**Final Report**

**Full Unit Project 2016  
Benedict Wilkins**

**Supervisor: Zhiyuan Lou**

**Table of Contents**

[**Abstract** 5](#_Toc478400595)

[**Chapter 1: Introduction** 6](#_Toc478400596)

[**1.1 Goals and Structure** 6](#_Toc478400597)

[**1.2 Personal Development** 7](#_Toc478400598)

[**Chapter 2: Smart Meters, Energy Demand Management and Motivation** 8](#_Toc478400599)

[**2.1 Energy Demand Management** 8](#_Toc478400600)

[**2.2 Techniques for Supply Management** 8](#_Toc478400601)

[**2.3 Demand Side Management and Smart Meters** 8](#_Toc478400602)

[**2.4 Altering Behaviours** 9](#_Toc478400603)

[**2.5 Motivation** 10](#_Toc478400604)

[**Chapter 3: Introduction to Multi-agent Systems** 11](#_Toc478400605)

[**3.1 Intelligent Agents** 11](#_Toc478400606)

[**3.2 Multi-agent Systems** 12](#_Toc478400607)

[**3.3 Distributed Agent Environment** 12](#_Toc478400608)

[**3.4 Multi-agent Systems in Context** 13](#_Toc478400609)

[**3.4.1 Classifying the agent environment** 13](#_Toc478400610)

[**3.5 Agent Architecture** 14](#_Toc478400611)

[**3.6 Perceive, Decide, Execute** 14](#_Toc478400612)

[**3.7 Environment Architecture (Simulation)** 15](#_Toc478400613)

[**3.7.1 House Environment** 15](#_Toc478400614)

[**3.7.2 Physics** 15](#_Toc478400615)

[**3.7.3 Global Environment and Physics** 15](#_Toc478400616)

[**Chapter 4: Software Engineering in the Java Based Multi-Agent System** 17](#_Toc478400617)

[**4.1 Design patterns** 17](#_Toc478400618)

[**4.1.1 State** 17](#_Toc478400619)

[**4.1.2 Observer/Observable** 17](#_Toc478400620)

[**4.1.3 Factory** 17](#_Toc478400621)

[**4.1.4 Singleton** 17](#_Toc478400622)

[**4.2 Check style** 17](#_Toc478400623)

[**4.3 Test Strategies and TDD** 18](#_Toc478400624)

[**4.4 UML** 18](#_Toc478400625)

[**4.5 Revision Control** 19](#_Toc478400626)

[**Chapter 5: Data Generation** 20](#_Toc478400627)

[**5.1 Introduction** 20](#_Toc478400628)

[**5.1.1 Data Collection in Context** 20](#_Toc478400629)

[**5.2 Data analysis** 21](#_Toc478400630)

[**5.2.1 Low-Carbon-London-Dataset** 21](#_Toc478400631)

[**5.2.2 First look at Seasonality** 21](#_Toc478400632)

[**5.3 Building a Sampling Model** 22](#_Toc478400633)

[**5.4 Combined Sampling Model** 24](#_Toc478400634)

[**5.5 Seasonal Sampling Model** 25](#_Toc478400635)

[**Chapter 6: Machine Learning** 26](#_Toc478400636)

[**6.1 Introduction** 26](#_Toc478400637)

[**6.2 ARMA** 27](#_Toc478400638)

[**6.2.1 AR Model** 27](#_Toc478400639)

[**6.2.2 MA Model** 28](#_Toc478400640)

[**6.2.3 ARMA Model** 28](#_Toc478400641)

[**6.3 Weka** 29](#_Toc478400642)

[**6.3.1 Initial Exploration of Weka** 30](#_Toc478400643)

[**6.4 Model Evaluation** 30](#_Toc478400644)

[**6.4.1 Overfitting** 30](#_Toc478400645)

[**6.4.2 Cross Validation** 31](#_Toc478400646)

[**6.4.3 Cross Validation for Time Series Data** 31](#_Toc478400647)

[**6.5 Time Series Models in Weka** 32](#_Toc478400648)

[**6.5.1 MultilayerPerceptron** 32](#_Toc478400649)

[**6.5.2 Support Vector Machines** 37](#_Toc478400650)

[**Chapter 7: Developing the Model Further** 42](#_Toc478400651)

[**7.1 Feedback loop** 42](#_Toc478400652)

[**7.1.1 Thresholding** 43](#_Toc478400653)

[**7.1.2 Modification** 43](#_Toc478400654)

[**7.1.3 Implementation of Models, Modifiers and Thresholds** 43](#_Toc478400655)

[**7.2 Missing Data** 45](#_Toc478400656)

[**Chapter 8: Simulation Scenarios** 47](#_Toc478400657)

[**8.1 Scenario: Basic** 47](#_Toc478400658)

[**8.2 Scenario: Stress** 47](#_Toc478400659)

[**8.2 Scenario: Modification** 48](#_Toc478400660)

[**8.2.1 Magnitude** 48](#_Toc478400661)

[**8.2.2 Conclusions Regarding Modification and Underlying Model Functions** 49](#_Toc478400662)

[**8.3 Scenario: Weka Multilayer Perceptron Forecasting** 49](#_Toc478400663)

[**8.3.1 Null Modifier** 49](#_Toc478400664)

[**8.3.2 Magnitude Modifier** 50](#_Toc478400665)

[**8.3.3 Conclusions Regarding forecasting** 50](#_Toc478400666)

[**Chapter 9: User Guide** 52](#_Toc478400667)

[**9.1 Dependencies** 52](#_Toc478400668)

[**9.2 Running** 53](#_Toc478400669)

[**9.2.1 Simulation** 53](#_Toc478400670)

[**9.2.2 Data Fitter** 53](#_Toc478400671)

[**9.2.3 Machine Learning** 53](#_Toc478400672)

[**9.2.4 R** 54](#_Toc478400673)

[**9.2.5 Python** 54](#_Toc478400674)

[**Chapter 10: Professional Issues** 55](#_Toc478400675)

[**10.1 Licensing** 55](#_Toc478400676)

[**10.1.1 Weka** 55](#_Toc478400677)

[**10.1.2 JFreeChart** 55](#_Toc478400678)

[**10.2 Usability** 55](#_Toc478400679)

[**10.3 Safety and Reliability** 56](#_Toc478400680)

[**Chapter 11: Evaluation and Review** 57](#_Toc478400681)

[**11.1 Term One Review** 57](#_Toc478400682)

[**11.2 Term Two Review** 57](#_Toc478400683)

[**11.3 Project Review** 57](#_Toc478400684)

[**11.3.1 Project goals** 57](#_Toc478400685)

[**11.3.2 Original Reports** 58](#_Toc478400686)

[**11.3.3 Original Proof of Concept Programs** 58](#_Toc478400687)

[**11.4 Project Diary** 59](#_Toc478400688)

[**11.4.1 Term One** 59](#_Toc478400689)

[**11.4.2 Term Two** 60](#_Toc478400690)

[**11.5 Future Work** 61](#_Toc478400691)

[**Bibliography** 62](#_Toc478400692)

[**Appendices** 65](#_Toc478400693)

[**Appendix 1. Project Plan MAS Structure** 65](#_Toc478400694)

# **Abstract**

The purpose of this project is to design and implement a framework that will lead to the automated management of energy distribution networks. To demonstrate the effectiveness of the framework, it will also attempt to provide a means of increasing the efficiency of such networks by utilising current software and hardware technologies - Smart Meters, software agents and artificial intelligence. This efficiency increase will ultimately rely on social engineering and the concept of energy demand management (to be discussed shortly) to work, the project assumes that it will and focus solely on solving technical challenges that arise in build the framework and system to support these ideas. As the project does not have access to a real energy distribution network, the system will be demonstrated by simulation. The simulation system has been developed in the form of an agent environment.

# **Chapter 1: Introduction**

## **1.1 Goals and Structure**

First we present the proposed solution for each of the goals mentioned in the abstract. To facilitate a demand side management strategy (see *Chapter 2*) we will provide forecasts of the future consumption in a smart grid (an energy network containing smart meters). These forecasts can then be used by an energy provider to pre-empt energy distribution requirements and attempt to change the future behaviour of consumers in the desired regions. A network of agents will be created running on the Smart Meter hardware to collect data and feed it over network to a forecasting algorithm, in the system this algorithm will reside in specialised learning agent(s) located on company servers. The simulation system will be built as an agent environment and will try to represent this real world setting as best as possible so that the agents developed may be deployed directly in a real world setting. The system is extendable by design, and the framework will be in place to apply it to any equivalent problem.

Many problems arise in developing such a system and as such the structure of the project is quite complex. An attempt will be made here to make this structure as transparent to the reader as possible by describing its contents. This paper is roughly divided up into four sections (with the second and third merging somewhat) in the following order:

* Introduction
* Background theory
* Design, implementation and testing
* Evaluation, review and future work

We currently reside in the first, the second will focus on introducing and discussing the concepts that are at the core of the project. These are Energy Demand Management, Agents, and Artificial Intelligence. They can be found in their respective chapters:

* *Chapter 2*, focuses on the problem the project is attempting to solve, the motivation for the project, its importance, as well as a brief introduction to the following chapters.
* *Chapter 3*, introduces software agents, the motivation for using them as part of the solution and how they will be utilised.
* *Chapter 6*, provides background theory on basic machine learning techniques, then moves on to more advanced algorithms such as artificial neural networks and support vector machines.

Note that each of the above chapters also contain information regarding the third section, the third section also includes:

* *Chapter 4*, gives a discussion of software engineering in context as well as some implementation and design details.
* *Chapter 5*, a detailed discussion of house the simulation system is to be built - specifically how energy consumers will be modelled.
* *Chapter 7,* explains how the system handles such problems as:
  + When and how to change a consumers behaviour,
  + how a forecaster deals with a dynamic model and
  + what to do if data is missing.
* *Chapter 8*, demonstrates the complete system by given a number of difference scenarios, they output and any interesting observations.

The final section contains some extra information about the management of the project, an evaluation, user guide and a small section about relevant professional issues, all of these can be found at the end of the project after *chapter 8*. The second chapter will begin after a brief statement about personal development - what I hope to achieve by undertaking this project.

## **1.2 Personal Development**

I hope that by the end of this project my understanding of Machine Learning in general when applied to an industrial problem will be deeper. If the original project goals are satisfied I believe I will be in a good position to move into a career focused in artificial intelligence, especially one that has some involvement in the energy industry (e.g. British Gas). Although I am already a strong java programmer I also hope that my ability will improve as I am introduced to complex machine learning libraries. A side from java I will be using mathematical programming languages such as R in data analysis and exploration algorithms. All of this will further enhance my position when I eventually move from academia to industry. A review of my progress in these areas (essentially a self-evaluation) will be given towards the end of the report.

# **Chapter 2: Smart Meters, Energy Demand Management and Motivation**

## **2.1 Energy Demand Management**

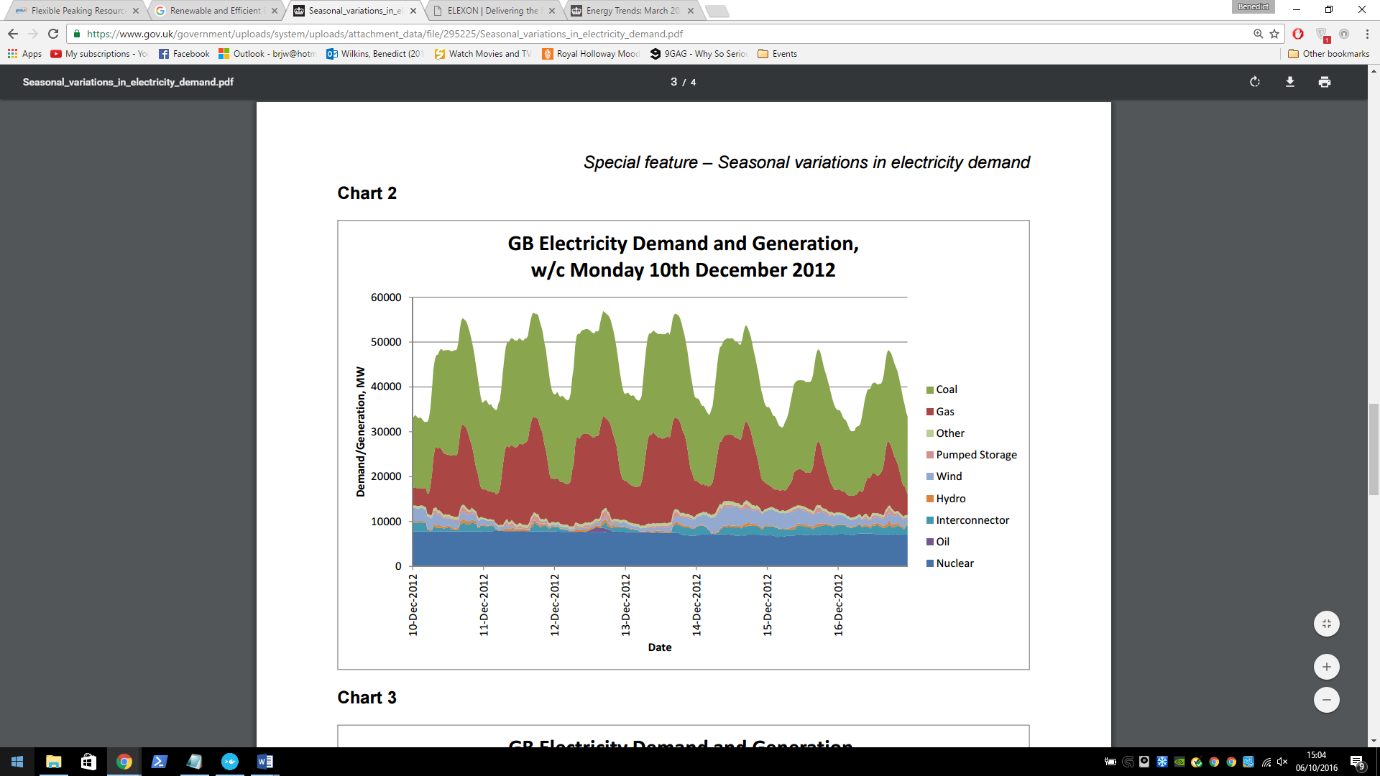
Energy Demand Management is in essence about managing the consumption/production of energy to ensure an effective energy network and to minimize cost and environmental damage. One of the main issues that EDM faces is how to efficiently supply energy to consumers. Peaks in energy demand arise when consumers – domestic or industrial, have synchronized habits. These lead to energy demand fluctuations (EDF), daily, weekly and seasonally [1]. These fluctuations must be dealt with by energy provides so not to result in damage to the energy network, blackouts and unpredictable service.

## **2.2 Techniques for Supply Management**

There are two general categories of power plant; base load and peaking. Base power plants are used to supply base load power – the minimum requirement of energy over a period of time. They are usually nuclear, coal or large hydro-electric plants [2]. These plants are only turned off for maintenance or upgrades and usually provide power to a large area. Peaking power plants are one way to deal with EDF. They produce a variable amount of power which is matched to the current demand. Peaking power plants are expensive, heavily dependent on the fossil fuel market (as they are usually gas/coal burning plants) and are only usually used as a last resort after employing the methods mentioned below. See *Figure 1.*

Other techniques include energy storage and energy purchasing. Energy storage involves storing a large amount of energy and releasing it back to the grid during peak times. One of the most common examples of large-scale energy store are hydroelectric dams/pumps. Water may be pumped during non-peak times or naturally build up behind a dam, later to be let through the turbines when it is required. A lot of energy is lost during this process and so it is not ideal. Energy purchasing is when an energy provider buys in energy from a separated grid. An example may be a British energy provider buying power from the French natation grid. Depending on the energy market this may be more desirable than using their own grid infrastructure.

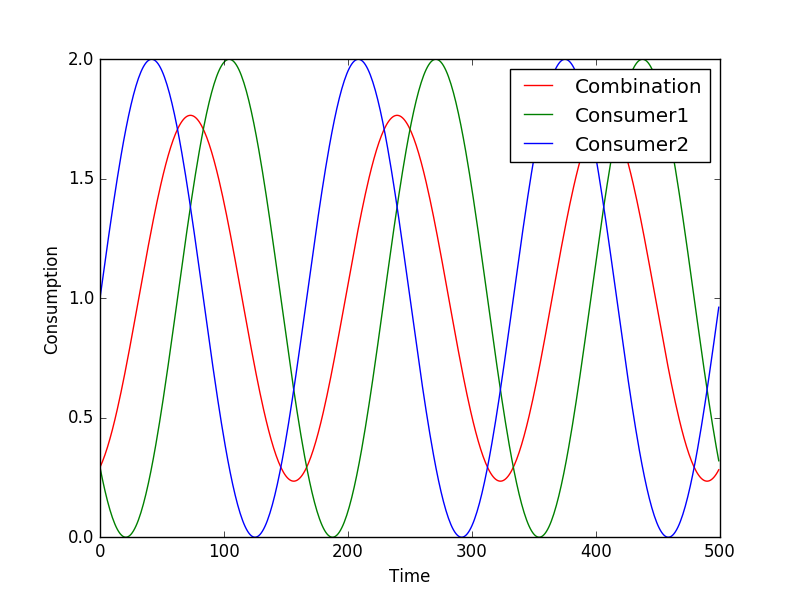
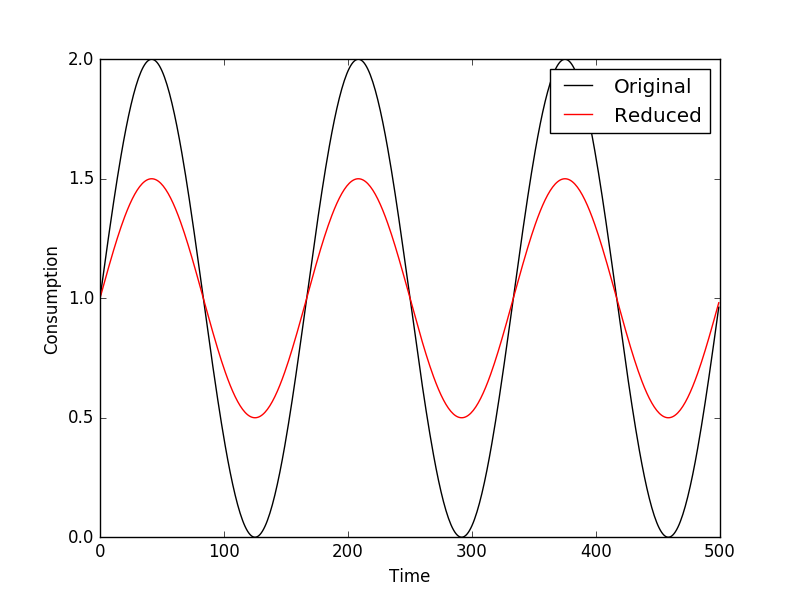
## **2.3 Demand Side Management and Smart Meters**

Moving to the consumer side of EDM. Another way of approaching the problem is to alter the habits of the consumers to reduce EDF. This will result in reduced cost for both energy providers and consumers. Demand response (DR) uses financial incentives to encourage consumers to alter their energy consumption habits. This method has been tested in the health sector; altering habits that relate to health by heavily taxing cigarettes and alcohol for example, has shown its effectiveness [3]. There have been various models that support DR in this projects context, see [4]. The National Grid (NG) in the UK has previously had meters to monitor electricity demand on a larger scale. Now however, with the advent of smart meters and their increasing popularity in the market, it is possible to collect house-hold specific data making it possible to implement DR.

***Figure 1.*** *Taken from Special feature – Seasonal variations in electricity demand. See* [1]*.  
The above shows the power demand/generation on the GB National Grid from 10 – 16 of December 2012 starting at midnight (00:00). It illustrates daily energy demand fluctuations and shows the use of peaking and base load power plants. Nuclear power plants are the latter. It is clear to see that gas and coal fire power plants are being used to deal with the EDF and so are the peaking power plants.*

## **2.4 Altering Behaviours**

So far we have mentioned altering the behaviour of the consumer, but have not addressed how they should change. It may seem obvious at first, we just want reduced energy consumption, that everyone should just stop using as much. This may be a result of the DR strategy put in place by a company that has deployed this system, however we should think about other ways in which a beneficial behaviour change can be made. One such way is to try to alter the times at which consumption occurs, if different consumers consume at different times the peaks will reduce, the network becomes easier to manage and less energy and money is wasted. For an illustration of the shifting compared to reducing see *figure 2.*



**Figure 2.** An illustration of the difference between shifting and reducing consumer behaviours. The red line in both shows the result of each transformation. The effect of both is essentially the same, while shifting the consumer behaviour will likely be easier in practice than reducing it, especially if we consider a business that requires a given amount of energy per day, perhaps they could shift their load hours to night for a cheaper rate. In this system these concepts are thought about in more detail in Chapter 7. Source code: SinGraph.py.

## **2.5 Motivation**

Financial motivation alone is hopefully enough to justify undertaking such a project from an economical point of view (of course more research would have to be done into the costs of deployment), if the reader is not so inclined by finance then the environmental benefits from a successful deployment should be convincing enough. As well as these incentives, as here is research it follows that a good opportunity has presented itself, namely to explore and test the capabilities of modern agent and AI technologies with an immediate potential for deployment.

Artificial intelligence and in particular machine learning is a heavily researched area in computing and is well suited in achieving this projects goal. Machine learning models require a few things to work but in essence; lots of data and a pattern. Both of those things seem to be present; data will be readily available from a large number of smart meters around the country and there is a pattern, looking a *figure 1* gives a good indication that there is at least a daily pattern. ANNs should be able to model this pattern effectively and provide good predictions as a result. ANNs have been used previous to forecast for arguably more complex system including the stock market. Different models are discussed in *chapter 6*.

Using a multi-agent framework will provide a flexible but robust structure for the system, it allows the system to be easily distributed which is key if the system is to scale well. The real smart meters can be represented as agents – they operate in some environment and their goal is to record consumption data from their environment. Using agents as a framework provides a good theoretical foundation for the project. It also allows ease of experimentation, agent behaviours can be added or replaced allowing role switching etc. In essence the use of agents allows for a fully automated system.

# **Chapter 3: Introduction to Multi-agent Systems**

## **3.1 Intelligent Agents**

There are a variety of definitions for what a software agent actually is. A good way to look at an agent is as a metaphor – the real world. We may consider a human as a real world agent. We exist in an environment - the universe, we have beliefs, desires and intentions which lead us to make decisions and ultimately perform actions which may or may not affect our environment. In essence this is what a software agent does but in a software context; it and its environment will be running as software on one or more machines. [5]

Continuing the metaphor, we as humans have a means to affect and observe our environment though use of our body. Parts of our body are used to perform actions in our environment – for example our hands can be used to pick up objects. In agent terms these parts are known as actuators or effectors. Other parts of our body are used for observation – for example our eyes, we look at the world and gather information about our environment. These parts are known as sensors. See *figure 3* for an illustration of this idea. It is interesting to see how far this metaphor can be taken and how well it represents software agents – this will be discuss further in the *Agent Architecture* section.

The environment is key aspect of agent based systems. An environment is just some space in which the agents in the system are housed. Agent environments have a number of properties [6], in the later section *MAS in context* some attempt will be made to classify the environment of the system to be developed with these properties in mind.

In a lot of cases agents are in environment where other agents are also situated and so the need for communication between agents arises. These kind of systems are known as multi-agent systems.

Agent

Environment

Action

Perception

***Figure 3.*** *Illustration of an agent acting on and observing an environment via is actuator and sensor.*

## **3.2 Multi-agent Systems**

Agents interact with others agents in a variety of different ways depending on context. These interactions can be classified as a result from the following two agent behaviours; cooperative and competitive. Cooperative agents work together to achieve a common goal while competitive agents work to achieve their individual goals. It is not unusual for cooperative agents to display competitive behaviour and vice versa [7]. If an agent is *clever* enough it is easy to see how a competitive agent may behave like it is a cooperative. It may temporarily work with other agents if it finds that this is a more effective way of achieving its own goal.

Communication between agents is an important aspect of a multi-agent systems (MAS) and it provides the basis for agent interaction. Cooperative agents must have a means of communication in order to complete their goal as a group. They may need to dynamically organise themselves or communicate information that may be useful to the other agents. Competitive agents may also need to communicate. An agent based e-commerce trading system is a good example of a need for competitive communication. Each agent in the system will represent a real world entity such as a person or corporation. Their goal is to negotiate the best deal for the entity they are representing. They will require some communication protocol in order to negotiation [8]. It is up to the agent designer to provide these communication protocols in a way that best suits the system.

## **3.3 Distributed Agent Environment**

The nature of the smart grid forces the system to utilise some distributed agent environment. Smart meters are physically separate entities each with an environment that they are responsible for – the household of the consumer. The smart meter agents (SMA) will need to be capable of communicating information to physically separate systems. This can be done using an IP protocol. Each SMA will be capable of communicating in this way with its manager agent. There are a few different possible configurations of management agents all with different associated positives and negatives and are to be discussed in this section.

The hierarchical structure of the MAS implies groupings of the agents. The SMAs already have a natural grouping, by geographical location or address which is already used by all household service providers. The manager agent’s job should be to forward data on behalf of its group as well as manage and control them by forwarding commands from higher up the hierarchy. (See *Appendix 1.* for details on the agent hierarchy).

To keep the system as de-centralised as possible it may be desirable to have the SMAs doing management themselves. They could delegate one of the other agents in their group to be a manager. This configuration may avoid system failures; if the delegated agent fails, a different SMA can be delegated as the manager. There is a potential privacy issue with sending data to a Smart Meter in a household, some measures would have to be in place to prevent data theft or alteration at the management agent location.

The alternative configuration is to have the management agent reside further up the hierarchy. It would be more complicated to deal with system failures as the system would have no alternative management agent. However there will be no privacy issues with sending the data.

## **3.4 Multi-agent Systems in Context**

There have been various other studies involving multi-agent systems and the smart grid [9] [10] [11]. W. *Multi-Agent Systems (MAS) controlled Smart Grid - A Review* in particular gives a good summary of relevant studies. Most are concerned with managing operations and control of the smart grid and some address problems such as the centralisation of power plants and service restoration.

This project is less concerned with the technology that will be used to implement the control/management of the smart grid after predictions have been made. However from the studies mentioned previously it seems likely that MAS will play a key role. This motivates the use of an extendable MAS in this project.

### **3.4.1 Classifying the agent environment**

(From afore mention standard environmental attributes (Russel and Norvig 1995))

To begin classifying the environment we must first define what exactly we mean in context. Each SMA is situated in its own ‘house’ environment in which it interacts (take readings). The house environment is however linked to the global environment; the environment containing all houses and any other agents in the system. This is the environment which we will try to classify as it contains all other environments.

**Inaccessible** – due to the nature of networking. Each SMA does not know what is happening in some other part of the environment. It could ask for more information from the other agents but again there is no guarantee of a reply. Similarly the manager agents (whether they are central or not) do not have direct access to the house environments.

**Nondeterministic** – again due to the nature of networking. It is possible that the system will fail at some point, an SMA may fail and so no data will be sent up the hierarchy. It should be noted that the receiving agents should be able to handle not receiving any data. We do not want the employed machine learning method to fail because of this.

**Nonepisodic** – This attribute is a little harder to pin down. It could be said that the SMAs will reside in an episodic environment, all they will do it send data regardless of success or failure in the previous episode. In the simulated system the agents will receive an energy usage value every logical[[1]](#footnote-1)\* half hour interval. However it may be the case that in the real system the agent will be continuously monitoring the energy usage. Whether we consider the physical monitoring device part of the agent is up for debate and should be carefully considered in a fully deployable version of the system. The non-SMA (the agents higher up the hierarchy) are situated in a nonepisodic environment. Although data may be sent every half hour from the SMAs perspectives their time may be out of sync (this should be minimised as it will have an impact on prediction). The non-SMAs must be capable of continually receiving packets from the SMAs.

**Dynamic** – The environment will change independently of the each agent. Network failures and other agents may affect the environment.

**Discrete** – SMAs may only take an energy reading and send it. Non-SMAs may only receive messages, send messages.

## **3.5 Agent Architecture**

The arrows represent the forwarding of events, results and actions.

Body

* The mind generates an action based on some reasoning, this is forwarded to the brain, which forwards it to the body. The body then generates an event from the action which is forwarded to the actuator. The actuator then forwards the event to the environment.
* The sensor receives a result from the environment, this is the agent’s perception. The sensor forwards it to the body and in the same fashion as above it reaches the mind as a result. The mind can then process the result as it sees it.

**Figure 4.** Illustration of the agent architecture. See [12] and [13].

In Java:Each part of the agent is represented as a class, each class is an observer observable pair (each class can communicate via observer/observable design pattern with its respective parts). The environmental also uses the observer/observable design pattern in order to send and receive results and events respectively. The body class contains all of the parts of the agent and so is essentially the complete agent. A body may also have an appearance which is defined as being; the external appearance of the body – what the agent looks like to other entities in the environment.

The framework for the agents is given by the GAWL (Generic Agent World Library) package. This framework also includes packages for, actions, events, perceptions, environment, physics and the observer/observable implementation. GAWL is essentially a revised version of the GOLEM framework see [12] and [13].[[2]](#footnote-2)\*

## **3.6 Perceive, Decide, Execute**

The architecture that is being used is an architecture used by the KGP model agency [14], where the agent control is based on a perceive-decide-act cycle, but implemented in Java rather than Prolog. In this project the agent control cycle will be explicitly referred to as the perceive-decide-execute cycle. The model is supported as a library within a revisited version of STARLITE [15] (STARLITE+), an agent platform that has been developed in the DICE lab at Royal Holloway ([http://dice.cs.rhul.ac.uk](http://dice.cs.rhul.ac.uk/)) to support the deployment of communicating software agents. Additionally see [16].

## **3.7 Environment Architecture (Simulation)**

### **3.7.1 House Environment**

A house environment is what will represent a real consumer house hold in the simulation. It contains a single SMA as well as its own physics and space (see *figure 5*). The space is responsible for generating readings. It uses an underlying data model which can be changed or replaced, this is implemented as a class: *MathematicalFunction* (See section 7.1.3). To see how an agent interacts with its environment see *figure 7.*

**Physics**

**Space**

**House  
Environment**

**Data Model**

**Figure 5.** Illustrates the house environment.

### **3.7.2 Physics**

The physics of an environment performs the actions that it has received only if they are possible in context. As an example, an SMA may request a consumption reading, this can be thought of as the action ‘take a reading’. This will be evaluated by the physics and performed. Performed in this case means querying the space and getting a reading. This is then returned to the SMAs respective sensor as a result. For more details on the how this process works see *Figure 7* (or the official documentation for GAWL)

### **3.7.3 Global Environment and Physics**

Each SMA in its respective house environment sends a messages to the global environment (See *Figure 6* below). The global environment houses the predictor and group agents as well as all house environments. House environments communicate the global environment directly – i.e. if an SMA has a message it will be sent to the global environment by the house environment containing the SMA. The group agent may represent a neighbourhood, region, country or other defined at the start of the simulation. The group agent will aggregate the data in some way and format it so that it may be useful to the predictor agent (Chapter 7).

In principle the may be any number of group agents representing different areas, the hierarchy of agents is arbitrary. As an example we may have multiple real neighbourhoods about which we want to make an aggregated prediction (a prediction grouped by houses in each neighbourhood). This can be done by simply having multiple Group (neighbourhood) agents with their respective House Environments. There is an obvious bottle neck at the Global environment/physics, this may be avoided by making a logical partition of the global environment e.g. for each group; a version of the global environment will be created to handle their communication specifically. In the real system the idea of Global environment will not hold as strong. The SMAs will forward their messages via IP communication – the machine who receives the message will essentially be the Global Environment. The IP communication has been implemented using java sockets. (See the *environment.communication.module* package). As IP communication is not important for the simulation, it will not be mentioned again for the duration of the report. However, SMAs have the capability to communicate using sockets if they use *IPCommunicationSensors* and *IPCommunicationActuators* in the *agent* package. The next chapter focuses on the implementation of all of the concepts mentioned in this chapter using software engineering techniques.

**Global Environment**

**House Environment**

**House Environment**

**House Environment**

**Global Physics**

**Figure 6**. Illustrates the Global Environment. The arrows show the transfer of events between environments, physics and agents. The each HouseEnvironment contains a single SMA, it is responsible for forwarding the SMAs messages in the form of events to its parent environment (the GlobalEnvironment). The group agent stands in for all intermediate agents (there may be arbitrarily many of them). There may also be many predictors. For more details on the agent hierarchy see Appendix 1.

# **Chapter 4: Software Engineering in the Java Based Multi-Agent System**

This chapter will discuss different software engineering concepts that are relevant in developing such a system, these include design patterns, test strategies, design, revision control and methodologies.

## **4.1 Design patterns**

### **4.1.1 State**

Allows an object to change its behaviour depending on its internal state by effectively changing its class.

The state design pattern has been used in many areas of my code. One example, the *SmartMeterAgentBody* class *brainHandler* attribute. The *brainHandler* attribute can be in one of two states – *NormalBrainHandler* or *IPBrainHandler*. This depends on the *Actuator* given to the agent at instantiation. One state allows the agent to handle *IPCommunicationActions* the other allows any non-IP related action e.g. *TakeReadingAction*.

### **4.1.2 Observer/Observable**

The observer design pattern is used for event parsing between objects without explicit method calls. In java the *update* and *notifyObervers* methods are used to do this.

A custom version of the java implementation is used in GAWL for event parsing between agent parts, agents and environments, and environments and their associated physics. For each of these there is one or more classes in the system that represent them; *NationalGridUniverse, SmartMeterAgentBody, HouseEnvironmentPhysics* etc.

### **4.1.3 Factory**

The factory design pattern provides an abstraction from object creation. No knowledge of the creation of the object is required, only the method in the factory for the desired object.

The *AgentFactory*, *EnvironmentFactory* and *HouseModelFactory* classes is are examples of the factory design pattern in the system. It abstracts from house model creation, only an error term is required for creation.

### **4.1.4 Singleton**

The singleton pattern is used to ensure that only one instance of a class can be created. This instance can be accessed in a static way, the class is responsible for the creation of this single instance.

The Factory classes above are examples of the singleton design pattern. Factories are always singletons as there should only ever be one instance at runtime.

## **4.2 Check style**

During the Java development process the google coding standards/check style was used to keep the formatting of code consistent and readable and to make it easier to follow proper java coding standards.

## **4.3 Test Strategies and TDD**

TDD is a software development strategy in which the requirements of the system are broken up into essentially atomic tests. The tests are written and then code is written so that tests succeed. All tests are re-run each time the system is tested. The system is built up in this way so that (hopefully) by the end everything will work. In java JUnit is used for TDD, JUnit and a TDD strategy was used when testing some important utility classes such as *TestArgumentUltilities*, *TestTimeDateTracker*, and *TestMathUtilities* all of which can be found in the *test* package.

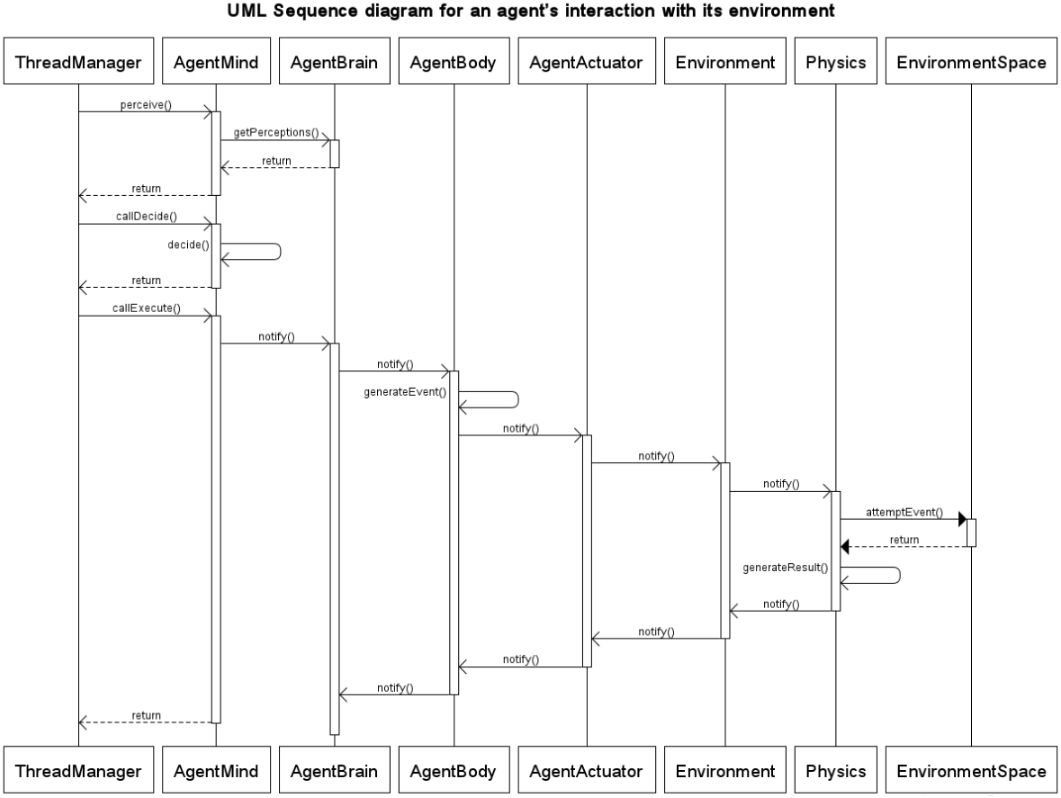
TDD was not a development strategy that would provide any advantage in the rest of the project. It becomes difficult to create unit tests for classes that interact heavily with one another and in a multi-threaded system it was nearly impossible. Instead, I used an integration based test strategy, testing the interacting between Observer/Observable pairs in a chain using simple print statements. For example to test event parsing between all parts of an agent – place a meaningful print statement in each *update* method and evaluate their order/correctness manually.

## **4.4 Software Engineering Methodologies**

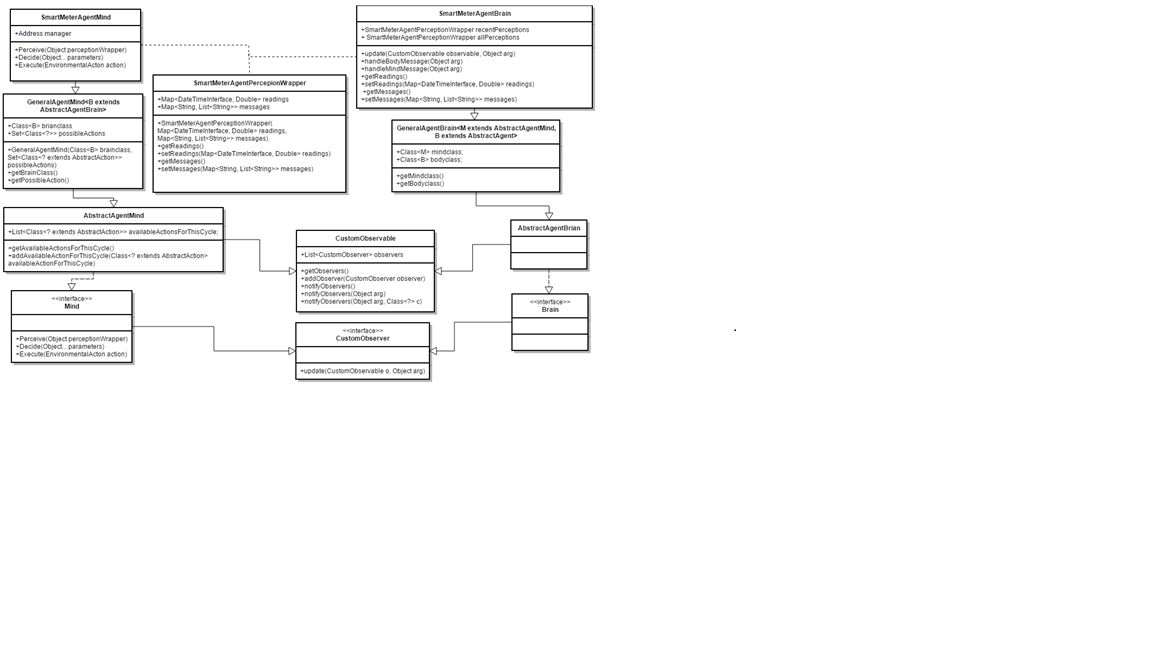
As this project did not have end user involvement, it was difficult to fully apply any agile methodology. However I did follow some of the agile software development principles, namely, “*working software over comprehensive documentation”* and “*responding to Change over following a plan”.* Regarding the former, there is complete documentation of the project (in terms of this report and full java documentation), however I found that, especially because I was working solo and regularly, working code was more important. That isn’t to say that the report and code documentation wasn’t developed in tandem, it just wasn’t my priority. Regarding the latter, after realising that the project was quick ambitious with the work load, I had to quickly adapt to try to satisfy the main project goals in a satisfactory manner. This involved the use of a library for the forecasting part of the project rather than to develop it myself. I believe this was the right decision as following the origin plan would have led to an unfinished project.

I tried to follow the *Spiral Model* as I thought it was most suitable, not quite agile but better than the *Waterfall* model. The project was developed iteratively using the four phases of the *spiral* *model*, *Identification, design, build, evaluation.* Each iteration of the spiral focused on a particular requirement of the project (roughly matching up to the chapters in this report, i.e. chapter 3, chapter 5 and chapter 6 with some chapters requiring more than a single iteration). The agent system was the beginning (architectural design and build), the others continued as system design/build. The most difficult part of the iteration was the *Evaluation* section. I had regular meetings with my supervisor to discuss the project of the project, these meetings as well as personal judgement were my only indicators of success or failure at the end of a cycle. The cost of the project could be evaluated as time rather than finance, this was usually the deciding factor on success of failure (i.e. how much time was taken over a certain requirement) and ultimately lead to the decision to alter some the initial requirements namely, implementation of forecasting. [[3]](#footnote-3)

## **4.5 UML**

UML diagrams are used to explain and model a program graphically. They illustrate the design and architecture of the program and can also be used to identify architectural strengths/weaknesses. Various UML diagrams were created for the project to illustrate the architecture of different areas. As the project is already quite large they are limited to only the most explanatory. ****

**Figure 7.** UML sequence diagram illustrating how an agents interacts with its environment. *All interactions are via observer/observable notification. All messages (except the Physics 🡪 EnvironmentSpace interaction) are asynchronous. The diagram is for a single agent only for simplicity, however in reality there may be multiple agents interacting with a physics and environment (for example in the NationalGridUniverse).* As a concrete example; an SMA may attempt to take a reading from its house environment in this way. The ThreadManager class is responsible for controlling all agents and calling their; perceive, decide, execute methods appropriately. Each agent runs in its own thread.

****

**Figure 8.** UML class diagram illustrating the SMA architecture.The SmartMeterAgentPerceptionWrapper is used to as way of parsing the most recent perceptions received by the agent from the brain to the mind. \*It should be noted that the original purpose of the brain was not storage of perceptions (it was to link the mind with the sensors and actuators) however in the current implementation of the GAWL library the body current has that job and so until this is altered the brain will be used as a storage for perceptions.

As well as the above, see *figure 29, figure 30.*

## **4.6 Revision Control**

Revision control systems such as Subversion and Git are used to store, manage, share code between developers working on a project. The case base keeps the entire history of the project (the versions) allowing more efficient code management.

I have used Git as my revision control system (GitHub) out of personal preference. I have tried to regularly commit to this repository but my work on the project was quite modular so they tended to be fortnightly (see *chapter 11*) and quite large. I have two branches: *agentsdemo* and *master*. *Agentsdemo* contains all the multi-agent system related code as well as the data generation code and testing. The master branch contains all reports, R and Python scripts as well as various other documents. The repository can be found at this address: <https://github.com/BenWilkins20/3rd-year-project.git>. Email – [ZAVC926@live.rhul.ac.uk](mailto:ZAVC926@live.rhul.ac.uk) to receive viewing permissions. The final version of the project is available as v1.0 under *tags* in the repository.

# **Chapter 5: Data Generation**

## **5.1 Introduction**

When generating data we want it to represent a real scenario as best as possible. The best way to do this is to acquire some real data; the energy usage of different households, and fit a representative model to it. We can then sample from the fitted model to generate realistic data. It is necessary to perform some analysis of the data (see below section Data Analysis) to find a good representative model. The analysis of the data gives some insight into desired properties of a prediction model e.g. linearity.

### **5.1.1 Data Collection in Context**

It is important to look at the type of data that the system will be using. According to the Smart Metering Implementation Program [17], *‘GSME shall be capable of recording Consumption in each thirty minute period’.* (GSME - Gas Smart Metering Equipment), the same applies for Electric Smart Metering Equipment (ESME). There is also a daily recording option however this will not be suitable for the predictions that this project is concerned with. In principle the system will be able to support any reasonable time scale, but the most useful will be on an hourly time scale (because of the relatively large hourly EDF see figure 1.) The implementation details of the smart meter or what type of meter is irrelevant as long as the data is useable. With this in mind, the system isn’t necessarily limited to house-holds. Data may be collected from businesses/industrial settings. However it may be useful to make the distinction between the two as the scale/fluctuation pattern of each may be quite different.

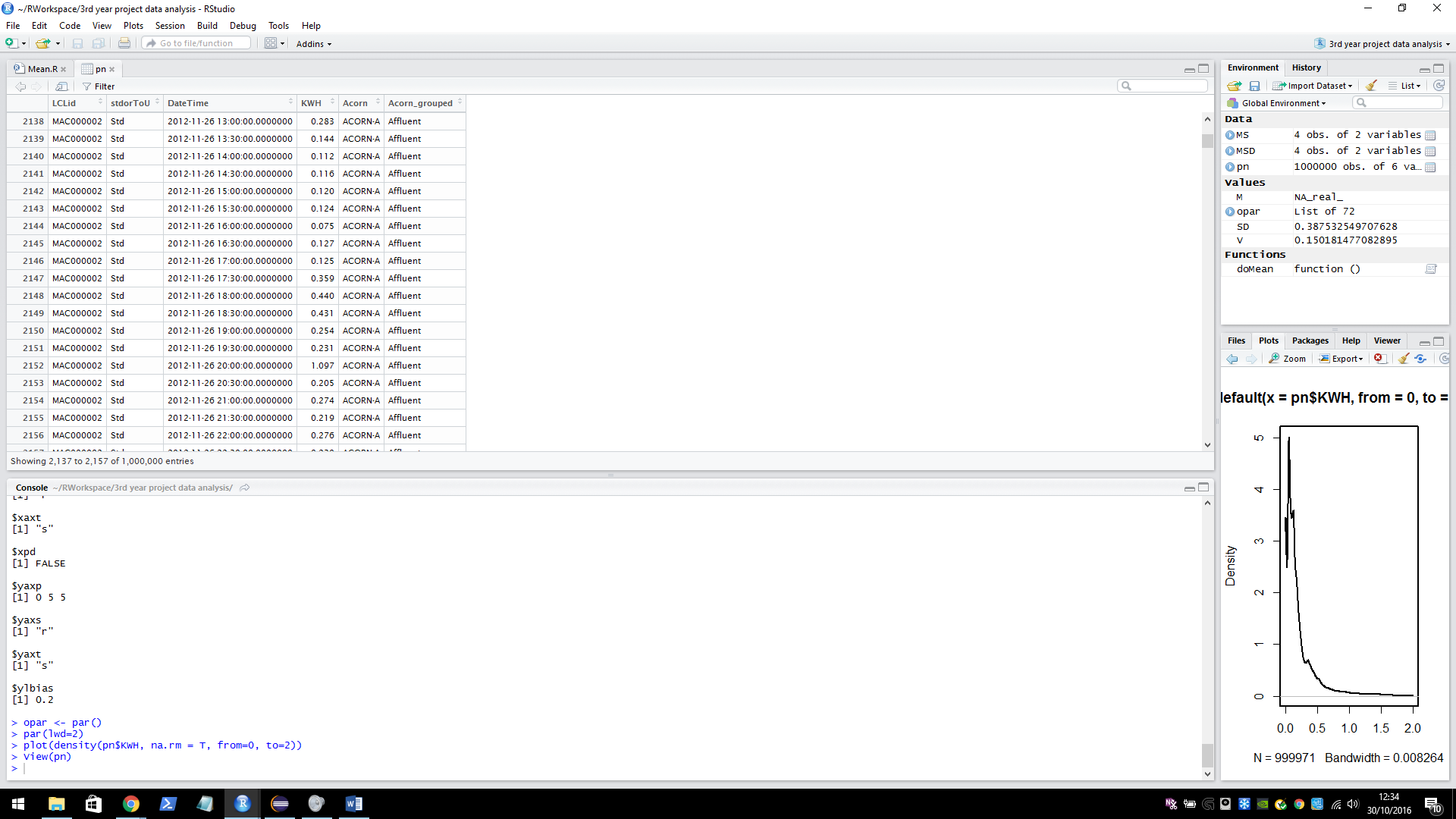
The below section contains analysis of the Low-Carbon-London dataset. This data set exactly the kind of data that the simulation will be using – half hourly data in different households for different financial situations.

## **5.2 Data analysis**

### **5.2.1 Low-Carbon-London-Dataset**

The data set includes KWH per half hour readings for a number of households. The data set is grouped into Affluent, Comfortable, Adversity and ACORN-U depending on the customer status.

The first section from this containing 1 million entries was used in the analysis. The complete dataset contains 167 million entries. The sample below shows 8 rows of the data set with the column names.

**LCLid**: the unique house identifier.

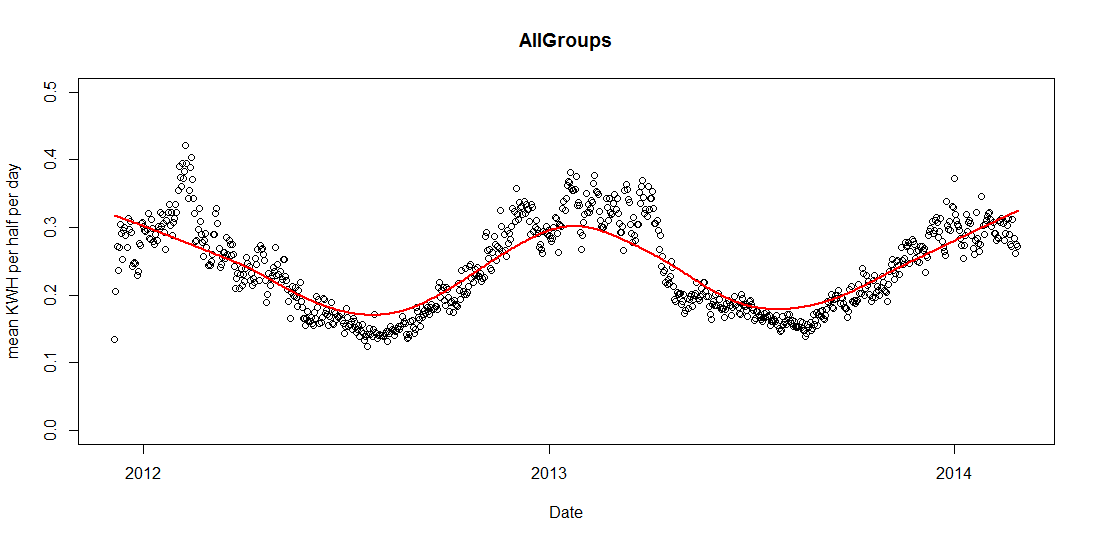
**stdorToU**: Tariff.

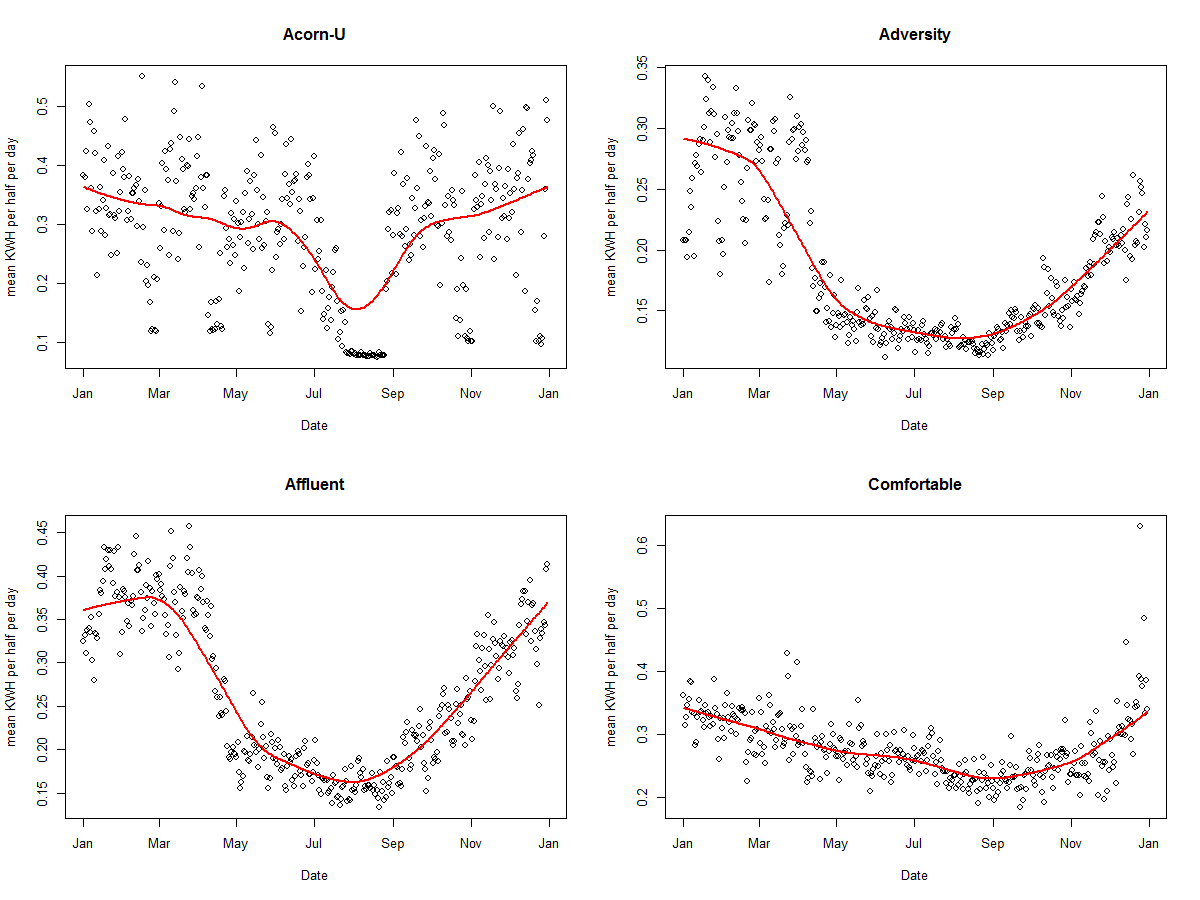
**KWH**: energy used KWH per half hour.  
**Acorn/Acorn grouped**: the grouping of the customer.

The data set can be found here <https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households>. Details on the ACORN groups can be found here <http://acorn.caci.co.uk/downloads/Acorn-User-guide.pdf>.

### **5.2.2 First look at Seasonality**

The plot below illustrates seasonality over the two years that data was collected. We can see that the average KWH usage is higher in the winter than in the summer. On the y axis: the mean KWH per half hour usage per day, on the x axis: the date (in days).

The seasonality of each group varies, however for the sampling model one seasonality function will be developed that represents all groups. This will be similar to the above graph (the average of all groups). The graphs below show the seasonality of each group in the 2013 year.



## **5.3 Building a Sampling Model**

When building the sampling model the equations for each group will take the form:

b­1\*normal(μ1, σ1) + b­2\*normal(μ2, σ2) + c

Manually fitting two normal curves to the mean KWH per half hour of the ACORN-U group. The blue line shows the fitted combined normal curve, the curve loops around the boundary, this is because hours are continuous – it is time series data. The red line shows a calculated mean values of the data group per half hour. The Y axis shows KWH per half hour, the X axis shows 48 half hourly time intervals which covers one day (24 hours) starting at 00:00:00 and ending at 11:30:00. (The graphs below were generated in the java class DataFitter using the JFreeChart graphing library).

|  |  |
| --- | --- |
|  | **Group: Acorn-U**  b­1 = 3.8  μ1 = 40.0  σ1 = 3.0  b­2 = 6.0  μ2 = 25.0  σ2 = 7.0  c = 0.1 |
|  | **Group: Adversity**  b­1 = 1.0  μ1 = 1.5  σ1 = 1.5  b­2 = 2.0  μ2 = 20.0  σ2 = 14.0  c = 0.12 |
|  | **Group: Affluent**  b­1 = 6.0  μ1 = 48.0  σ1 = 9.0  b­2 = 1.5  μ2 = 24.0  σ2 = 5.0  c = 0.15  discrepancy in normal curve \* |
|  | **Group: Comfortable**  b­1 = 5.0  μ1 = 40.0  σ1 = 5.0  b­2 = 4.0  μ2 = 22.0  σ2 = 6.5  c = 0.1 |

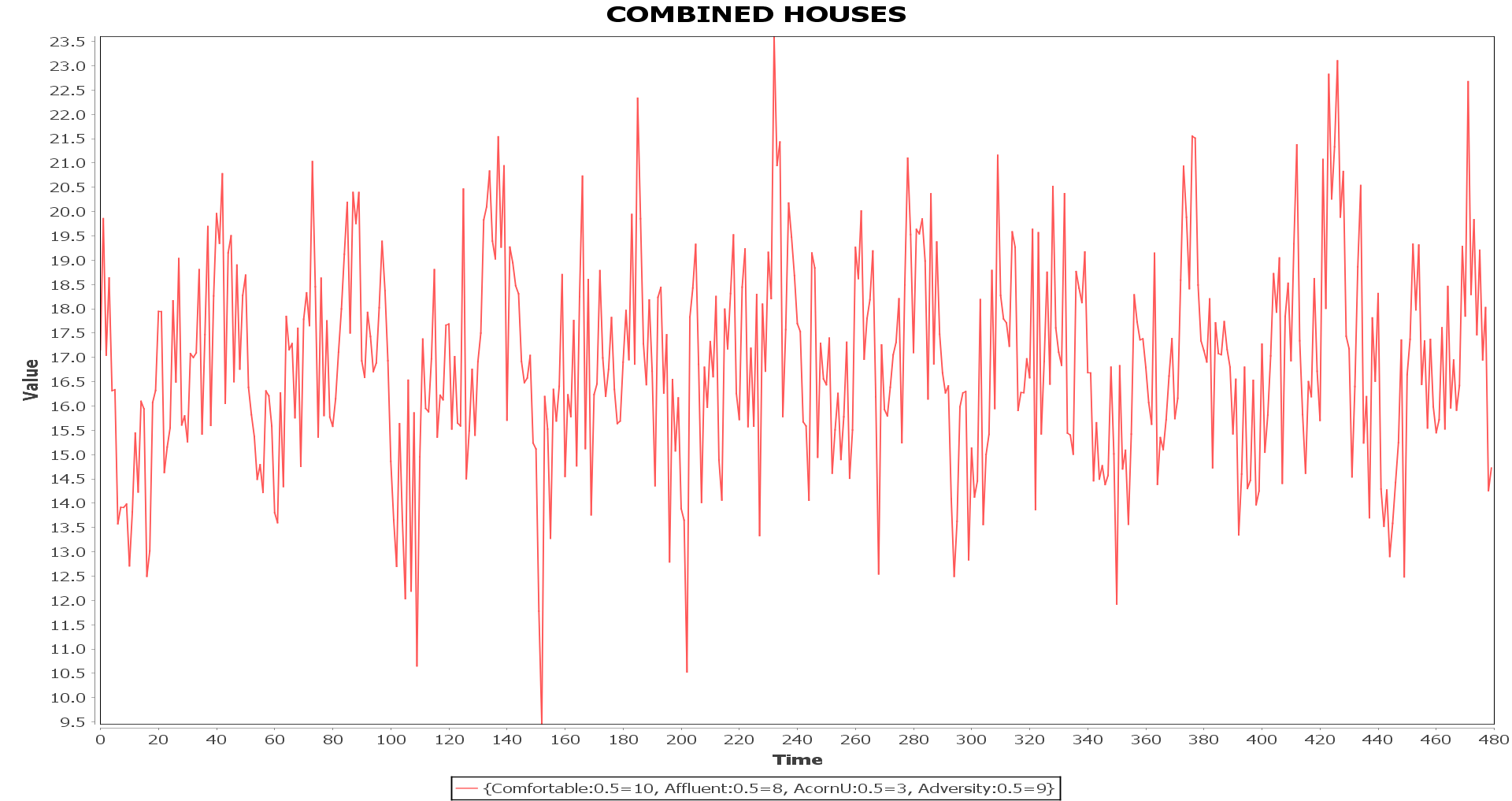
**\*** In the Affluent graph there is a small discrepancy in the normal line. This is because the equation wraps around the boundaries up to half of the range in each direction. It occurs when there is a significant different in the middle values (for each wrapped half range), in most cases the difference is negligible because the normal curve tends to 0, it has negligible effect when added. In this case however the stand deviation of the curves is sufficient enough so that the line is not close to 0 and so the addition is noticeable. It will have a very small effect on the data sampling at the centre point but as the curve is an approximation and some error term will be added anyway it is not something to be too concerned about.

After a pure model has been built a random error term (*E*) will be added to each house representative of the noise in the real data. This term will be assigned at the beginning of a simulation.

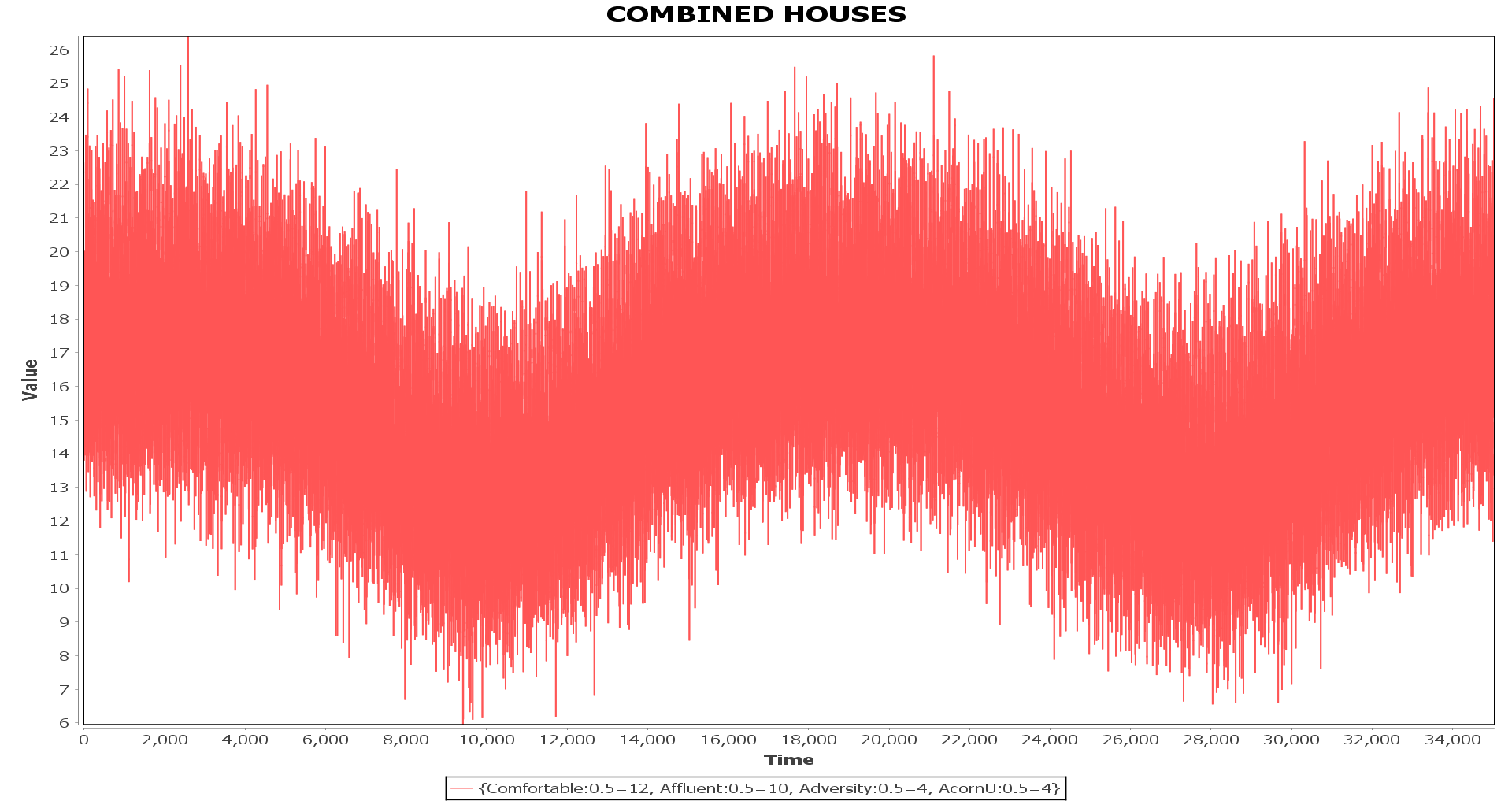
## **5.4 Combined Sampling Model**

The next step is to combine house models. The system has the capability to auto generate combinations of houses, this is representative of multiple *SMAs* each employing their own consumption model. The below graph shows an example of a random combination of houses combined using an additive combine (see the *Combinator* Interface and *AdditiveCombinator* class). In this case *E* given to each house model was 0 i.e. it is a pure model. The combination of houses used is as follows: 11 \* Adversity, 6 \* Affluent, 5 \* AcornU, 8 \* Comfortable for a total of 30 different houses (Information can be seen in graph legend, here value = energy consumption).

To demonstrate *E*, here is another example of a random house combination where *E* = 0.5.



## **5.5 Seasonal Sampling Model**

Yearly seasonality is added to a model using the *SeasonalModifier* class. This model was built in a similar way to the house models (except using a single normal curve per year). Below shows data over a 2 year time frame, with *E* = 0.5 for some random set of houses. The series starts and ends during the winter season.

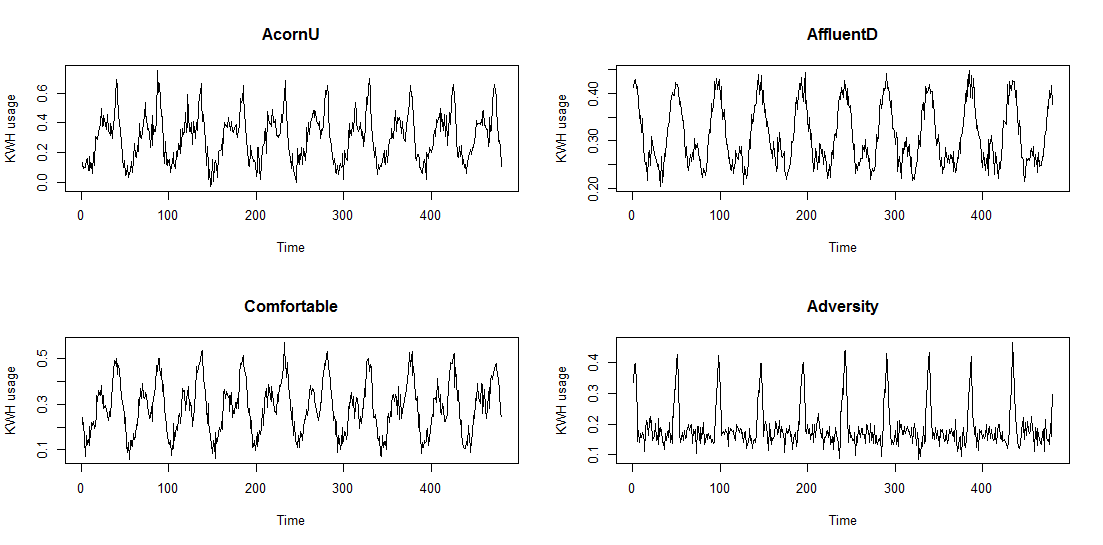
**Figure 9.** Illustration of yearly seasonality.

# **Chapter 6: Machine Learning**

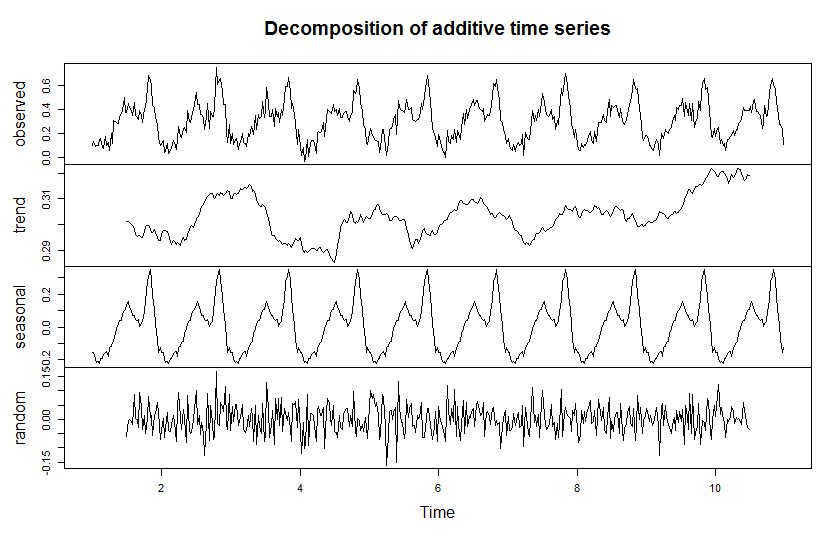
## **6.1 Introduction**

Here we will finally discuss how the forecasting agent should be implemented, or more specifically, what algorithm should be used by the forecasting agent. The initial thought was to use artificial neural networks (ANN), they their advantages but also their disadvantages. They are essentially a black box; the interpretability of their results is very bad. It will be advantageous in a business oriented environment to have at least some interpretability. There is an opportunity to test the use different machine learning techniques with the success of the data generation module. If a prediction model can be found that works as well or better than an ANN model then it should be seriously considered. Taking this opportunity, the next section will begin to test some alternative machine learning models.

It should be noted that this testing is completely separate from the multi-agent system as generating data through a simulation is unnecessary. A dedicated data generator will provide enough data to perform the tests (See *ExperimentDataGenerator* in the *demo* package). Different combinations of house models will be used to see if they will affect the prediction performance of the alternative methods. The testing will be done using R and using time series packages (R stats library).

Here we present a selection of generated data that will be used shortly for analysis with ARMA.

**Figure 10.** Illustrates data generated by the four different house models.

  
The above is a sample of generated data over 10 days. It is nice to see that the seasonal (daily) looks somewhat similar to *Figure 1*. The next step it to forecast for future data. Note that the data can be described by an additive time series model (which is required for the R function that did the decomposition (uses the ‘TTR’ R library)). At this moment the generated data does not include any yearly seasonality. Currently the series is stationary – the mean, variance and covariance (of the i­th and i­th + n) term do not change with time (this may change when yearly seasonality is added). We can apply a basic ARMA (or AR/MA) model for forecasting without having to employ any methods to make the series stationary (differencing, de-trending etc). The study of ARMA below will use the Acorn-U generated series as its example.

**Figure 11.** Illustration of decomposition of Acorn-U generated data.

## **6.2 ARMA**

ARMA is a combinations of the AR (Auto-Regressive) and MA (Moving average) models, each will be briefly described below.

## **6.2.1 AR Model**

An AR model is one where (value at current time) depends only on its own past. The equation an AR model of order is:

Where is the value at time, is the intercept, is some past value with some coefficient and is some error term. An AR model of order 0, , of order 1 AR(1) = and so on. Usually p < 5.

## **6.2.2 MA Model**

An MA model is one where depends only on the random terms given by a white noise process. A white noise process is defined as:

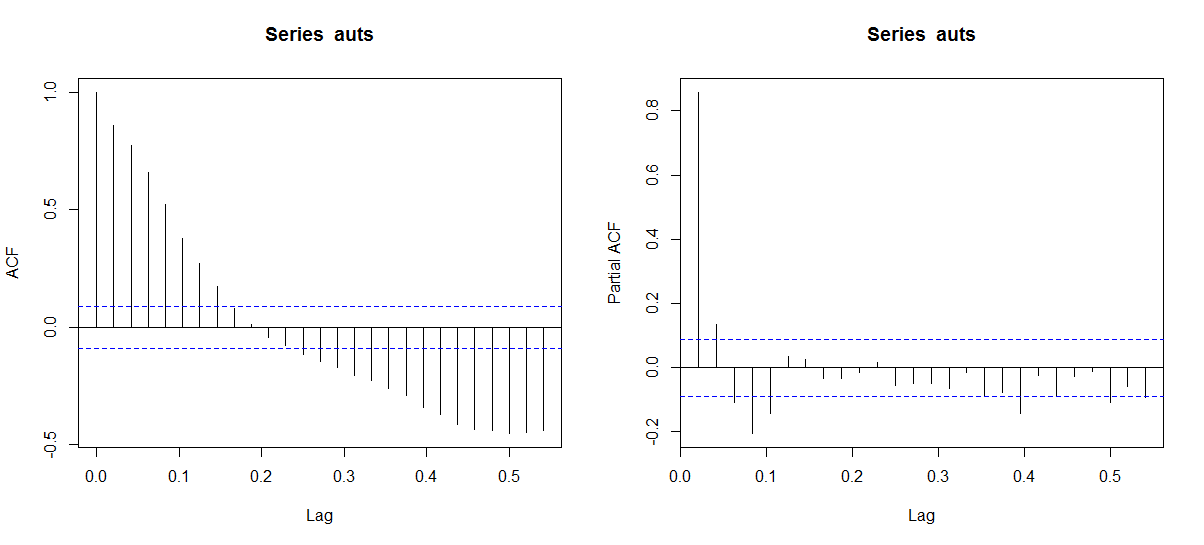
The equation for an MA model of order is:

Note that comes from, and so on.[[4]](#footnote-4)

## **6.2.3 ARMA Model**

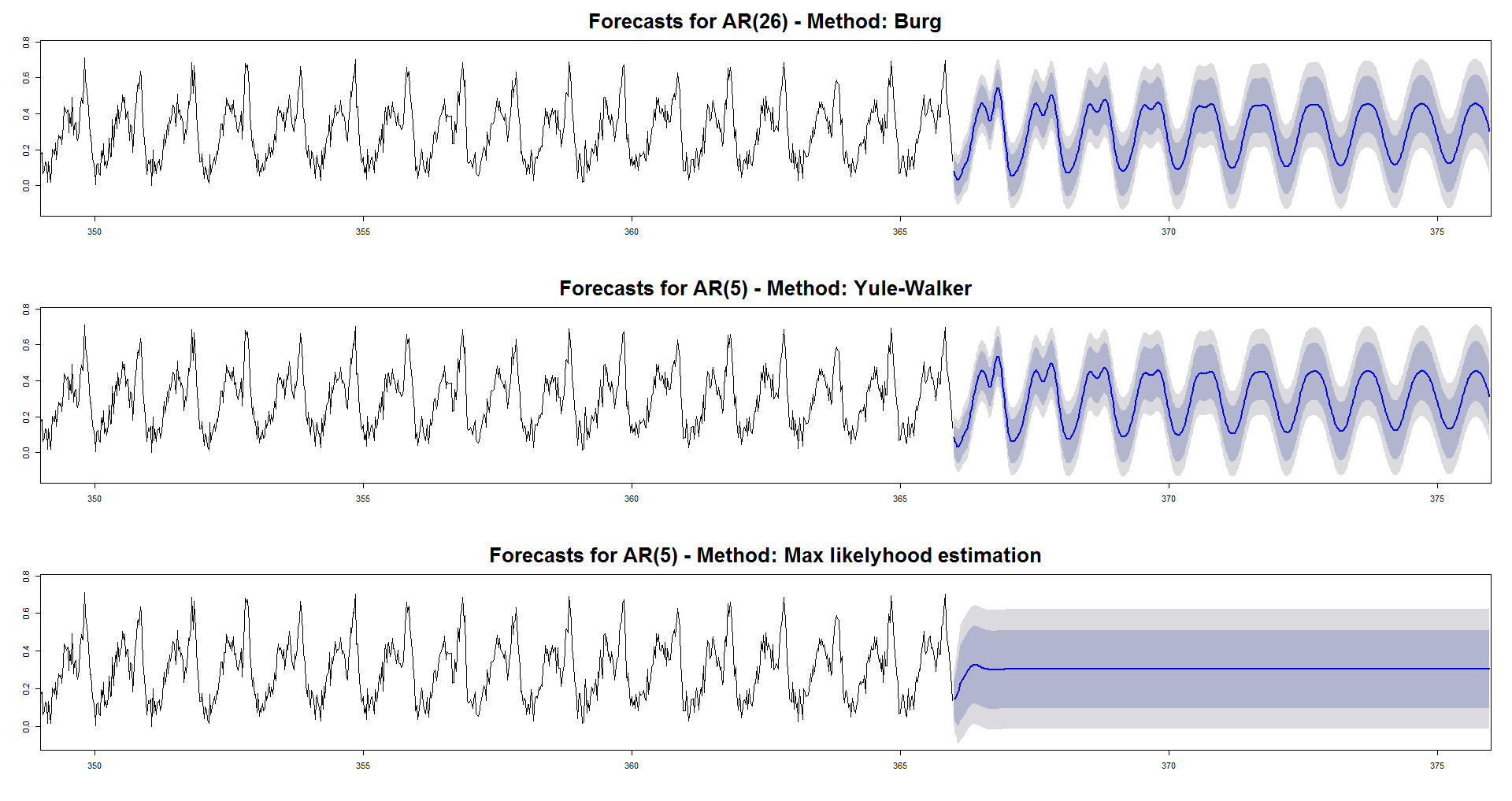
We combine the above equations to get an ARMA model of order

To find the how many lags to use (), we use the autocorrelation function (ACF), it is defined as follows:

The partial autocorrelation function (PACF) is used to measure the degree of association between and. In PACF, we do not consider the effects of the intermediateto , only and are considered in the ACF. Both ACF and PACF are used to see to what extent values in the series are related to past values, this helps us choose a suitable and . Below we look at the PACF and ACF for the previously given generated data (Acorn-U).

**Figure 12.** Illustration of (partial) auto-correlation.

The PACF drops significantly after the 1st lag this implies that the series is an Auto Regressive (AR). Although the drop is significant, it does not level out until roughly lag 5. So the order can be between 1 and 5.

The AR model was tested with 3 different methods (R built in) – Burg, Yule-Walker (YW) and Maximum likelihood estimator (MLE). The models use 1 year of generated data (17520 half hour intervals). 

**Figure 13.** Shows forecasts for different AR implementations.

The Burg and YW methods perform quite well, they smooth out over time but the forecast is fairly accurate for the first few days. (The R code can be found in the *Analysis.R* file).

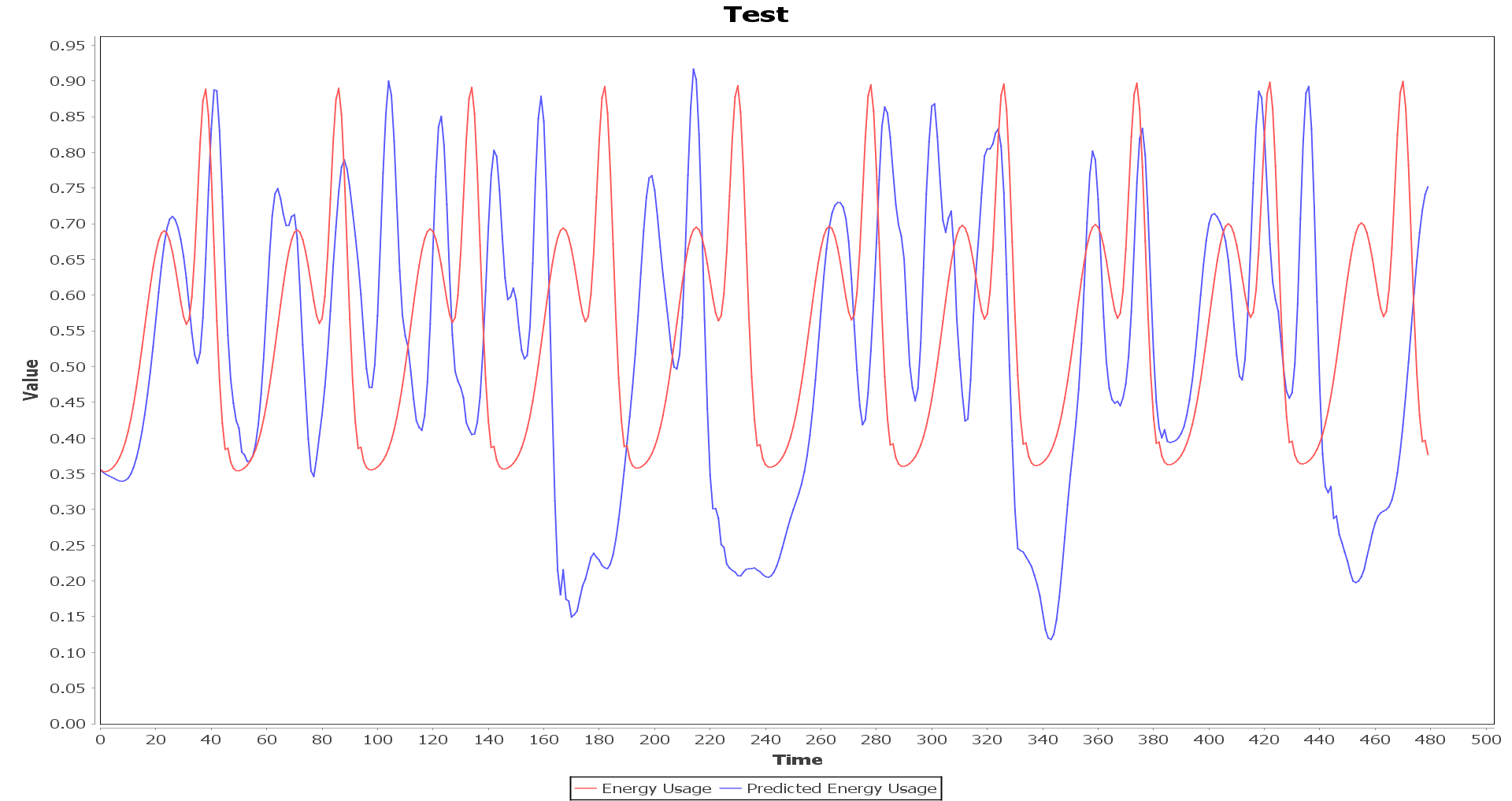
Although the forecasts given above are not bad, it is likely ARMA will not perform well when yearly seasonality and behaviour changing is introduced. For this reason the next section will focus on more advanced machine learning algorithms for forecasting and will introduce Weka as a tool for testing these algorithms.

## **6.3 Weka**

Weka is a java based data mining tool, it includes tools for data pre-processing, classification, regression and visualisation among others. The Weka library can be accessed directly from java code which allows easy design of custom algorithms and machine learning schemes. Weka is available under the GNU General Public License and so is suitable for use in this project at least while it remains private research. If this work was ever to become commercially viable, Weka would be replaced by a specially design machine learning library to do this is a partial goal of the project but for the moment the Weka library in particular the algorithms provided by the TimeSeriesForecasting package will be used in testing.   
(Weka home page: <http://www.cs.waikato.ac.nz/ml/weka/>,   
TimeSeriesPackage:<http://wiki.pentaho.com/display/DATAMINING/Time+Series+Analysis+and+Forecasting+with+Weka>,<http://weka.sourceforge.net/packageMetaData/timeseriesForecasting/index.html> ).

### **6.3.1 Initial Exploration of Weka**

After building a façade between the Weka library and the current project build: (See: *MachineLearningMain* and *LearningExperimenter* classes (demo package)). These classes take the auto generated data from the *ExperimentDataGenerator* class and apply a given Weka Classifier (algorithm) to the data. These classes are a simplified test bed for machine learning algorithms, the agent simulation does not need to be run (unless we want to see how the algorithm handles behaviour alterations (to be discussed)).

The below graph shows the forecast given by Weka’s MultilayerPerceptron classifier with its default parameters. The given data is the pure Acorn-U model (0 error). The model was trained over 300 days (14400 instances) and attempts to forecast 10 days (480 instances).

**Figure 14.** Graph giving an initial look at forecasting in Weka

## **6.4 Model Evaluation**

To perform evaluation of different models provided by Weka I will be using various error metrics the main being MSE: where is the prediction of the real value. To begin talking about the methods I will be using for evaluation some terminology and concepts must be introduced, these are as follows.

### **6.4.1 Overfitting**

When applying a model to some data, we are trying to represent a function that describes the process that generated the data. Data often comes impure – an element of random noise is involved, this we do not want to capture as it does not correspond to the process that generated the data but likely from the instruments or imperfect environments that the data was collected by/from. When a model captures this noise, we are overfitting the model. This usually happens if the model is excessively complex in comparison to the function we are trying to model. We do not require a 17th order polynomial model if the underlying model is linear. If a model is over-fitted it will likely perform poorly on data that has not yet been seen but very well on the current training data which is not what we want!

Underfitting is also an issue that must be considered, it is simply the opposite of overfitting – the model we are using is not powerful/complex enough to represent the function we are after, we want to avoid this as well. There are a number of techniques to avoid overfitting and underfitting, one of these which will be used in this project is Cross Validation.

### **6.4.2 Cross Validation**

Cross validation is a model evaluation method, in essence it tests a model against new data during the training process to hopefully reduce the chance of overfitting. K-fold cross validation involves splitting our data into K folds. In each fold we split the data in to test and training data (how this is done varies) and apply our model. For example K = 3:

Test

Training

Fold 1:

Test

Training

Fold 2:

Training

Test

Training

Fold 3:

**Figure 15.** Illustration of cross validation.

A good value for k may be 10, we then split the data randomly into 90% training and 10% test data. Cross validation may be used for general machine learning models however it must be altered slightly for time series data as we cannot take random subsets of our data to form the test and training sets.

### **6.4.3 Cross Validation for Time Series Data**

In cross validation for time series data we still split into K folds. The folds look slightly different however. Say k = 3 again.

Test

Training

Fold 1:

Test

Training

Fold 2:

Test

Training

Fold 1:

**Figure 16.** Illustration of cross validation for time series.

## **6.5 Time Series Models in Weka**

There are a few available time series models in Weka, each will be discussed in the following section. The Time Series Weka package performs the configuration of these models to suit them to time series data. This will also be discussed, as well as the control over parameters (such as lag) that the package provides.

They are as follows:

* MultilayerPerceptron (Artificial Neural Network)
* SMOreg (Support Vector Machine Regression Model)
* Gaussian Process
* Linear Regression

## **6.5.1 MultilayerPerceptron**

The MultilayerPerceptron classifier provided by Weka is an implementation of an Artificial Neural Network (ANN). To begin discussing this model some time will be taken to introduce ANNs in general.

A basic Neural Network can be visualised as follows:

Input

Hidden

Output

Networks are made up of layers of nodes (represented as circles here): Input is where data is fed into the network. Hidden are the nodes between Input and Output (they are called hidden because it is difficult to explicitly define their contribution to learning once training has begun). Output are the nodes which output data. The structure of the network depends on the problem to be solved and may have more than one hidden layer (known as a Deep ANN) and/or different size input/output layers.

**Figure 17.** Illustrates a simple ANN.

The model in question uses the famous Backpropagation algorithm to train the network and the activation function (or logistic function) for neurons is sigmoid [18] see **Figure 2***Figure 18*. Now we present a very basic example to illustrate the backpropagation algorithm. We start by introducing the weights, the input values (i.e. the training data) and the desired output values (i.e. the test set). (The rows in each matrix are the weights of edges connected to the next set of neurons from the previous, e.g. )

First we perform the ***forward propagation***:

Compute net input for and from and.

‘Squash’ the net input to get the output of each neuron. In this case we will use a sigmoid activation function where. The reasons for using such a function will become clear later.

Compute the net input for and by repeating steps 1 and 2 using weights and input

We now compute the total error using where n is the number of output neurons.

Now we move to *the* ***backward propagation***:

We want to know how each weight affects, to do this we take the derivative of with respect to a given weight:

(This is where the from the calculation comes in handy)

The following is one of the reasons why sigmoid functions are often used as activation functions, because they are easily differentiable.

We now compute for each weight and get the following matrix:

Now we must carry out a similar process for. However we must consider that has an effect on all outputs and so we have:

(Where n is the number of output neurons and is the error of a single output neuron)

, recall that and

First compute

Then, and

Compute for each weight

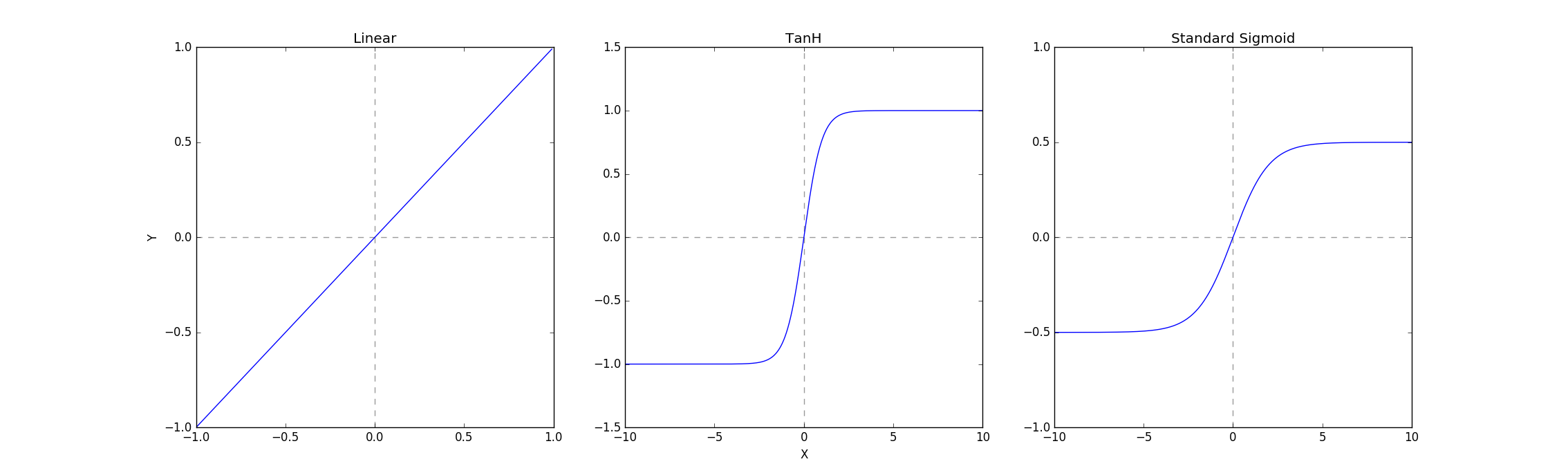
Now that we have and we can update the all the weights in both and.

This is done using the formula:

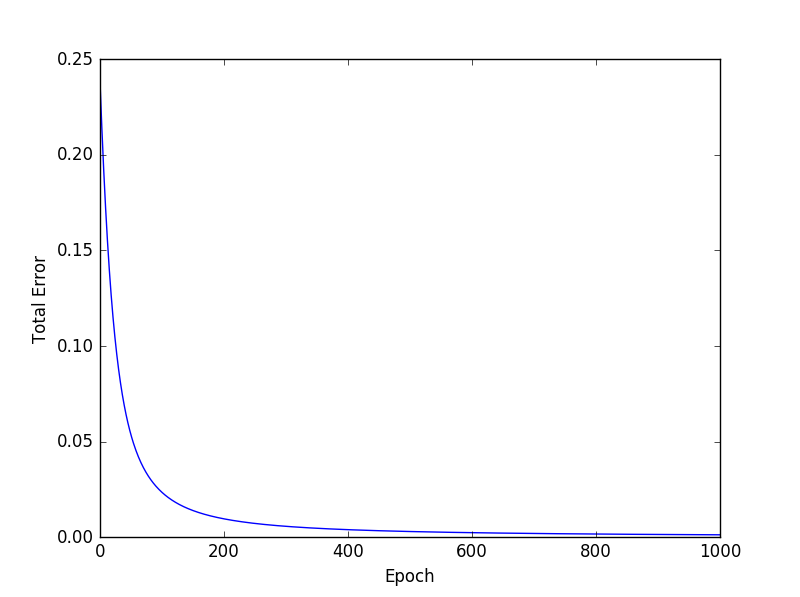
(Where is the learning rate)

Now if we apply this formula with

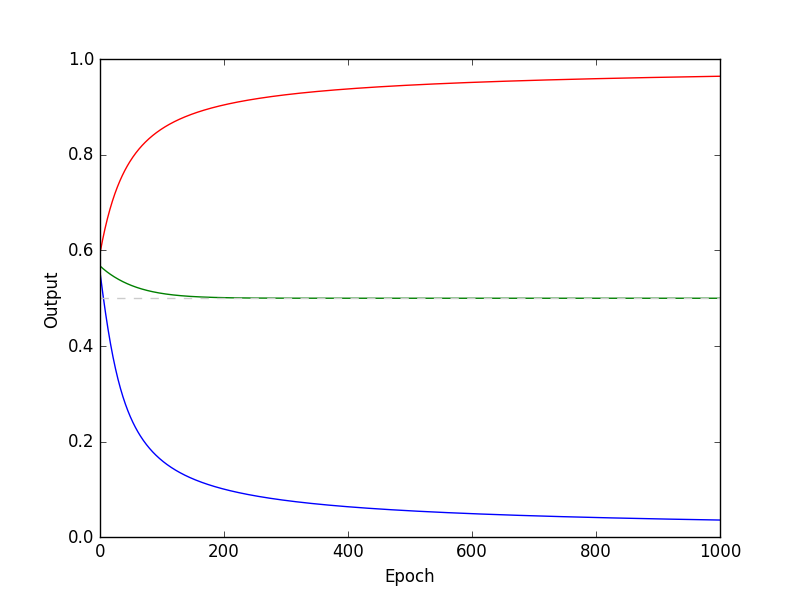
Now, we repeat the process a number of times to train the network, if we check on each epoch (iteration) it should reduce. After training the network for 1000 epochs, we see reduce until it is very close to zero see we also see trend towards (see *Figure 20*).



**Figure 18.** Shows three logistic functions that may be used in an ANN as ‘squashing’ or activation functions, these functions are used to keep neuron output bounded. Source code: ActivationFunctions.py

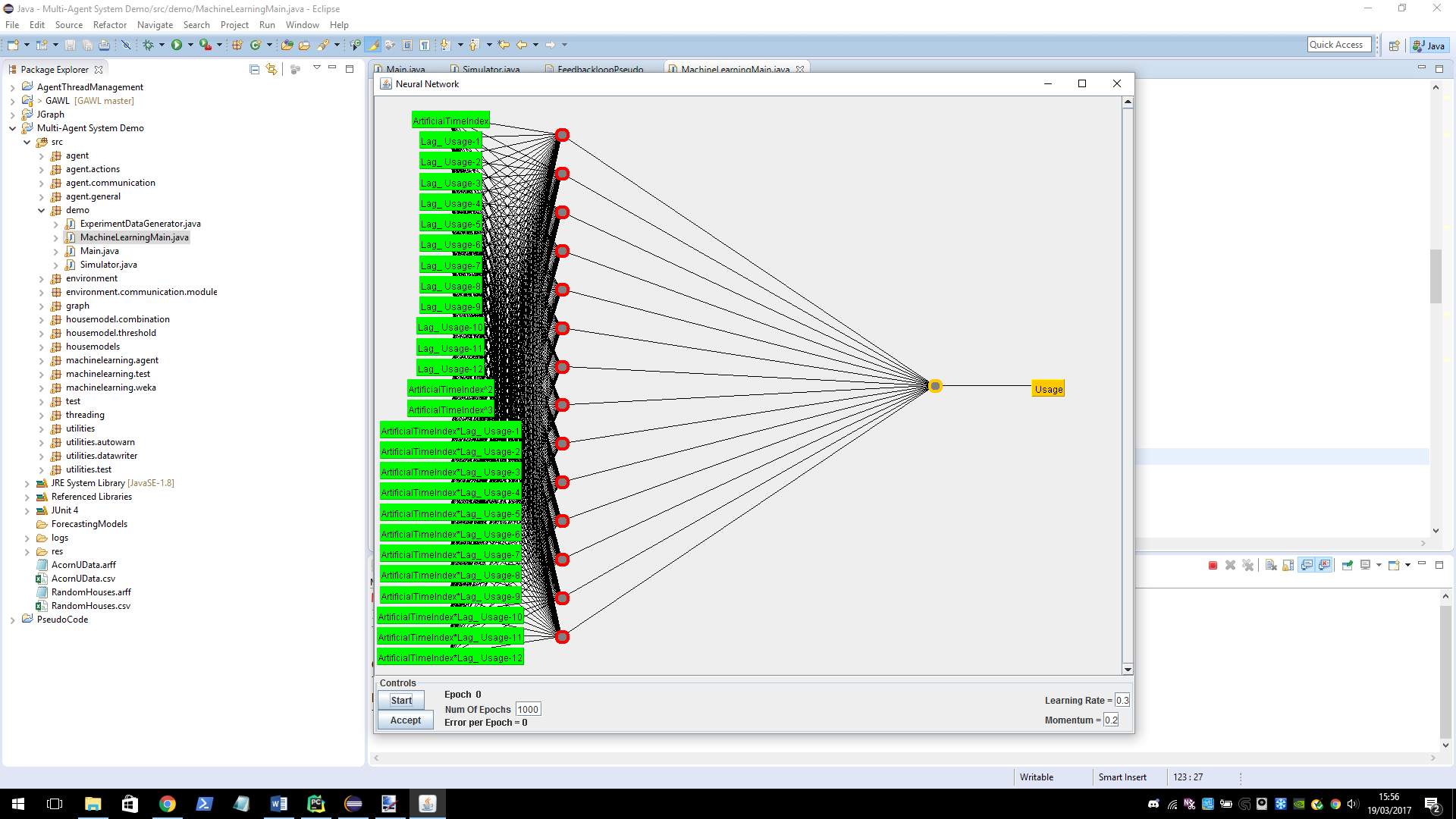


**Figure 20.1** shows the reduction as the ANN is trained. Source code: ANNtest.py



**Figure 20.2** Shows how the output changes as the ANN is trained. Source code: ANNtest.py

Now back to Weka and the implementation used for testing in this project. Weka provides a visualisation tool for MultilayerPerceptron which displays the structure of the network as well as some key information such as the learning rate, error during training and the current epoch. The default structure is given in *figure 21,* the results of employing Weka implementation are given in *Chapter 8.* As well as the MultilayerPerceptron forecaster, SMOreg was also tested. The next section will focus on the Support Vector Machines (for regression) and the Weka implementation of SMOreg.



**Figure 21.** Visualisation provided by the Weka library of the default MultilayerPerceptron structure for time series analysis.

## **6.5.2 Support Vector Machines**

We shall start by giving the theory behind support vector machines, how the maths is used for classification problems. Looking at *figure 22*, the goal of Support Vector Machines (SVM) is to final a maximum margin hyperplane (MMH), a hyperplane that is farthest away from the all training data but separates them perfectly. We know that if we can separate out data with a hyperplane then there are infinitely many such hyperplanes, so how do we find the MMH among the infinite set?

**Figure 22.** Illustration of binary classification, in 2 dimensions produced by SVM. The solid line shows the maximum separation of the two classes. Source code: SVMGraphs.py



Let’s get into the derivation,

We say the vector is perpendicular to the MMH (which we are yet to find) and the vector is some unknown that may lie anywhere in the data space.

We then say that if then we have a positive sample, otherwise a negative, where is some constant.

How do we find and,

We define to be any positive sample, to be any negative sample, and



We then multiple 1 and 2 by 3, giving:

The come out to be the same as we are multiplying the right hand side by -1 or +1 based on whether it is a positive or negative sample. This leads to:

and

when is on the boundary of the margin (in *Figure 22 on one of the dotted lines)*. From this we can find the width of the margin (see *Figure 23* for an illustration of this)



**Figure 23.** *Shows the vectors and on the boundary of the maximal margin.* Source code: SVMGraphs.py

We can find the of the maximal margin using,

as we previously define to be perpendicular to it.

Now we can use from before, we know the so,

Substituting this into the formula we get,

So we have found that we want to maximise, as this is the of the margin.

To simplify the maths later on, we say that we want to minimise or to make it even easier.

We now need to introduce *Lagrange Multipliers,* this technique is used to find a local minima or maxima subject to some multivariate function that is equal to a constant. In our case that multivariate function is:

The form of a *Lagrange Multiplier* is:

So, substituting in what we have:

To find the local minima we must differentiate with respect to and and these are the only varying values and set them to 0.

Which implies,

If we plug these back in to the original equation:

Notice that the optimisation of this function depends only on the dot product of pairs of inputs and. Once we have optimised this function, we have found the maximal margin hyperplane! We say,

We have found the solution to linearly separable spaces like in *Figure 22*, what happens if the data is not linearly separable like in *Figure 25?* Well, because the optimisation only depends on the dot product of input pairs, we can use *Kernels*. Kernels transform one space into another space, in our case perhaps we wish to map our 2-dimensional input space to a 3-dimensional or higher feature space. A kernel function has the form:

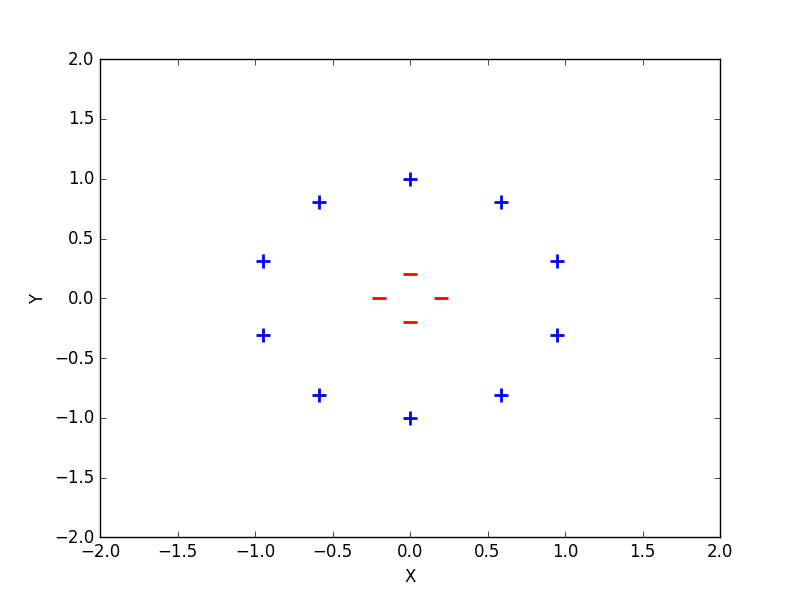
There are many different kernels that can be used, including:

Degree- polynomial kernel:

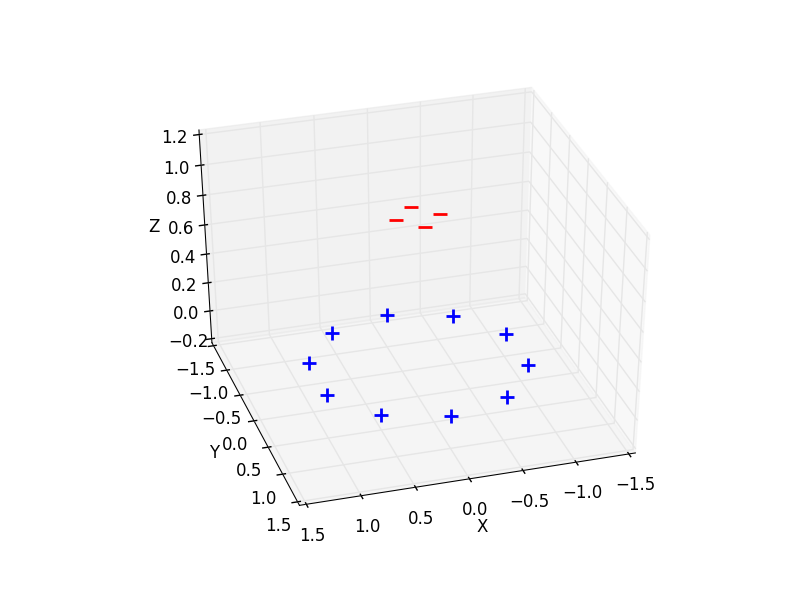
Radial kernel:

In our case, will have some mapping:

Notice that in *Figure 25* a hyperplane can now separate the positive and negative examples, we can use SVM to find the maximal margin hyperplane in this mapped space.[[5]](#footnote-5)

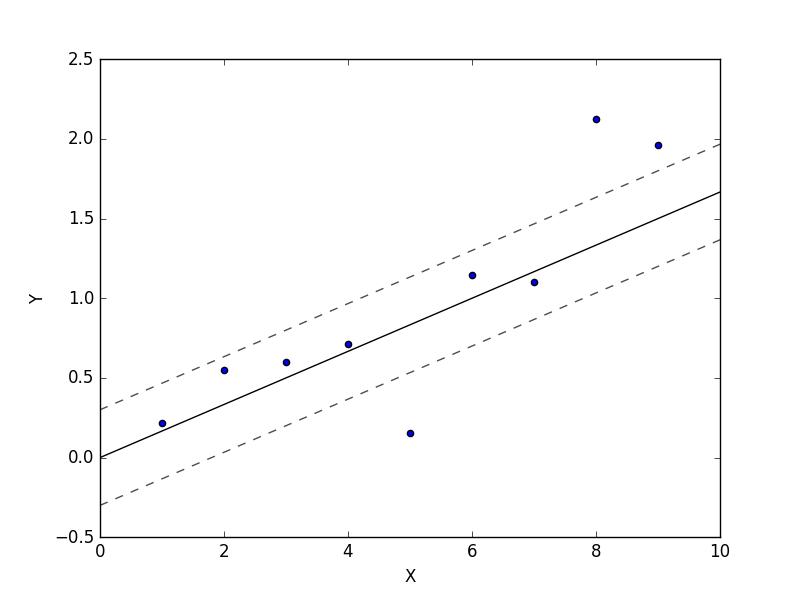


**Figure 25.1.** Illustration of a non-linearly separable space. Source code: SVMGraphs.py



**Figure 25.2.** Mapping of figure 24 into 3-dimensional space through the use of a kernel function. Source code: SVMGraphs.py

So far we have discussed SVM for classification, however as we are applying it to time series will shift the focus to regression. SVM for regression does a similar thing, however we are trying to optimize the bounds for a given regression. Consider *figure 26,* we will use a simplelinear example, say



**Figure 26.** Illustration of 1-dimensional linear SVM regression. Source code: SVMGraphs.py

[19][20]

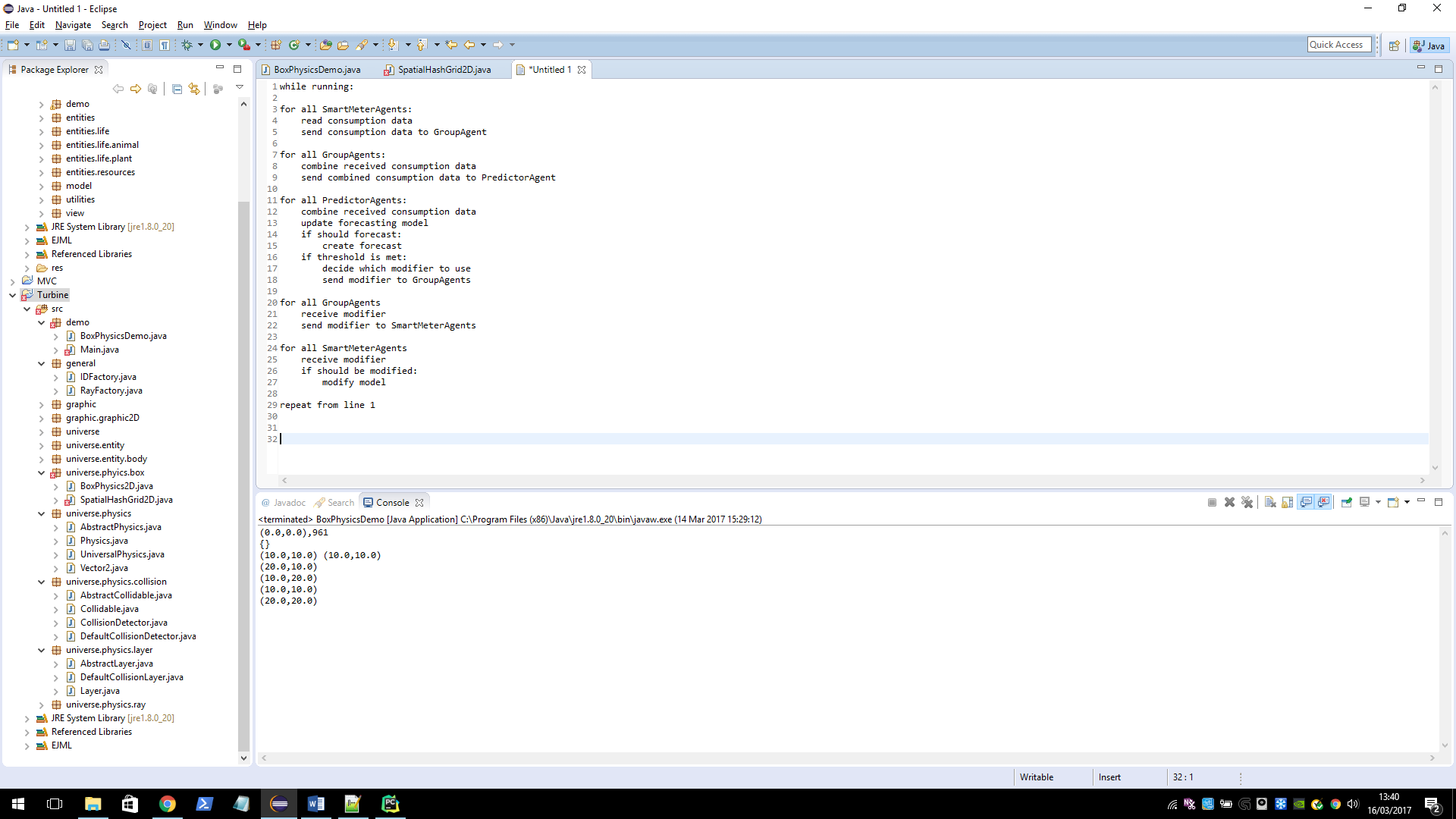
# **Chapter 7: Developing the Model Further**

One of the main goals of this project is to simulate Demand Side Management (DSM). We want the system to be able to adapt to changes in the customers behaviour – most importantly we want the forecasts to be representative of constantly changing consumer behaviour. We will be assuming that DSM is working i.e. the incentive for the consumer is high enough for them to alter their behaviour accordingly. To simulate this a feedback loop has been introduced into the system.

## **7.1 Feedback loop**

The feedback loop involves many different new concepts, all will be introduced and discussed shortly. Here is a section of pseudo code that describes the full feedback loop, it will be referenced later.

**Figure 27**. Pseudo code for the feedback loop.



Notes:

* The structure of the network in which messages are sent is arbitrary. The only properties that hold are:

SmartMeterAgent <-> GroupAgent

GroupAgent <-> PredictorAgent

GroupAgent <-> GroupAgent

* The actions in each loop are by no means comprehensive, the actions presented only describe the feedback loop.

To begin we should briefly return to the idea of combination (*Figure 27,* lines 8 and 12) the concept was first introduced in section *5.4* above and here we will continue in an agent context see the below *figure 28.*

**Figure 28.** Illustration of a simple combination, at the bottom node the values are raw consumption readings for one time interval. Traversal upward shows the values being combined additively. The structure of this hierarchy of combinations is arbitrary, there may be many layers of group agents.

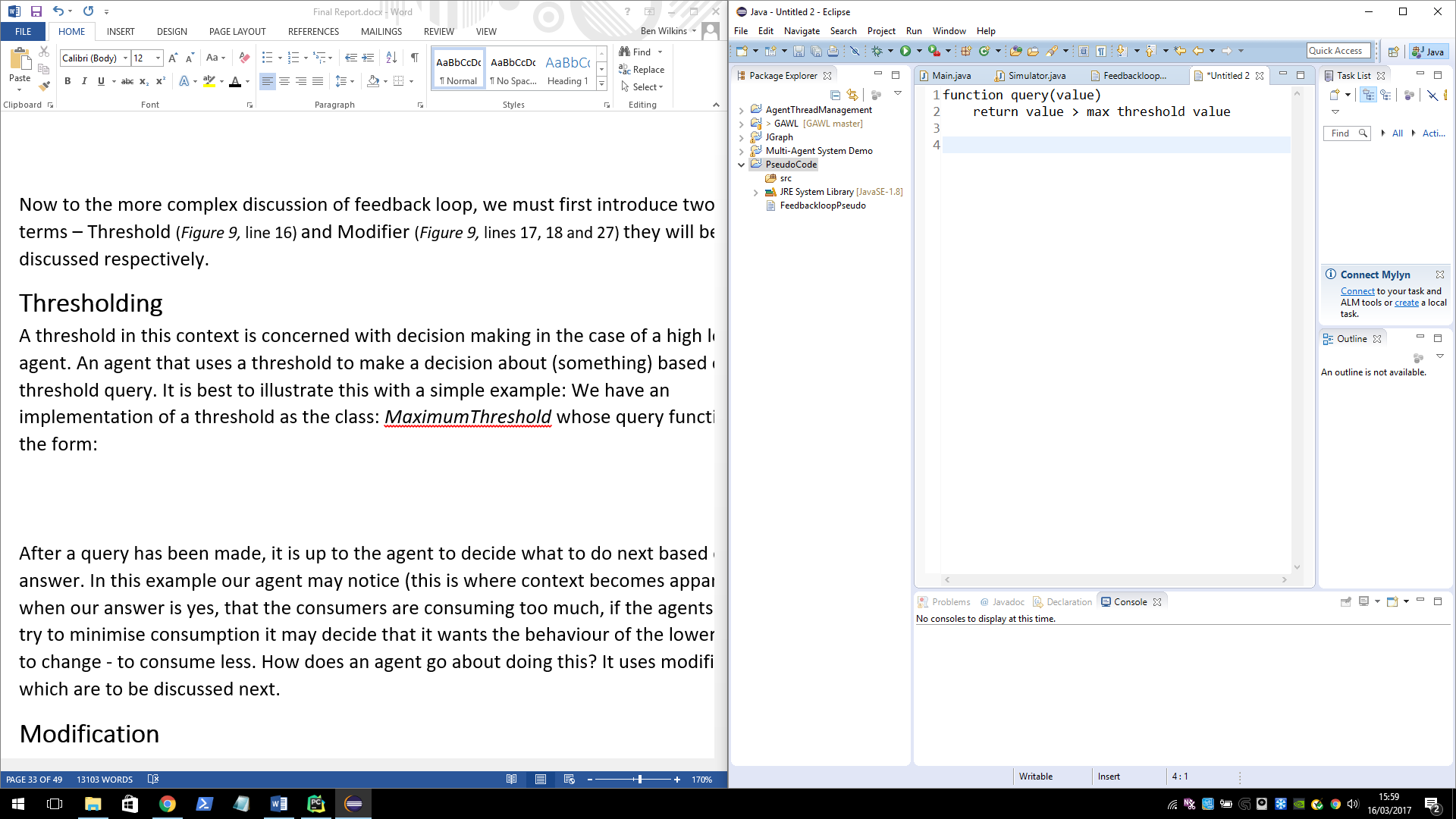
Predictor Agents

Group Agents

Smart Meter Agents

Now to the more complex discussion of feedback loop, we must first introduce two new terms – Threshold (*Figure 27,* line 16) and Modifier (*Figure 27,* lines 17, 18 and 27) they will be discussed respectively.

### **7.1.1 Thresholding**

A threshold in this context is concerned with decision making in the case of a high level agent. An agent that uses a threshold to make a decision about (something) based on a threshold query. It is best to illustrate this with a simple example: We have an implementation of a threshold as the class: *MaximumThreshold* whose query function is of the form:

Where ‘*max threshold value’* is some pre-set value, the agent is asking: is this ‘*value’* too big. After a query has been made, it is up to the agent to decide what to do next based on the answer. In this example our agent may notice (this is where context becomes apparent) when our answer is *True*, then the consumers are consuming too much, If the agents goal is to try to minimise consumption it may decide that it wants the behaviour of the lower agents to change - to consume less. How does an agent go about doing this? It uses modifiers, which are to be discussed next.

### **7.1.2 Modification**

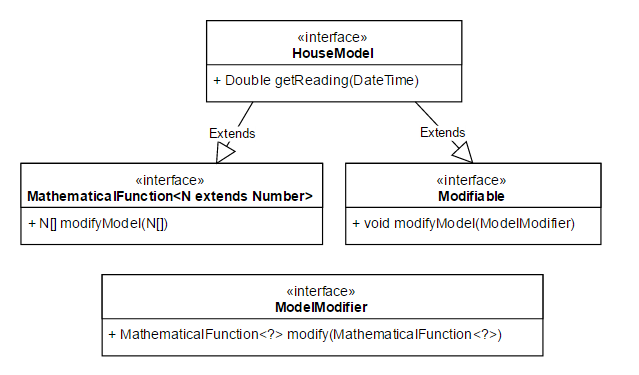
Again it would be best illustrate the concept with an example. Say we have a class *ModelModifierMean*, and a function *ModelNormal* which represents a standard Gaussian distribution from which samples will be taken. The sole job of the *ModelModifierMean* class will be to alter the mean value of the *ModelNormal* function and so to ‘shift’ the function to the right or left. There are no restrictions on modification of a function other than the ones that are inherent in the implementation (to be discussed shortly). In principle, or at least in this system, a *ModelModifiers* job is to alter the ‘behaviour’ (simply the function) of a model used in a system. The term behaviour is used instead of function as it better describes the happenings in a real house hold i.e. the behaviour of its occupants, although in the simulation they can be used interchangeably.

### **7.1.3 Implementation of Models, Modifiers and Thresholds**

It will be helpful to discuss the implementation of these concepts in java as this (as in most cases of implementation) reduces perhaps first unseen restrictions. At the base lies the interface *MathematicalFunction* whose only method is: *compute* which takes in an *Array* of arguments of generic type (extending *Number)*and returns an *Array* of the same type. It is up to a class implementing this interface as to what compute does, but this method is essentially the function. (It should be noted that function is being used here as a computational term concerning numerical inputs and outputs, and is not related to the algebraic definition).

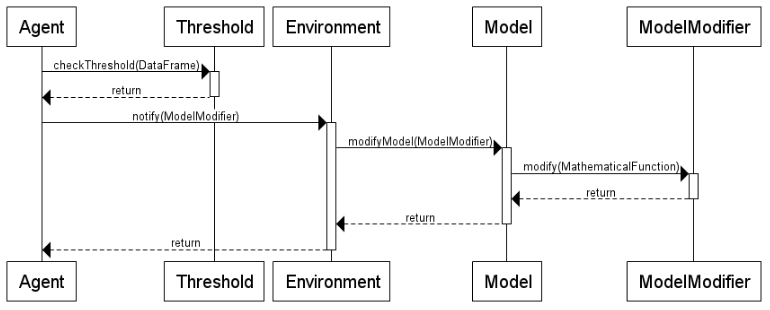
Given a *MathematicalFunction*, we would like to be able to modify it when it suits us. This is done by a class implementing *ModelModifier* whose only function is *Modify* which takes a single argument of type *MathematicalFunction* and returns the modified *MathematicalFunction* (of course in java the changes to the argument will be reflected in the callers reference to the object, giving a return value allows the function to be completely changed, even perhaps to return a different function altogether). In general, the relationship between *MathematicalFunction* and *ModelModifier* should be one to many. That is for each concrete implementation of *MathematicalFunction*, if we wish it to be modified we require a concrete implementation of *ModelModifier* for each example of modification we desire.

We should now move to discuss *Models*, this will be done in context to provide a more concrete example. Any modify-cation of a *MathematicalFunction* in this system will be done through an intermediary - (as the name *ModelModifier* suggests) a *Model* and in this case, a *HouseModel*. The key interface for this process is *Modifiable*, this interface has a single method modifyModel which has a single parameter of type *ModelModifier*, a concrete implementation of this method should call the *modify* method and provide its underlying *MathematicalFunction* to it. In this system a *HouseModel* (which is the interface underlying all models that represent the behaviour of a household) extends both, *MathematicalFunction* and *Modifiable* and when calling *modifyModel* it will provide its underlying mathematical function (which is stored as an object).



***Figure 29****. UML Class diagram showing the relation between the above mention interfaces.*

The implementation of *Thresholds* is last to be discussed, it is given as an Interface with a single method *checkThreshold* which returns a True or False (Boolean) value has a single parameter of type *DataFrame* (a class that contains some data be it numerical or otherwise). A concrete class implementing *Threshold* should provide an analysis of the data in the given *DataFrame* and reduce it to a Boolean answer, the interpretation of this answer is up to the caller (i.e. the agent). Below shows the relation between each concept in the form of a UML sequence diagram, it also illustrates part what is meant by the ‘feedback loop’ in the system (the part being the altering of behaviour).



**Figure 30.** Simplified illustration of modification in the feedback loop as UML sequence diagram.

A number of intermediate steps are omitted (for simplicities sake) namely – the steps involved with the notify method between Agent and Environment. Here Agent means top level agent, e.g. the Predictor agent or Neighbourhood agent. Now we are in a good position to really define what we mean by feedback loop. After the top level agent depicted here has altered the behaviour of some underlying model(s) this is reflected in the data that is being collected by the lower level SMAs, this data is then streamed up the agent hierarchy as presented in earlier sections. This is essentially the definition of the feedback loop (given diagrammatically blow).

It should be noted here that a *Model* is not obligated to change its behaviour and in the context of this project it seems right that a *Model* most, if not all of the time should not change its behaviour, at present probability is used as the deciding factor (i.e. each model may have a 5% chance to change when presented with a *ModelModifier* by the top level agent).

Models

Modify

Provide

**Figure 31**. Diagrammatic feedback loop. The bottom edge represents data collection performed by the SMAs.

Given this development in the system it will be possible to simulate such an environment and test such algorithms. To further illustrate the capabilities of this development some scenarios will be run and results given in the next *Chapter 8*: Simulation Scenarios*.*

## **7.2 Missing Data**

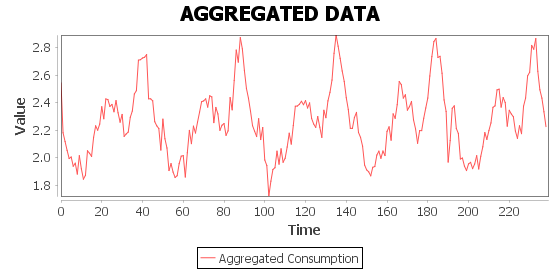
Here we will address the problem of missing data. In a real world setting in which network failures are possible, it may be that some SMAs fail to report readings for given times either completely i.e. they are lost, or partially i.e. they are sent later in time than what is expected. In both cases the top level agents must be able to deal with such an event. Presently a simple mechanism is in place to deal with this – when combining data from each SMA, if any data is missing for a given time, the average of the other examples for that time is used to fill the empty slot. This is by no means a comprehensive solution, and is hardcoded into the *ReadingCombinator* class (which handles the combination of readings from SMAs given a base *Combinator* such as an *AdditiveCombinator*). As a future extension to the project, this should be implemented in a more flexible fashion – to give the option to provide different ways of handling missing data. One such example may be to take the previous reading from the same house.

It should also be noted here that somewhere in the simulation there is a failure; occasionally an SMA does not report its reading during simulation, this is an unexpectedly difficult bug to find and the prime suspect is the thread management system. Due to time constraints it will not be full addressed. As luck would have it, it provides a good demonstration for missing data and so also for the techniques used to solve the missing data problem. For this reason it is not a top priority at this time, however, admittedly it does illustrate a flaw in the system architecture.

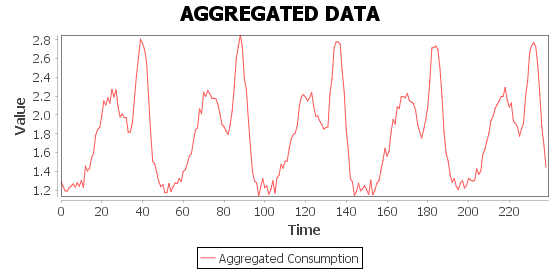
# **Chapter 8: Simulation Scenarios**

## **8.1 Scenario: Basic**

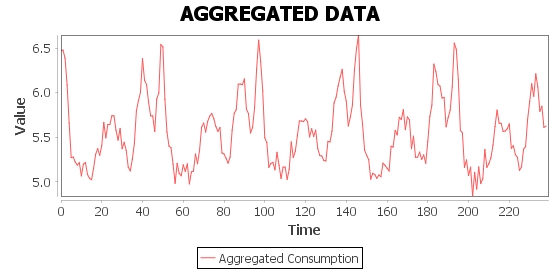
The first set of scenarios (*Figure 32*) will show the simulation system working without any more complex functions acting on it (i.e. prediction, modification, etc). Each was run for approximately 5 days (240 intervals), the error term *E* will be set small (0.05) for the sake of demonstration.



**Figure 32.2.** Four house {Acorn-U,   
Adversity,   
Affluent, Comfortable}



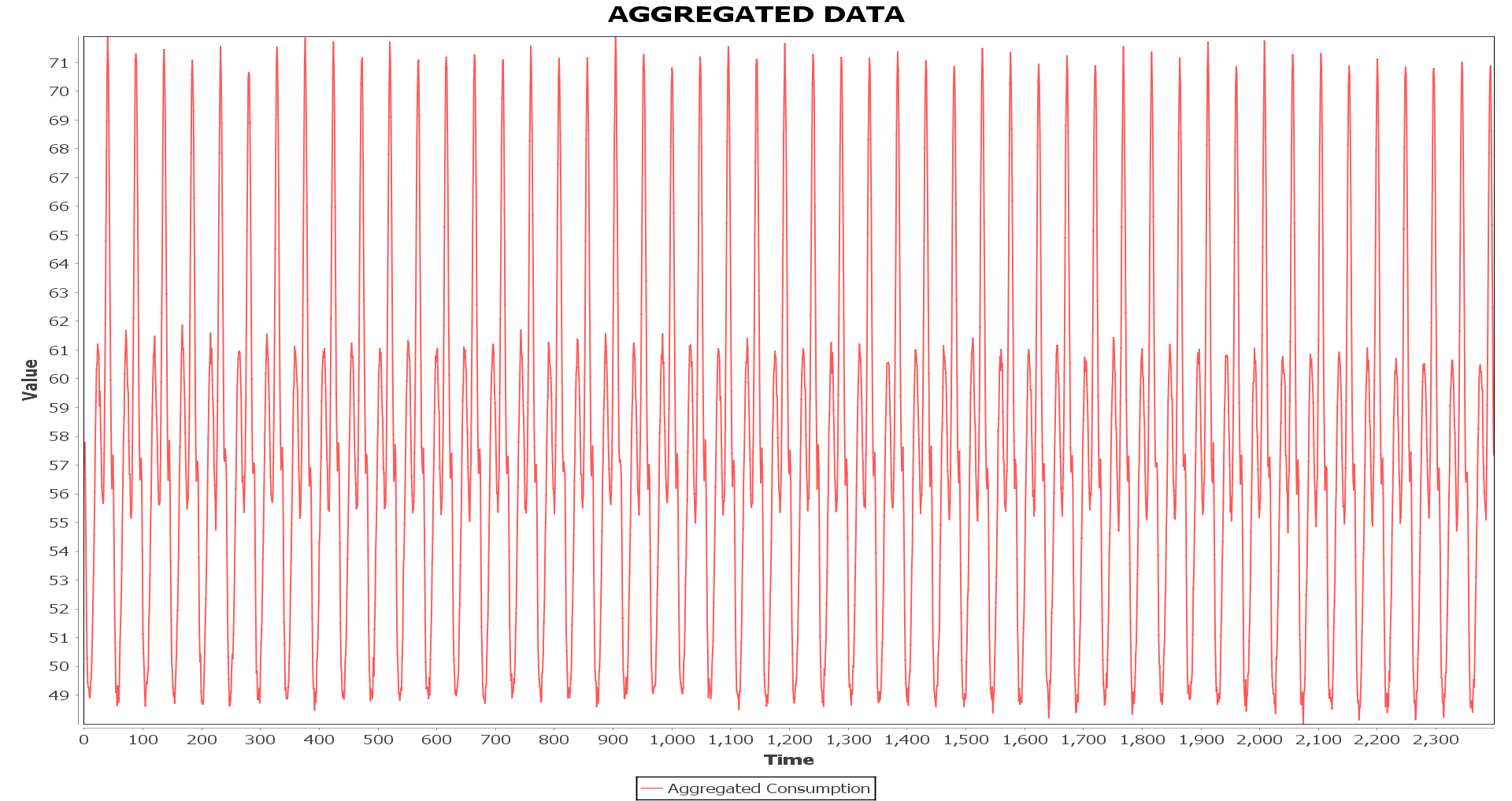
***Figure 29.1.*** *Three houses {all Acorn-U}*



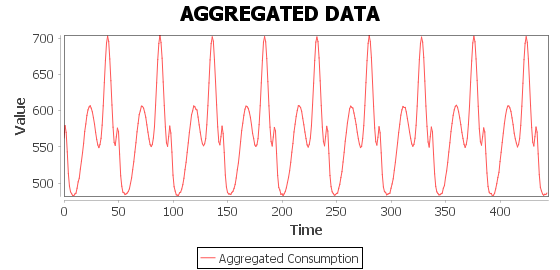
***Figure 29.3.*** *Ten random houses, {Acorn-U x2,   
Adversity x4,   
Affluent x4 }*

## **8.2 Scenario: Stress**

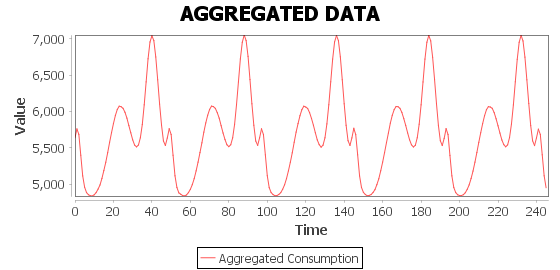
This section will focus on trying to break the system, to see at what point the system no longer works (if applicable). Tests are given for different numbers of agents ranging from 100 to 10000. Each will give additional information such as, time per cycle (see section 3.6) at key points, the number of missing data values etc. See *figure 33.*



**Figure 33.1.** Running 100 random houses for 50 days. Info: There was no cycle time difference when comparing the start to the end, average cycle time 0.020 seconds. A total of 9 missing values were given.



***Figure 30.2.***  *Running 1000 random houses for 10 days. Info: There was no cycle time difference when comparing the start to the end, average cycle time 0.186 seconds. A total of 6 missing values were given.*



***Figure 30.3.***  *Running 10000 random houses for 5 days.   
Info: There was no cycle time difference when comparing the start to the end, average cycle time 2.020 seconds.*

It seems that the cycle time is linearly proportional to the number of agents, this is very good as it means the system is scalable.

## **8.2 Scenario: Modification**

### **8.2.1 Magnitude**

**Figure 34.** Shows magnitude modification over time.

The magnitude modifier scales the function by some value, in this case the function is scaled by 0.99 (i.e. all values in the function will be reduced by 1% each time the function is modified). In another example it may be possible for the function to grow however in this context an agents goal should always be to reduce energy overall consumption.

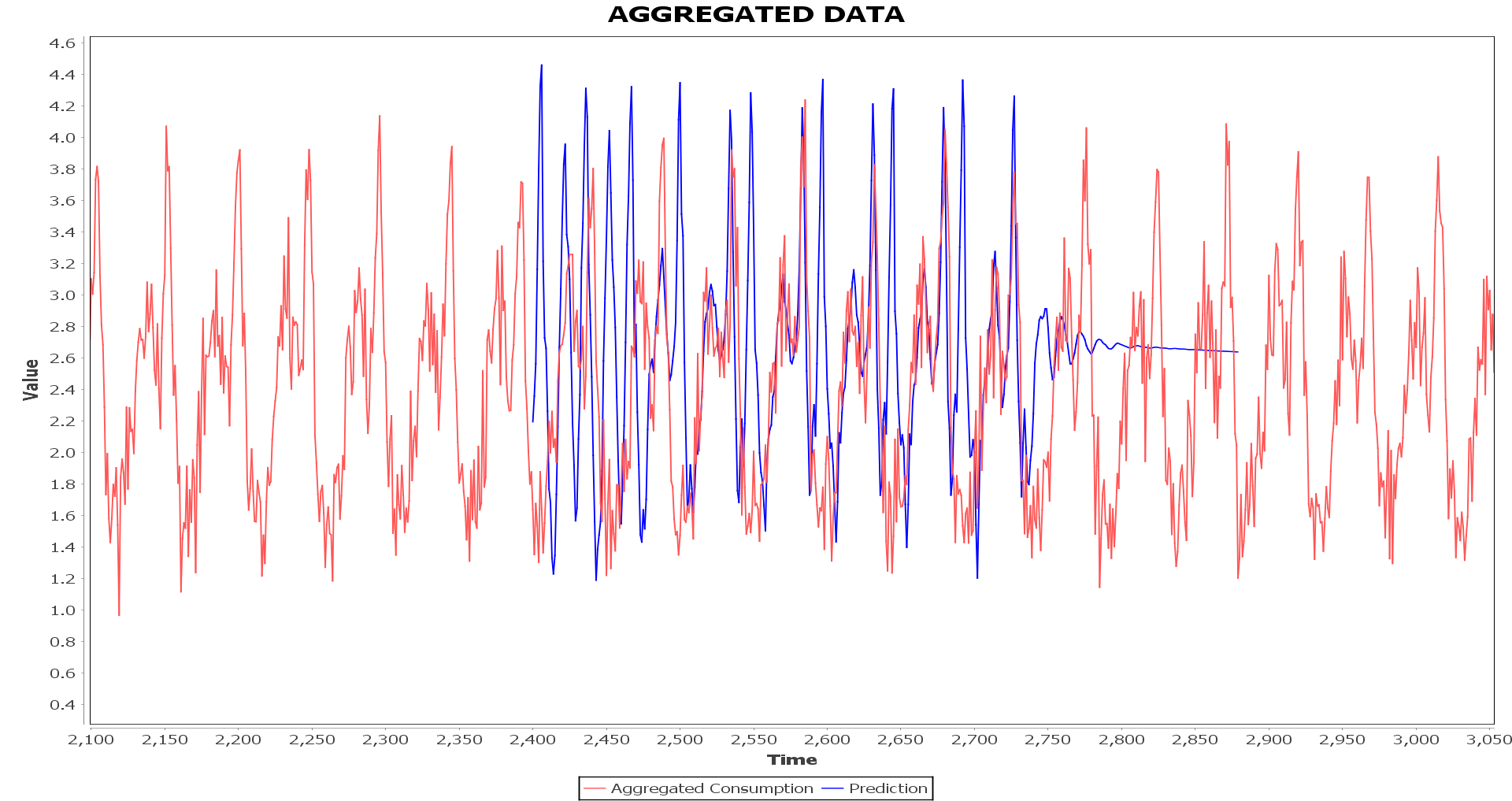
### **8.2.2 Conclusions Regarding Modification and Underlying Model Functions**

It has come to light that the combined normal function and its implementation was not the most practical and extendable model to use. This became apparent after an attempt at creating a *ShiftModifier* (a modifier that would shift the function left or right by some amount), as the implementation used a ‘loop around the edge’ approach it was difficult to specify the correct modification in the modifier. Given that the system is set up to use any function as a base for house models solving this issue can be thought of as a possible extension to the project. To change the function from being two normal distributions to being sinusoidal would be one valid option. In hindsight it would have been better to implement the model function as sinusoidal in the first place.

## **8.3 Scenario: Weka Multilayer Perceptron Forecasting**

### **8.3.1 Null Modifier**

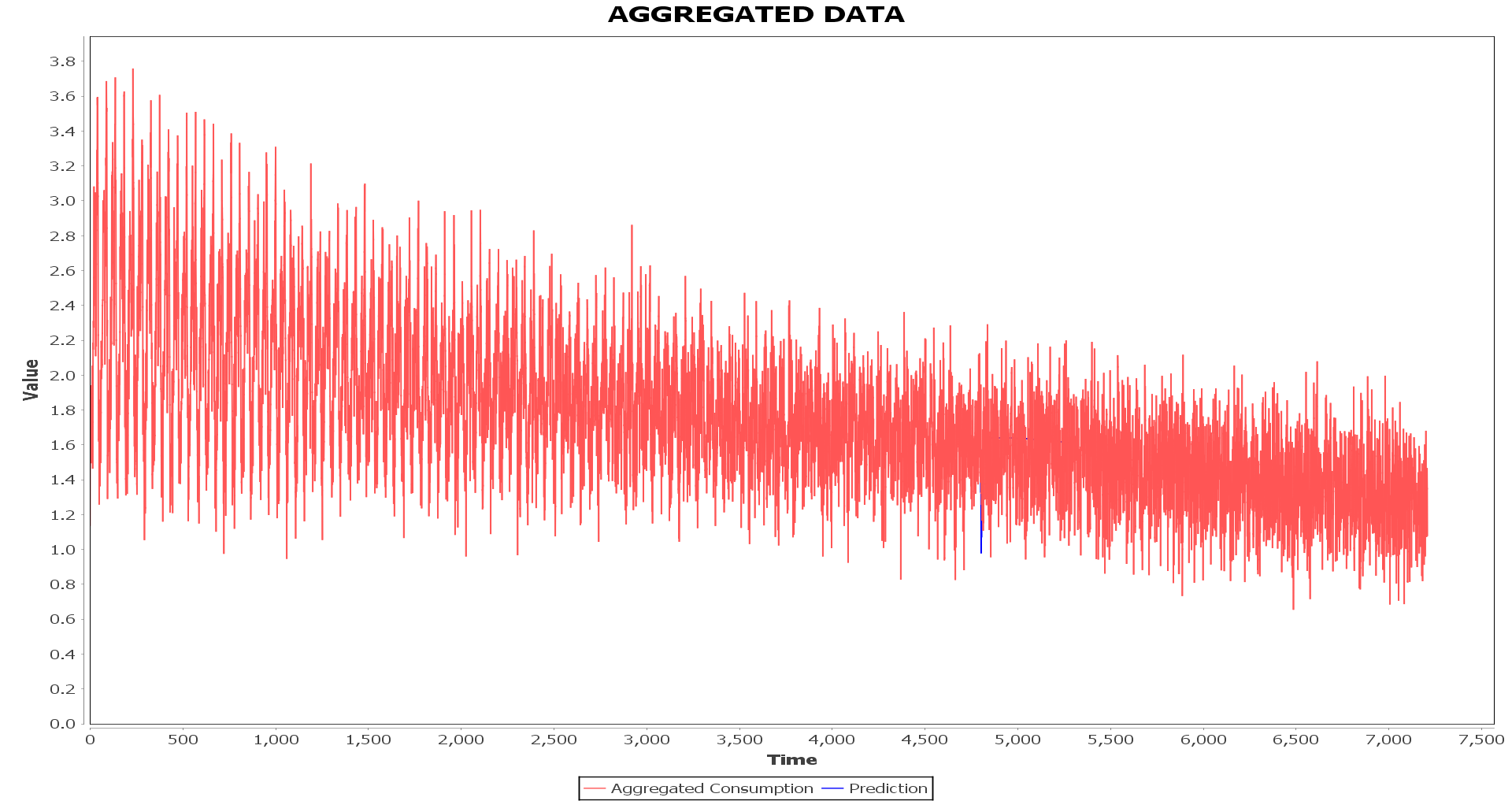
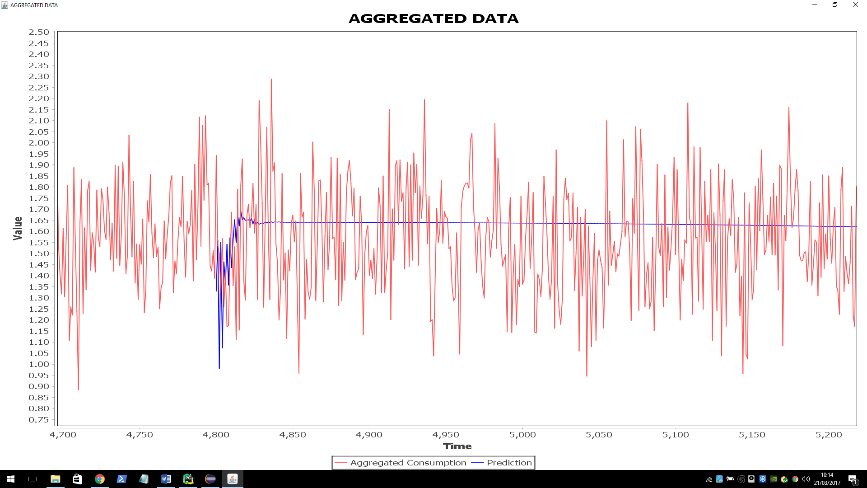
Given a null modifier, the Weka implementation of ANN performs quite well, a forecast of the next 10 days was given after training on the first 50 days.



**Figure 35.** Graph showing a forecast given by the MultilayerPerceptron Weka classifier.

### **8.3.2 Magnitude Modifier**

When modifiers are introduced, the forecaster performs very poorly. The total error remains at roughly the same during training, it fluctuates between 0.05 and 0.06.

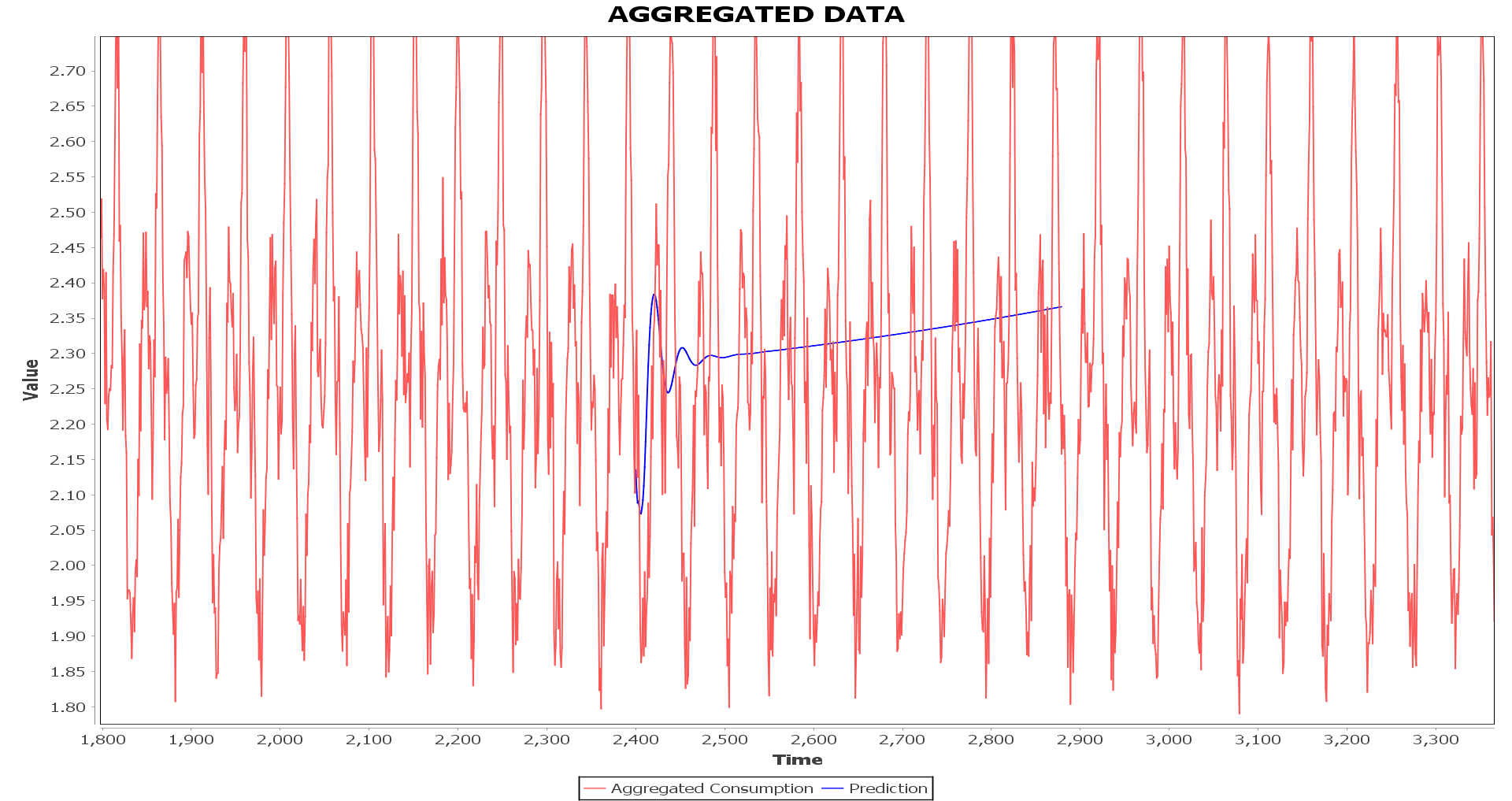


**Figure 36.** Graph showing a forecast given by the MultilayerPerceptron Weka classifier, when a Magnitude modifier is used. The model was trained for 100 days, and forecasted the next 10 days. (3000 epochs, Learning rate = 0.5).

Some experimentation was done with different parameters but the above was the best found, others were wildly off the mark. Perhaps after more experimentation or after some serious pre-processing of data a better forecast could be found, however due to time constraints it will be left as is for now.

### **8.3.3 Conclusions Regarding forecasting**

There is still a lot of work to do to find a suitable forecasting algorithm for this data, to do this is a project in itself. The purpose of this section was to demonstrate that the architecture is in place to do just that – algorithms can be tested using the simulation system. The SMOreg Weka classifier was also tested (on its default settings) see *Figure 37.* However this performed much worse than the multilayer perceptron classifier with a null modifier, no further experimentation with this classifier was done due to time constraints. Finding more suitable algorithms and optimizing forecasting will be part of future work. It is known that SVM regression is very sensitive to the user defined parameters, in particular to the kernel function. It is likely that an attempt at optimizing the parameters will give a much better result.



**Figure 37.** Result of testing the SMOreg Weka classifier on its default settings (null modifier). The result is very poor, however extensive testing was not done – only the polynomial kernel was used. The classifier was trained on 50 days of data and predicts for 10 days. A lot of work still needs to be done in finding a suitable parameters/algorithms for forecasting.

# **Chapter 9: User Guide**

## **9.1 Dependencies**

The project requires a number of libraries to run, some of these are in the form of jars (for the simulation) others are for the supplementary programs written in R and Python.

**Java,** all of the following jar files are required for the system to run properly, they have all been provided in the *jars* folder, otherwise here are the jars with their respective descriptions and download links. The project was developed using Java 8, it is required to run the system!

* **Weka**
  + Weka.jar latest version. Can be found by downloading Weka <http://www.cs.waikato.ac.nz/ml/weka/downloading.html> and searching for the jars, or by direct download from <http://www.java2s.com/Code/Jar/w/Downloadweka370jar.htm>.
  + *mtj.jar v1.0 or higher.* A java matrix library (that should be included in the Weka jar but is not for some reason) available: <https://github.com/fommil/matrix-toolkits-java>
  + *TimeseriesForecasting1.0.jar* or higher. The Weka library for time series prediction, it should be downloaded via the Weka package manager.
* **JFreeChart**
  + *Jfreechart-1.0.19.jar* or higher.Available: <http://www.jfree.org/jfreechart/download.html> the following dependencies are included with the jfreechart download.
    - *Commons-codec-1.3.jar*
    - *JCommon-1.0.14.jar*
* **GAWL**
  + gawl.jar. This jar cannot be found online (yet) and does not have an official version, it will be provided in the jar folder. The system is strongly dependant on this jar and as it is under development any other version will likely not be compatible.

**Python,** there a number of python programs that have been used to generate graphs for the reports as well as the most notable; the basic implementation of an ANN. These programs were developed using *Python 3.5* and require the following packages: *numpy* and *matplotlib*. These can be found online set up in a python environment, they often come with math/science python packages.

**R,** there are also a number of R programs used in data analysis (ARMA) and creating graphs. All that is required to run these is an R interpreter, RStudio (recommended, available at <https://www.rstudio.com/products/rstudio/download/>) or any other equivalent software is suitable. Any packages required for a program can be downloaded via the package manager.

## **9.2 Running**

As the system is still in development the easiest way to run is to import it into an IDE -Eclipse. Some parameters must be changed in the source code, namely the forecasting algorithm to use. All of the dependencies for the project are explained in the above section. The following sections will explain in detail how to run the system, as well as the complementary programs. Most of the entry points for the project reside in the *demo* package.

### **9.2.1 Simulation**

The main simulation system can be run from the *jars/runnables/SmartGridSimulator.jar* file (Java 8 must be installed on the system), using the command *java –jar \*/SmartGridSimulator.jar*. To change the structure of the agent hierarchy modify the *jars/runnables/config.txt,* to see an example of this see *jars/runnables/exampleconfig.txt*. It is recommended that the entire project is imported into Eclipse, this way many more parameters can be altered (for example the forecasting algorithm – altered in the *demo/simulator* source file). Most of the parameters that can (and should) be altered reside in the *demo/simulator* source file. To run from Eclipse, use *demo/Main* as the entry point.

All test code resides in the *test* package, it may be run directly but requires the *JUnit4.0* library. (Some of the test code is outdated and may fail)

### **9.2.2 Data Fitter**

After importing the project into eclipse, the *DataFitter* may be run. This will show the graphs given section 5.3. To alter the fit, values in the *HouseModelCombinedNormal(Affluent, AcornU, …)* classes must be modified. See the documentation on *utilities/NormalDistribution* and *utilities/CombinedNormalDistribution* classes for details on values that may be changed.

### **9.2.3 Machine Learning**

**Fast Data Generation:** The system may be used to generate data without the use of the agent system. This method of generation is much faster. It can be done using the *demo/ExperimentDataGenerator* class. This class will write data to a file (defaults to *“RandomHouses.csv”*), the output file may be changed by changing the *FILEPATH* variable in the source code. If the output file already exists, the system will ask if it should be overwritten, answer with yes or no.

**Fast Algorithm Testing:** Forecasting algorithms can also be tested directly, this should be done using the *demo/MachineLearningMain* class. Currently only the Weka *MultilayerPerceptron* is set up to run. To set up another algorithm, the user should extend *AbstractLearningTestCase*. See *LearningTestCaseMultilayerPerceptron* for an example. This test case should then be added to the *LOGMAP* inside *MachineLearningMain,* this will allow the results of the training and forecasts to be written to a log file (which resides in the *logs/algotest* directory). The user must also provide a method to generate the test cases, a way of generating the parameters for the algorithm automatically. The data used for testing comes from the above mentioned data generator, by default it uses *“RandomHouses.csv”* as the training/test data. If this file cannot be found it will invoke the data generator and create a new set of data. The data file path may be changed by changing the *DATAPATH* parameter in the source of *MachineLearningMain*.[[6]](#footnote-6)

### **9.2.4 R**

All R Scripts can be found in their respective folders in the *RScripts* folder. There are various R scripts that are complementary to the project, the one of most interest resides in the *ARMA* folder, namely *Analysis.R*. This script uses some R packages (*TTR* and *forecast*), which can be downloaded using an R package manager, to apply ARMA (as described in section 6.2). It can be run using any R Interpreter (recommended RStudio). There are various other scripts (*Analysis* folder) that were used for the initial data analysis (section 5.2), the data files have not been included as they are very large and so these scripts will not run.

### **9.2.5 Python**

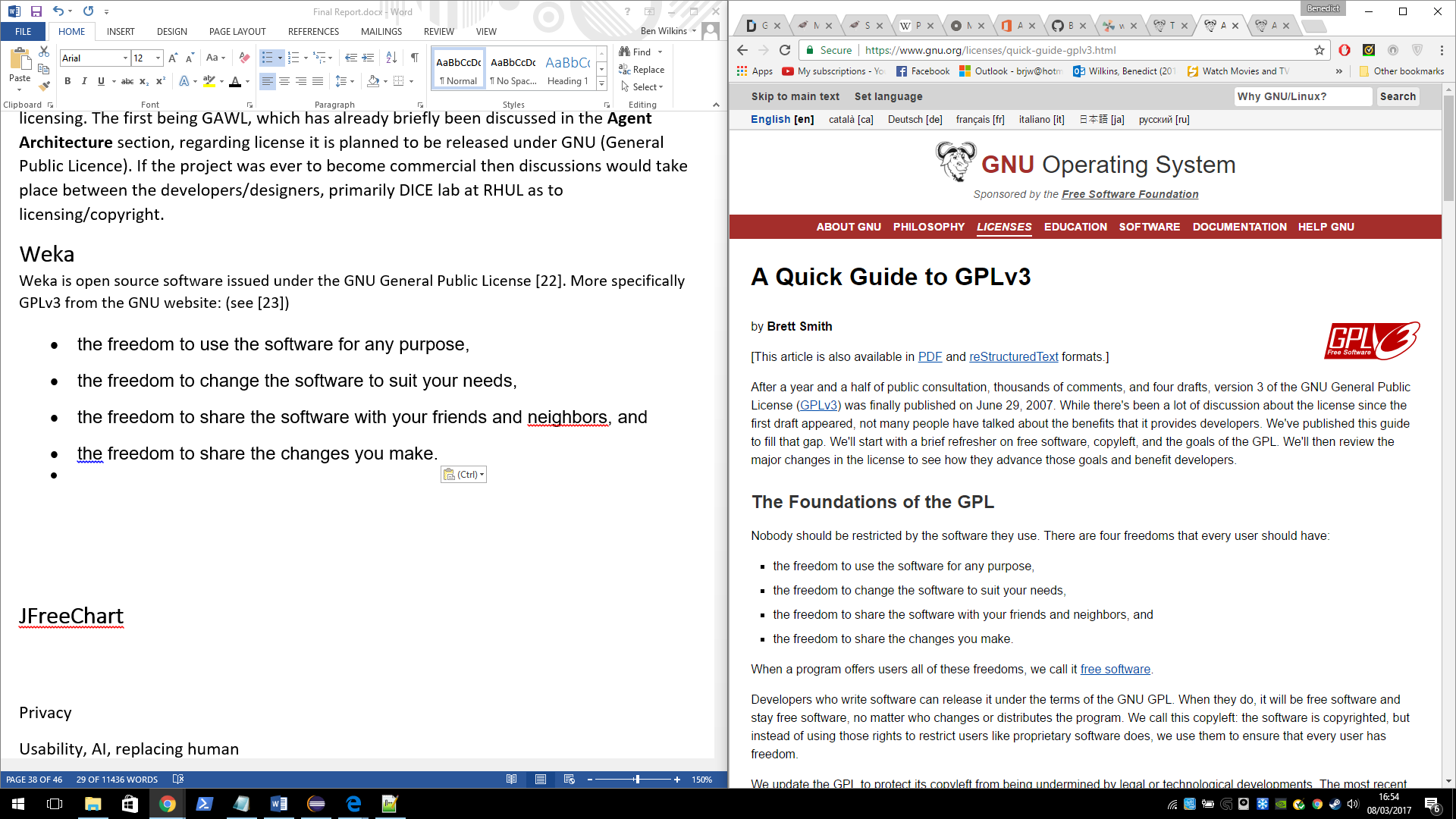
All python files can be found in the *Python* folder. The most notable file is *ANNtest.py* this file contains the code for the basic ANN used for the example given in section 6.5.1 See section 9.1 the required python libraries. These files can be compiled and run by a Python 3.5 interpreter. The other Python files contain code for generating the various graphs used in the report (each of which states this).

# **Chapter 10: Professional Issues**

## **10.1 Licensing**

In this project the issue of licensing becomes apparent on inspection of the libraries used. There are three libraries used in this project, each will be discussed in the context of licensing. The first being GAWL, which has already briefly been discussed in section 3.5, regarding license it is planned to be released under GNU (General Public Licence). If the project was ever to become commercial then discussions would take place between the developers/designers, primarily DICE lab at RHUL as to licensing/copyright.

### **10.1.1 Weka**

Weka is open source software issued under the GNU General Public License [21]. More specifically GPLv3. (from the GNU website, see [22])

In short, this project is free to use Weka in any way as long as proper credit is given to the original author and when this software becomes available it remains under GPL. This means that all code for the project would need be publicly available, essentially the same licensing rules that apply to Weka would also apply to this project. Regarding public release of the project, it is likely that if it were to ever arise Weka will have been phased out. The algorithms provided by Weka primarily examples used in demonstrating the capabilities of the system. At full system release the goal is to have specialised machine learning algorithms that adapt accordingly in real time, these will not be related to the Weka library and may even be implemented in a language other than Java.

### **10.1.2 JFreeChart**

JFreeChart is given under the GNU Lesser General Public License. LGPL is only slightly different to GPL, namely, any modifications to the LGPL library code must be available, however linked proprietary code does not have to be available i.e. the sections of the project unrelated to JFreeChart do not have to be made available [23]. JFreeChart in this project is being used as a visualisation tool, again it is not at the core of the project may not be included in a publicly available version. After removing the licensed libraries, an appropriate licenses for the project (depending on its outcome, commercial or otherwise) may be selected without concern for inherited licenses.

## **10.2 Usability**

The main topic in usability in relation to this project is the role that the system will play in social economy, namely what jobs it will perform and how that may affect the livelihoods of working people. In a multitude cases through history machines have replaced humans in jobs that are manual and repetitive for example on productions lines or in field working. It is only recently that machines are becoming more capable and beginning to replace humans in jobs that require more thought. The simple reason is that they are more cost effective, they don’t require rest, holidays, salary (other than maintenance) and don’t strike. However, there is a problem that must be addressed; the people who have lost their jobs.

The topic of technological unemployment is an interesting and heavily debated one, there have been quite a number of points of view of the problem ranging from denial of its existence, to a complete restructure of the economy or political system [24]. Looking at history, and in particular the case of the Luddites; who were a group of English textile workers that destroyed the machines that were taking their livelihoods. We introduce the Luddite fallacy, as *“the simple observation that new technology does not lead to higher overall unemployment in the economy. New technology doesn’t destroy jobs – it only changes the composition of jobs in the economy.”* [25]. Whether this will apply to the AI ‘revolution’ is again up for debate [26]. Some refer to it as the Technological Singularity, the idea that at some point in the future AI will overtake humanity. A great discussion of this can be found [27] however this is moving beyond our scope.

## **10.3 Safety and Reliability**

Any business deciding to utilise a system needs some guarantees that it will work. A formal mathematical analysis of the system is ideal, but not always possible especially for large complex systems. It is best to look at areas of the system whose requirements of particular importance which will vary from system to system but is usually related to some legal or financial point. To take an example of this system, energy companies expect payment for the service they provide - the supply of energy to consumers. These payments are derived from consumer’s energy usage which is traditionally acquired by a human employee taking a reading from a meter at the consumer’s premises. The company puts some trust in the employee to take a correct reading and report back accurately. This system must do just that, some guarantee that the system will respect the interests of the company that uses it should be made.

# **Chapter 11: Evaluation and Review**

## **11.1 Term One Review**

I have not followed the original project plan, especially the sections related to ANNs. I was too ambitious with the time frames that I designated for each report/program, it would have been wiser to move the machine learning related tasks to the second term. Now knowing that an ANN based model may not be the way to go, it is fortunate that I had not already implemented the relevant proof of concept programs/written relevant reports. Most of the machine learning related work will be done next term. I have however had the opportunity to do some fairly extensive background reading on ANNs for time series prediction and machine learning in general [28] [29] [30] [31] [32] [33] which will give me a very good base for next term. I have progressed quite far with the simulation system. Data generation is complete, the multi-agent system is almost complete all that needs to be done is to create some integrated machine learning capability in an agent mind.

## **11.2 Term Two Review**

I am happy with how the second term went even though there is still a lot to do in the machine learning section. The project was ambitious with the amount of work required to fully complete it. I have not implemented any machine learning algorithms that can be used in the system however I did complete the proof of concept ANN program (which was in the original plan). The simulation system has been completed – only the combined normal modes should be changed to be sinusoidal in the future (section 8.2.2). The second term was better planned than the first term, it was easier to see what needed to be done after the agent system was built. I decided that having a complete simulation system was more important than developing suitable forecasting algorithms and that using Weka for demonstrative purposes was acceptable.

## **11.3 Project Review**

The following goals, reports and programs were taken from the original project plan but also include goals that developed as the project progressed.

### **11.3.1 Project goals**

1. To develop a frame work in which energy consumption data may be read, communicated and evaluated. **Complete.** (All of the above can be done and more – the feedback loop, modifying behaviours, forecasting with a library etc.)
2. To develop an agent based simulation system for the above. **Complete.** (The agent simulation is complete, the agent hierarchy was implemented, house environments, different types of agents with different goals etc.)
3. To develop a data generation system that models real energy consumption. **Complete.** (After the sinusoidal alteration to the models)
4. To use machine learning (ANNs) to forecast energy consumption. **Partially Complete.** (This has been done using the Weka library, however it would be nice to future optimise the forecasts and perhaps to developed tailored algorithms for the problem).

Most of the goals of the project were satisfied, I think the project as a whole has been very successful. That the system does what was intended in the beginning, and does it well.

### **11.3.2 Original Reports**

1. Smart Meters and Energy Demand Management. **Complete.** (See Chapter 2)
2. Introduction to Multi-agent Systems. **Complete.** (See Chapter 3)
3. Introduction to Artificial Neural Network. **Complete.** (See section 6.5.1, it includes a derivation of back propagation and explains the concepts as the plan suggests)
4. Artificial Neural Networks for Time Series Prediction. **Incomplete.** (Due to time constraints a detailed discussion of ANNs for time series forecasting was not given. However, other forecasting algorithms were discussed namely; ARMA and SVM, I think this is sufficient especially after the developments in the plan for the second term to justify the absence of this report)

Given that the only report that wasn’t completed was 4 and that the projects goals changed after the first term to focus on time series forecasting in general (and in the projects context). In the end most of the effort in report writing went into describing the system and the processes involved with building it rather than machine learning theory (however there is some detailed theory in section 6.5. I believe this was justified as explaining the project and documenting it was more useful to the reader (background theory can always be found else ware).

### **11.3.3 Original Proof of Concept Programs**

1. A basic Multi-agent System. **Complete.** (Although a separate MAS was not built, the simulation system accomplishes everything that this program would have, I decided early in the first term that it would be better for time management to build the simulation system instead).
2. A basic ANN. **Complete.** (This was not completed until the end of the second term, it was implemented in Python using numpy – one of technologies discussed early in the project. I felt that it was more important to get everything working and use a library instead of implementing it myself earlier on. However just for demonstrative (it was used in section 6.5.1 as an example) purposes this was implemented anyway, even it is quite rudimentary)
3. ANN model. **Incomplete.** (For the same reasons as 4 in the previous section, this was not implemented. It was decided after the first term that libraries would be used instead to improve time management)
4. Data generation System. **Complete.** (See Chapter 5), the system generates *realistic* data as required in the original plan)
5. Time series Data Pre-Processor. **Complete.** (The data can be combined by intermediate agents as well as predictors, missing data can also be handled (even if it is hard coded currently. Normalisation/standardisation is handled by the Weka algorithms and so this sort of pre-processing was not necessary, however it could easily be implemented in an agent).

The next section shows the development of the plan over the duration project. The changes have been reflected in this section.

## **11.4 Project Diary**

I had meetings with my supervisor – Dr Zhiyuan Lou and occasionally Prof Kostas Stathis every fortnight. The meetings were very useful in evaluating the progress of the project. Because of these fortnightly meetings I tended to work around meetings every two weeks, this is why my revision control (git repository) sees fortnightly commits. The meetings are as noted below:

### **11.4.1 Term One**

**12th October:** General discussion about the project and how to proceed. Discussed the multi-agent architecture to be used and what reports I will be writing this term.

**26th October:** Kostas was involved with discussions on the project. Talking about the different layers of the architecture – including where in the agent system the predictors should reside. Talked about having area/neighbourhood agents having prediction capabilities for the houses they are responsible for. Discussed moving away from the ANN model in favour of a simpler model. This may be the way to go as ANN can be difficult to use/explain. I have put the multi-agent report on hold to work on the data generation section of the project. We agreed that this section was more appropriate as a means to continue the project effectively as the data will be relied upon when starting the prediction section. A good portion of the Multi-agent system has been completed by this point, agent communication via sockets, house environments with (after data generation section has been completed – will be implemented) the capability of holding and retrieving data from a generic data generator. The presentation at the end of term should have a demonstration graphic of the prediction working – e.g. a graph with 2 lines, one for real data and one for the prediction. The lines will extend across the graph with time.

**9th November:** Kostas was involved in the meeting again. I briefly demonstrated the multi-agent system to Zhiyuan and Kostas as well as the data generation program and report. We discussed the architecture of the House Environment, Kostas suggested that a data reading agent should be used to get readings from the generator and forward them to the Smart Meter Agent. This method seems better than the current one – where the smart meter agent reads at a clock tick on a global timer. We again spoke about the ANN implementation and after looking at the data analysis section of the Data Generation report decided that testing different prediction models would be a good addition to the project. Having this meeting allowed me to finish the Introduction to Multi-Agent Systems report, specifically the sections about the architecture used in the project.

**23rd November:** Final meeting this term. With Zhiyuan only, we spoke about the presentation, interim viva and how to proceed with the project. He advocated that I had at least some machine learning work to present for the interim report. I will now work on some basic forecasting on generated data. The forecasting will be done using prebuilt time series machine learning packages (probably in R). At least some experimentation will be complete by the time the report is due, the rest will be done over the Christmas break. I will be implementing the most successful machine learning methods next term.

### **11.4.2 Term Two**

**17th January 2017:** Meeting with Zhiyuan, We discussed the progression of the project, work over the Christmas break, and where to go next – what are the plans for this term. We agreed this this term should focus on integrating machine learning into the project. The agent section of the project is almost complete, it only requires integration of machine learning and agent minds (for the learning agents). We decided that Weka was the library of choice, over the next weeks I will spend time getting to grips with Weka and slowly integrate it into the project.

**31st January 2017:** Second meeting of the term with Zhiyuan. We discussed progress with Weka, I showed an example of the MultilayerPerceptron forecasting model training and forecasting on some auto generated data – it did not do very well. We concluded that more work was needed to tweak and test different algorithms. I am now working on an automated algorithm tester (some meta-learning!) to try and find the best algorithm and parameters for the job. I had previously been using the built in Weka forecasting algorithms on their default settings.

**15th February 2017:** Midweek meeting with Zhiyuan and Kostas, I presented my work so far to them, which included drawing an outline of the structure of the project and showing my progress up to this point. We discussed where the project should go from there – that the feedback look should be implemented, this involves some kind of thresholding mechanism that a top level agent may use in order to request a change in behaviour of the underlying consumption models. The models should be able to be modified in a number of ways: shifting, magnitude etc. Once this has been implemented the next step will be to build a (basic) visualisation of the system and to test prediction models supplied to the PredictorAgent. I have no made much progress on the report at this point but it is important to finish the code as then I will have a complete project (and it will be easier to write about).

**1st March 2017:** Meeting with Zhiyuan. I demonstrated that the work to be done that was discussed in the last meeting had been completed, namely, the feedback loop, thresholds, and a way to alter the behaviour of consumers using ‘modifiers’. We discussed where to move next, Zhiyuan suggested that I present some interesting scenarios in my final report and that I should focus on writing about what the project has achieved so far. It should be noted here that of course the complete goals of the project have no be met – the machine learning/forecasting is lacking to say the least. It will however, likely be pursued in future work as it is a question of time not capability.

**2nd March 2017:** Final meeting before the project deadline, we spoke about the structure of the final report, how it should flow properly and as the project is quite complex, it should introduce and explain each concept in a readable way. Zhiyuan suggested adding pseudo-code in the sections that look like a long description to make it easier to follow. I will have the report finished on time!

## **11.5 Future Work**

* **Machine learning:** The main bulk of future work involves machine learning, to continue testing and optimizing algorithms in the Weka library and then to go on to devise and develop a purpose built forecasting model.
* **Centralised System parameters:** Currently to change some aspects of the system, source code must be changed – for example to change the forecasting algorithm. It would be good to have a configuration file in which parameters can be easily set.
* **Command line runnable:** Although the system can be run from command line with a config file, the config file does not have all of the desired options. It would be nice to allow a more complex set up from the config file e.g. specifying the forecasting algorithm etc.
* **GUI:** Better than a configuration file would be to have a GUI, creating the agent hierarchy in a file would be difficult especially for large simulations, to construct it in a GUI would be much easier. The GUI could also be used to visualise and monitor the agent system and forecasting algorithm during runtime as well as change/tune other parameters.
* **Debugging:** Some other work involves debug the current system: the prime example is the occasion failure of data reaching the predictor agent mentioned in section *7.2*. Another example of a simple bug is data time handling, the system fails (non-fatally) if it encounters some dates e.g. for leap years
* **Missing Data:** It would be nice if the method for dealing with missing data could be altered easily, currently it is hardcoded. This could be done in a similar fashion to the threshold/modifiers (have modules (classes) that handle missing data than can be plugged/unplugged into an agent).

# **Bibliography**

|  |  |
| --- | --- |
| [1] | C. Gaven, “Seasonal variations in electricity demand,” Department of Energy & Climate Change, 2014. |
| [2] | European Nuclear Society, “Base load power plants,” [Online]. Available: https://www.euronuclear.org/info/encyclopedia/baseloadpowerplants.htm. |
| [3] | S. R. E. M. F. F. S. J. A. Emma L. Giles, “The Effectiveness of Financial Incentives for Health Behaviour Change: Systemtic Review and Meta-Analysis,” 2014. |
| [4] | C. D. A. L. S. P. A. M. Antonios Chrysopoulos, “Agent-based small-scale energy consumer models for energy portfolio management,” *2013 IEEE/WIC/ACM International Joint Conferences on (Vol. 2, pp. 94-101). IEEE.,* 2013. |
| [5] | M. Wooldridge, An introduction to multiagent systems, John Wiley & Sons, 2009. |
| [6] | S. Russell and P. Norvig, Artifical Intelligence. A modern apprach, Prentice Hall, 1995. |
| [7] | P. J. '. Hoen, K. Tuyls, L. Panait, S. Luke and J. A. La Poutré, “An Overview of Cooperative and Competitive Multiagent Learning,” *Lecture Notes Computer Science,* vol. 3898, pp. 1-46, 2006. |
| [8] | I. Rahwan, R. Kowalczyk and H. H. Pham, “Intelligent Agents for Automated One-to-Many e-Commerce Negotiation,” *Autralian Computer Science Communications,* vol. 24, no. 1, pp. 197-204, 2002. |
| [9] | C. Yilmaz, S. Albayrak and M. Lutzenberger, “Smart Grid Architectures and the Multi-Agent System Paradigm,” in *Proceedings of the 4th International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies.*, Chamonix, 2014. |
| [10] | A. Rogers, S. D. Ramchurn and N. R. Jennings, “Delivering the Smart Grid: Challenges for Autonomous Agents and Multi-Agent Systems Research,” in *Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*, Toronto, 2012. |
| [11] | M. S. Narkhede, S. Chatterji and S. Ghosh, “Multi-Agent Systems (MAS) controlled Smart Grid - A Review,” in *Proceedings on International Conference on Recent Trends in Engineering and Technology*, 2013. |
| [12] | B. Alrayes, O. Kafali and K. Stathis, “RECON: A Robust Multi-agent Environment for Simulating COncurrent Negotiations,” *Advances in Agent-base Complex Automated Negotiation,* pp. 157-174, 2016. |
| [13] | S. Bromuri and K. Stathis, “Situating Cognitive Agents in GOLEM,” *Lecture Notes in Computer Science,* vol. 5049, pp. 115-134. |
| [14] | A. Kakas, P. Mancarella, F. Sadri, K. Stathis and F. Toni, “Computational Logic Foundations of KGP Agents,” *Journal of Artificial Intelligence Research 33 (2008) 285-348,* 2008. |
| [15] | J. Hopkins, O. Kafali and K. Stathis, “Open Game Tournaments in STARLITE.,” in *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, 2015. |
| [16] | K. Stathis and F. Toni, “The KGP model of agency for decision making in e-negotiation,” 2005. |
| [17] | Department of Energy & Climate Change, “Smart Metering Implementation Programme, Smart Metering Equipment Technical Specifications,” Department of Energy and Climate Change, 2014. |
| [18] | The University of Waikato, *Weka MultilayerPerceptron Documentation.* |
| [19] | “Support Vector Machine Regression,” [Online]. Available: http://kernelsvm.tripod.com/. |
| [20] | MathWorks, “Understanding Support Vector Machine Regression,” [Online]. Available: https://uk.mathworks.com/help/stats/understanding-support-vector-machine-regression.html. |
| [21] | University of Waikato, *Weka Licensing.* |
| [22] | GNU, *GPLv3.* |
| [23] | P. Upfold, “The differences between the GPL, LGPL and the BSD,” 6 April 2007. [Online]. Available: https://fosswire.com/post/2007/04/the-differences-between-the-gpl-lgpl-and-the-bsd/. |
| [24] | J. Perry, “Ten Responses to the Technological Unemployment Problem,” 27 April 2013. [Online]. Available: http://declineofscarcity.com/?p=2790. |
| [25] | T. Pettinger, “The Luddite Fallacy,” 15 Janurary 2016. [Online]. Available: http://www.economicshelp.org/blog/6717/economics/the-luddite-fallacy/. |
| [26] | V. Wadhwa, “Sorry, but the jobless future isn’t a luddite fallacy,” 7 July 2015. [Online]. Available: https://www.washingtonpost.com/news/innovations/wp/2015/07/07/sorry-but-the-jobless-future-isnt-a-luddite-fallacy/?utm\_term=.44c6bbc11a09. |
| [27] | M. Shanahan, The Technological Singularity, MIT Press. |
| [28] | H. Allende, C. Moraga and R. Salas, “Artifical neural networks in time series forcasting: A comparative analysis,” *Kybernetika,* vol. 38, pp. 685 - 707, 2002. |
| [29] | MachineLearner, “CS231n Lecture 10 - Recurrent Neural Networks, Image Captioning, LSTM,” 14 June 2016. [Online]. Available: https://www.youtube.com/watch?v=iX5V1WpxxkY. |
| [30] | B. Oancea and S. C. Ciucu, “Time series forecasting using neural networks,” *arXiv,* p. 1, 2014. |
| [31] | S. Miller, “Mind: How to Build a Neural Network (Part One),” 10 August 2015. [Online]. Available: http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/. |
| [32] | T. M. Mitchell, Machine Learning, MIT Press, The McGraw-Hill Companies Inc, 1997. |
| [33] | PyData, “Alex Rubinsteyn: Python Libraries for Deep Learning with Sequences,” 4 December 2015. [Online]. Available: https://www.youtube.com/watch?v=E92jDCmJNek. |
| [34] | Smart Energy GB, “About the rollout,” [Online]. Available: https://www.smartenergygb.org/en/the-bigger-picture/about-the-rollout. |
| [35] | nelsonminar, “Energy usage notes,” 31 Janurary 2012. [Online]. Available: https://nelsonslog.wordpress.com/2012/01/31/energy-usage-notes/. |
| [36] | B. Davito, T. Humayun and U. Robert, “The smart grid and the promise of demand-side management,” McKinsey and Company, 2009. |
| [37] | British Gas, “Smart Meters,” 2016. [Online]. Available: https://www.britishgas.co.uk/smarter-living/control-energy/smart-meters.html. |
| [38] | “BBC News,” [Online]. Available: http://news.bbc.co.uk/1/shared/spl/hi/guides/456900/456991/html/. |
| [39] | M. Mazur, “A step by step Backpropagation Example,” [Online]. Available: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/. |
| [40] | “Chapter 1: Fundamental Concepts of Time-Series Econometrics,” pp. 1-4. |
| [41] | A. University, “Time Series Forecasting Theory | AR, MA, ARMA, ARIMA | Data Science,” Analytics University, 6 Febuary 2016. [Online]. Available: https://www.youtube.com/watch?v=Aw77aMLj9uM&t=3s. |
| [42] | MIT OpenCourseWare, “16. Learing: Support Vector Machines,” 10 January 2010. [Online]. Available: https://www.youtube.com/watch?v=\_PwhiWxHK8o. |

# **Appendices**

## **Appendix 1. Project Plan MAS Structure**

The agents will be placed in a multi-layered environment, each layer will contain one or more agents and each agent in a layer may be a manager of some agents in the layer below illustrated in figure 1. The system will be developed to have arbitrary layers in this fashion with the top layer agent(s) as prediction agents, the bottom layer as data collection agents and intermediate data pre-processing agents. Depending on the amount of data pre-processing to be done it may be preferable to distribute the system in terms of layers or agents. The agents will have the capability to communicate via an internet protocol of choice. This can be done using the Starlite framework as it allows sensor/actuator modules to be attached to the agents, these modules can be set up to accommodate any protocol. The full details of the Starlite framework will be discussed in the Introduction to Multi-agent Systems report mentioned below.

A0

A11

A211

A12

…

A1n-1

A1n

A212

…

…

…

…

…

A21m

A2n1

A2n2

A2nm’

…

***Figure 38.*** *AL is an agent at layer L. A0 will typically be the prediction agent – in this case the one that uses the ANN. Agents directly below A0 will be supplying it with clean formatted data. The bottom layer of the hierarchy will be reserved for data collection – in this case the smart meter agents. The lines represent communication channels between the agents. These channels will be used to send the collected data up the hierarchy. The channels will be bi-directional as an agent may want to send control data to an agent it is managing. See figure 2 for an illustration of communication channels.*

A0

A11

Upper layer agent

Lower layer agent

Control Data

Collected Data

***Figure 39.*** *Control data is sent down the hierarchy and the collected data – in this case the smart meter data is sent up the hierarchy. Control data may be any data and will depend on the systems use. In this system it may be instructions for data pre-processing.*

## **Appendix 2. Additional Relevant Work**

1. \* The half hour interval will not be in real time, it will be scaled to simulation speed. [↑](#footnote-ref-1)
2. \* Note: GAWL was developed by Prof. Kostas Stathis, Emanuele Uliana and myself and is based upon work that Prof. Stathis (and colleagues) have worked on previously including GOLEM and STARLITE. [↑](#footnote-ref-2)
3. Reference to Software Engineering Methodologies [43] [44] [↑](#footnote-ref-3)
4. ARMA references [40] [41] (Note that 18, does not give book title, only chapter 1 is available) [↑](#footnote-ref-4)
5. Credit to [42] Patrick Winston and MIT OpenCourseWare for providing a great lecture with an explanation and derivation of SVM for classification. [↑](#footnote-ref-5)
6. The complete Javadoc for the above sections can be found in the *jars/javadoc* directory. [↑](#footnote-ref-6)