* in the providers' number. The behaviour of the curve for predicted samples strengthens our original idea regarding the providers' variation due to the **day-night alternation cycle**[22] (as discussed in **[Chapter 5.3]**). We can be satisfied of these results, because they finally reflect and capture the behaviour of the real world system (with its dataset) analysed in the previous sub-chapters.

---

[22] There are several studies performed with respect to this factor (especially for **energy efficiency**). The energy demand, as the providers' demand, varies by the time of day, the day of the week, the latitude of the location, the seasons and many other factors. Thankfully, its shape and amplitude are reasonably predictable (with a certain degree of uncertainty, as always). From a practical point of view, shutting down some servers (providers) in specific time intervals (when the users' demand is lower) may be an effective strategy in order to reduce the costs of the companies.

5.7 Additional considerations and problematics

We can ask now to the original questions about online providers' prediction and crash prediction (with providers' dependence). We have only to observe the predicted values, searching for *local minima* if we want to identify the plausible time units for future crashes: in practice we solved (as possible) two problems with the same instruments. But there is not only gold here, Neural Networks are a powerful instrument, but are not the definitive King Midas[23] of the fight against uncertainty.

The inherently *black-box* structure of the Neural Networks can give problems when we are dealing with the parameters of the network. As we have seen, a more complex results can give more accurate results in terms of prediction, but clearly request more time and memory during the training step and can lead to *Overfitting*. During the experimentation there was another problem related to parameter tuning. In fact, with a bigger number of *Hidden Neurons* and *Delays* the predictions started to degenerate with strange, out-of-sense, fluctuations (also in the range of negative numbers, which is absurd) or alternatively with horizontal asymptote towards a given providers' number.

After decades of research it has not been proposed a standard reference for the right numbers of neurons to be used in a given network yet (there are some guidelines, but sometimes they do not work in the desired way). An important lesson here is that we can predict well with the *right* neural network, but we have to perform multiple testing with different *network parameters* (such as delays, hidden layers, hidden neurons, multiple training with different algorithms, different dataset subdivisions and so on) in order to achieve such *right* neural network (with satisfying and reliable results).

The last networks, **NN2** and **NN3** can be satisfactory, because capture the behaviour of the original curve (in the case without transient, as hypothesis), but remain a prediction: it can always happen something in the real world which contradict the prediction (also reversing it). This is the main cause of the poor application of these methodologies in other fields, such as finance for market stocks prediction: it is not just math, but there are several influences from real

[23] **King Midas** (from the royal house of Phrygia) is popularly remembered in Greek mythology for his ability to instantly turn everything he touched into gold. This came to be called the '*Golden Touch*', or the '*Midas Touch*'.

world human related factors. In fact, the state of the art predictors for the financial applications are usually 'enriched' with machine learning/data mining techniques with the capability to parse information from real world newspapers/websites or posts submitted via social networks (*Twitter* and *Facebook* are just two famous examples). But also with these improvements their results in prediction are still unsatisfactory! Maybe in the future something smarter will give better results and someone will become rich with a similar idea, who knows.

As we have anticipated (and also the common sense suggests), the original *Prediction Theory* clearly state that the more in the future we try to predict, the more the predictions are unsatisfactory. For example, if we want to predict the value of a stochastic variable $y(t + k)$, given the previous values $y(0), y(1) \dots y(t)$, for $k \to \infty$, the best prediction achievable is just the medium value (y_{MED} or μ) of the involved variable (this is the field of *Asymptotic Predictive Analysis*) [*Kenneth D. West, 1996*].

It is like a vicious circle: we started from uncertainty in the future values of the providers (related to the *Human in the loop factor*, according to our **[Figure 7]** taxonomy), we developed a methodology in order to have (reliable) predictions and we conclude here with the awareness of *uncertainty in the predictions*, which is the best target that can sometimes be achieved (represented as the **1st ignorance level** in the taxonomy presented in **[Chapter 2]**). Human behaviour is strange and has often a strong variance. Different individuals every day take different decisions on similar aspects of their life (love, travels, job are just trivial examples). This happens because everyone has a distinct personal background, with different ways to interpret the emotions and the situations in the world. Psychologists say that we are more complex than we can imagine and complexity means also difficulty in prediction. It is no exaggeration to say that the human brain is an impressive organ. No other brain in the animal kingdom is capable of generating the kind of higher consciousness associated with human ingenuity, with our ability to make plans, calculations and write poetry. Yet the most complex structure in the known universe, as it is often described, is more mysterious than the least explored regions of the deepest ocean.

After all, the world is beautiful also for the originality in the nature of human beings. If we want to close this humanistic digression and be more technical, we have to speak about *unpredictable environment due to (non-deterministic) human behaviour* and *uncertainty in predictions*.

Chapter 6 – Conclusions and future research directions

We have performed something which is very close to a travel through the world of uncertain during this work, trying to explore and highlight it as possible with the torch of mathematics and artificial intelligence. Starting from initial considerations, definitions and taxonomies about uncertainty we have developed methodologies that aim to improve our knowledge acquisition process, trying to remove as possible both aleatory and epistemic uncertainty. Therefore, we have softly moved from *uncertainty identification* (in models) to *uncertainty management* (*handling*) through the aforementioned techniques. In order to maintain also a parallel practical approach (in addition to the theoretical one), we have followed this 'conceptual path' given a specific application domain as case study. After an initial *taxonomical presentation* of the different sources of uncertainty, we have performed *uncertainty identification*. In particular, we have treated the sources of uncertainty called *'Noise in sensing', 'Decentralization', 'Granularity in models'* and *'Future Parameters Value'*, according to the taxonomy presented in **[Figure 7]**.

The analysed case study is a very realistic example from the real world; similar study cases will be probably analysed for years and years, considering that a world without mobile devices seems to be absurd in our days. The tendency, in fact, is to go towards an even more mobile world, Internet of Things (*IoT*), smart cars and cities are just a blueprint regarding the new world that we are going to see in the future. Nevertheless, we have to admit that it is impossible to predict what is going to happen in the future in a stochastic way.

We tried to do our best with some of the known instruments available, obtaining also some interesting results. They are a good starting point, but there is always a lot of possible additional work (and extensions) that it is possible to consider. New methodologies are created, improved and confuted day after day. From a formal point of view, it will be possible to integrate the presented approaches with hybrid methodologies as *Neuro-Fuzzy Networks* [*Robert Fullér, 1995*] or with more recent applications of the neural networks within the fascinating field of *Deep Learning* [*Yann LeCun et al. 2015*]. Also *Game Theory*, that we have briefly introduced in **[Chapter 2.5]** can give its strong contribution in decision making

problems with probabilities and sources of uncertainty. New hybrid systems can improve the results putting together the advantages of their components, leveraging on the best features of each approach.

Coming back to our *case study,* introduced in **[Chapter 3.1]**, we have to perform a final consideration also for the computing infrastructure and the related sources of uncertainty. The computing infrastructures are continuously under evolution and self-adaptiveness is becoming day after day the chosen design solution, both for small and big enterprises. Note that in order to deploy a self-adaptive system able to handle the sources of uncertainty is needed a specific *training* (strongly application-dependent). Considering our specific case study, there are already offline reproduction features for some application with smartphones/tablets nowadays (for example the famous musical application *Spotify* allows the user to save offline playlists and songs). This can be helpful from the availability point of view, because for a big amount of usage time the connection to Internet and the requests to the online providers can be avoided with offline materials. But this is not the only possibility to underline: with the progress it is very likely that we are going towards the direction of new, more reliable, infrastructures (*Cloud computing, CC,* is just one trivial example). Note that progress means improvement, but not perfection. If there is a thing that we have understood here is that is impossible to remove all sources of uncertainty in these application (and in our life, in general).

We will see what is going to happen in the next years, with only a certain idea: uncertainty will be always with us, and we have to learn how to live together with it. In a fair, elegant and conscious way.

Appendix

This main goal of this *Appendix* is to give a guideline regarding the code and the datasets hosted on the following *GitHub* repository and publicly available:

GitHub Repository: https://github.com/francesco-marchesani/master_thesis

There are three main folders: *Fuzzy Matlab, NN Matlab* and *Datasets*.

- In this **Fuzzy Matlab folder** it is possible to find MATLAB code (*.fis* format) concerning:

- α *or Input Parameters Availability* (starting from aleatory/epistemic uncertainty sources) → (*alfa.fis*);
- β *Availability* (starting from *Model Structure* aleatory/epistemic uncertainty sources + α) → (*beta.fis*);
- χ *or Overall Availability* (starting from *Model Context* aleatory/epistemic uncertainty sources + β) → (*overall.fis*).

Fuzzy Matlab Repository: https://github.com/francesco-marchesani/master_thesis/tree/master/Fuzzy%20Matlab

- In the **NN Matlab folder** it is possible to find MATLAB code concerning:

- Neural network **training and prediction script** → (*nn_script.m*);

- Suitable **MATLAB workspace** for the script → (*matlab_workspace_nn.mat*) with the following variables:

  - **n** = number of predictions to make;
  - **pn** = vector of previous observations *y(t)*;
  - **predicted_data** = data predicted with the NN;

- **c** = vector of previous observations + predicted data.

- **Additional NN workspace** with the generated net
  - **[d=10, hn=10]** → (*matlab_workspace_nn_additional.mat*).

Important Note: predicted data are located into the variable **y2** (of the workspace) after each execution of the *nn_script*! See the code documentation for more information.

NN Matlab Repository: https://github.com/francesco-marchesani/master_thesis/tree/master/NN%20Matlab

- Last but not least, the folder (**Datasets**) containing the datasets (with some additional scripts), especially the ones used for the Fuzzy Logic section (*web-sites.avt*) and for the Neural Network section (*skype.evt*).

Datasets Repository: https://github.com/francesco-marchesani/master_thesis/tree/master/Datasets

Bibliography

[1] *Arendt, P., Apley, D., Chen, W., Lamb, D., and Gorsich, D. (2012). Improving identifiability in model calibration using multiple responses. Journal of Mechanical Design, Transactions of the ASME 134, 10.*

[2] *Armour, P. G. (2000). The five orders of ignorance. Commun. ACM 43, 10, 17–20.*

[3] *Avinash K. Dixit,Barry J. Nalebuff, Thinking Strategically: The Competitive Edge in Business, Politics, and Everyday Life, W. W. Norton & Company; Reissue edition, ISBN-10: 0393310353, (1993).*

[4] *Bakkaloglu, M., Wylie, J. J., Wang, C., and Ganger, G. R. (2002). On correlated failures in survivable storage systems. Tech. Rep. CMU-CS-02-129, Carnegie Mellon University.*

[5] *Bert de Vries, Jose C. Principe, A Theory for Neural Networks with Time Delays. Department of Electrical Engineering. University of Horida, CSE 447. Gainesville, FL 32611, (1990).*

[6] *Bolosky, W. J., Douceur, J. R., Ely, D., and Theimer, M. (2000). Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In Proc. SIGMETRICS.*

[7] *Brighten Godfrey, <pbg@illinois.edu>, Repository of Availability Traces (Beta version 0.2), August 15, (2010).*

[8] *Brightwell, C. Kenyon, H. Paugam-Moisy, Multilayer neural networks: one or two hidden layers?, (1996).*

[9] *Brun et al, Engineering Self Brun et al, Engineering Self-Adaptive Systems Adaptive Systems through Feedback Loops, LNCS 5525, (2009).*

[10] *D. Perez-Palacin, R. Mirandola. Dealing with uncertainties in the modeling of self-adaptive systems ACM Trans. Autonom. Adapt. Syst. 9, 4, Article 39 (March 2010), 38 pages (technical report).*

[11] *D. Perez-Palacin, R. Mirandola. Uncertainties in the Modeling of Self-adaptive Systems: a Taxonomy and an Example of Availability Evaluation (2014). In Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering. ICPE '14. ACM, New York, NY, USA, 3–14.*

[12] *De Lemos, R., Camara, J., Cerqueira, R., and Kaaniche, M. (2013). Dependability in self-adaptive systems: How to justify trust in the face of "unknown unknowns"? LADC Year 2013 WDAS Panel, Full Material Accessible at source: http://www2.dc.ufscar.br/delano/WDAS/slides/deLemospanelWDAS.pdf.*

[13] *Dennis Lindley, Understanding Uncertainty, (2006), Wiley-Interscience, 11th edition, ISBN-10: 0470043830.*

[14] *Didier Dubois and Herni Prade, Possibility Theory and its Applications: Where Do we Stand?, (2011).*

[15] *Esfahani, N. and Malek, S. (2013). Uncertainty in self-adaptive software systems. In Software Engineer- ing for Self-Adaptive Systems II. LNCS Series, vol. 7475. Springer Berlin Heidelberg, 214–238.*

[16] *Glenn Shafer, A Mathematical Theory of Evidence, Princeton University Press, (1976).*

[17] *Guha, S., Daswani, N., and Jain, R. (2006). An Experimental Study of the Skype Peer-to-Peer VoIP System. In Proc. of the International Workshop on Peer-to-Peer Systems (IPTPS '06). Santa Barbara, CA.*

[18] *IBM, An architectural blueprint for automatic computing, (2005).*

[19] *Ibrahim Özkan and I. Burhan Türksen, Uncertainty and Fuzzy Decisions, S. Banerjee et al. (eds.), Chaos Theory in Politics, Understanding 17 Complex Systems, DOI 10.1007/978-94-017-8691-1__2, Springer ScienceCBusiness Media Dordrecht (2014).*

[20] *Jacob Bronowski, The Origins of Knowledge and Imagination, (1979), Yale University Press; Revised ed. edition, ISBN-10: 0300024096.*

[21] *Kenneth D. West, Asymptotic inference about predictiv ability, Econometrica, Vol.64, No.5, 1067-1084, September (1996).*

[22] *Kephart and Chess, 'The Vision of Automatic Computing' (2003), IEE, Computer (Volume: 36, Issue: 1, Page(s): 41 - 50).*

[23] *Kevin K Dobbin and Richard M Simon, Optimally splitting cases for training and testing high dimensional classifiers, 4:31, BMC Medical Genomics (2011).*

[24] *Kurt Hornik, Maxwell Stinchcombe, Halbert White, Multilayer feedforward networks are universal approximators, In Neural Networks, Volume 2, Issue 5, (1989), Pages 359-366, ISSN 0893-6080.*

[25] *Leon C. Megginson, Petroleum Management, Volume 36, Number 1, Key to Competition is Management, Start Page 91, Quote Page 91, Petroleum Engineer Publishing Company, Dallas, Texas (1964).*

[26] *Lofti Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, IEEE Transactions on Systems, Man and Cybernetics 3(1), p. 28, (1973).*

[27] *Nageswara S. V. Rao, Fei He, Jun Zhuang, Chris Y. T. Ma, David K. Y. Yau, Cloud computing infrastructure robustness: A game theory approach (2012).*

[28] *Pang, J., Hendricks, J., Akella, A., Maggs, B., Prisco, R. D. and Seshan, S. (2004). Availability, usage, and deployment characteristics of the domain name system. In Proc. IMC.*

[29] *Revaz Grigolia, Algebraic analysis of Lukasiewicz-Tarski's n-valued logical systems, (2015).*

[30] *Richard Lippman , An Introduction to Computing with Neural Nets, IEE Acoustics, Speech and Signal Processing, Apr. (1987), pp 4-22.*

[31] *Robert Fullér, Neural Fuzzy Systems, ISBN 951-650-624-0, ISSN 0358-5654, (1995).*

[32] *Rumsfeld, D. (2012). Dod news briefing - secretary rumsfeld and gen. myers. Accesible at http://archive.defense.gov/Transcripts/Transcript.aspx? TranscriptID=2636.*

[33] *S.Sapna, Dr.A.Tamilarasi and M.Pravin Kumar,Backpropagation Learning Algorithm based on Levenberg Marquardt algorithm, Sundarapandian et al. (Eds): CoNeCo,WiMo, NLP, CRYPSIS, ICAIT, ICDIP, ITCSE, CS & IT 07,pp. 393–398, (2012).*

[34] *Sepp Hochreiter and Jürgen Schmidhuber, Long Short-Term Memory, Neural Computation, Volume 9, Issue 8, November 15, (1997) p.1735-1780.*

[35] *Frank Schilder Werner Vogt Stephan Schreiber Hinke M. Osinga, Fourier methods for quasi-periodic oscillations, EPSRC grant GR/R72020/01 (2005).*

[36] *Smithson, M. (1989). Ignorance and uncertainty: Emerging paradigms. New York: Springer.*

[37] *Stribling, J. Planetlab all pairs ping. http://infospect.planet-lab.org/pings. (2005).*

[38] *Tannert C. et al. EMBO reports 8, 10, 892–896, (2007).*

[39] *Von Neumann John, Morgenstern Oskar, Theory of Games and Economic Behavior, (1944), Princeton University Press.*

[40] *Walker, W., Harremoes, P., Romans, J., Van Der Sluus, J., Van Asselt, M., Jassen, P., and Krauss, M. (2003). Defining uncertainty. a conceptual basis for uncertainty management in model-based decision support. Integrated Assessment 4, 1, 5–17.*

[41] *Yann LeCun, Yoshua Bengio and Geoffrey Hinton, Deep Learning (Review), Nature, Vol 521, 28 May (2015).*

[42] *Zadeh L. (1965), Fuzzy sets, Information and Control 8 (3): 338–353.*

[43] *Zadeh, L. A. (1999). Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets Syst. 100, 9–34.*

[44] *Zimmeramann, J. and Cremers, A. B. (2011). Rainbow of Computer Science: Dedicated to Hermann Maurer on the Occasion of His 70th Birthday. Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter The Quest for Uncertainty, 270–283.*