Dissertation

Mathematical and Experimental Study of Anaerobic Digestion Process and Biogas Generation from Fruit and Vegetable Waste

August, 2017

Puteri Kusuma Wardhani
Graduate School of
Environmental and Life Science
(Doctor's Course)
OKAYAMA UNIVERSITY

Abstract

Anaerobic digestion is a biochemical process in which microorganism break down the biodegradable material into biogas (methane and carbon dioxide) under the absence of oxygen. It has been used to treat many types of waste, not only due to energy production but also because it reduces pollutant. Since the complexity and operational stability of anaerobic digestion process may lead to the process failure, understanding of its process dynamics and operation optimizations become necessary. Mathematical modeling and simulations are essential tools for these purposes. Many mathematical models have been proposed and they are available in literature to obtain an optimal anaerobic digestion process. Nevertheless, the applicability of those models may be limited by complexity of anaerobic processes.

The general objective of the thesis is to develop mathematical models suitable to describe anaerobic digestion process of fruit and vegetable waste for process optimization. Scope of the thesis are as follows: Laboratory scale experiment to generate data of accumulated methane concentration and biogas volume obtained from fruit and vegetable waste as substrates and sludge from biogas plant as inoculum; Introduction of experimental outcomes into inverse problems of methane generation process; Numerical solution of the inverse problems and estimation kinetic parameters from proposed models; Simulation of a methane co-fermentation process with outcomes form the analysis.

Ordinary Differential Equation (ODE) model of anaerobic digestion processes of fruit and vegetable waste was constructed. The first-order model and Monod model were applied. Numerical results obtained were compared with experimental data, and showed a good agreement. However, the model incapable of describing anaerobic digestion under inhibitions. The kinetic parameters from proposed models were determined by solving the problems numerically. The method used was trial error method, however this method is time consuming.

The modification of model by Grau (1975) was proposed, and it was implemented in Matlab. Nonlinear least squares problem for determination of model parameters was solved with the nlinfit, in which the Lavenberg-Marquardt method is applied. The numerical results and experimental data were compared. The results lead to the conclusion that the Levenberg-Marquardt method demonstrated excellent performances for the nonlinear fitting task. Further, simulation results showed a good agreement with the experimental results.

ACKNOWLEDGMENT

I would like to convey my gratitude to the MEXT for granting scholarship for the completion of my doctoral studies. I also would like to thank the Okayama Prefectural Livestock Research Institute (Chikusan Kenkyuujo) for providing the inoculum for this study. This work was partly supported by JSPS KAKENHI Grant Number 16K05276.
My deepest gratitude goes to Prof. Watanabe Masaji for allowing me to carry out my PhD studies under his supervision. His great expertise and unconditional guidance were essential for the successful completion of my PhD. Special thanks go to Prof. Md Azhar Uddin from Lab of Material and Energy Science for his mentorship in the development of the anaerobic reactor, advice on experimental decisions, and management support. Special thanks also go to Prof. Kameshima Yoshikazu for his guidance in experimental analysis.
In a more personal level, I would like to thank my friends for the amazing experiences lived during the past years. My world was greatly broaden thanks to your friendship that kindly warmed my heart and for which I will always be very grateful. Finally and more deeply, I would like to thank my family for all their unconditional love and support. You taught me all the principles that guide my life and everything I am is because of you. Everything I have accomplished I owe it to you. Hence, this thesis is dedicated to you as it is more yours than mine.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Problem Statement

1.1 General Objective

Environmental pollution by solid waste is one of the major problems that human being face in the twenty-first century. The quantity and characteristic of solid waste arising from many activities are the results of growing population and increasing standards of living and technology, and abundance of natural resources [3]. Solid waste contributes significantly to the contamination of the environment by polluting the main elements (soil, water, and air) [2]. In the past, the amount of waste was small, therefore the environment was able to absorb it, causing just a few environmental problems. In modern times, solid waste has become one of the most crucial environmental problems. It poses a constant threats to humans and the environment, because some type of waste are toxic and unsafe [1, 4, 5].

Sustainability is the key to the solution to prevent or reduce those issues. One of the most effective and sustainable approach is to apply anaerobic biotechnology as an alternative [6]. This technology combines waste treatment with the retrieval of useful byproducts, and leads to valuable energy (hydrogen, butanol, and methane) and valuable products (biosolids, organic acids, etc.) [6]. It has been used to process many types of wastes, such as food waste, fruit and vegetable wastes, household waste, agricultural waste, and the organic compound of municipal solid wastes [5].

Anaerobic digestion is a biochemical process in which microorganisms break down the biodegradable material into biogas (methane and carbon dioxide) anaerobically [7]. The process consists of several stages. Firstly, complex organic compounds are hydrolyzed to simpler organic compounds. Secondly, they are fermented to volatile fatty acids by acidogen bacteria. Those microbes are generally both facultative and obligate anaerobic bacteria, which become the most essential factor in the anaerobic process. The stability of the process depends on the environmental conditions, such as pH, temperature, and dissolved oxygen. The latter stage is methane and carbon dioxide formation from fatty acids by methane-generate bacteria. The entire process results in the reduction of organic substance in the waste [1].

Despite many benefits of anaerobic digestion process, the process is burdened by complexity, poor practical and operational stability, high sensitivity to changes of environmental conditions, long retention and start-up times, and highly polluted supernatant, which preclude this technology from being used optimally [5]. Many researchers have investigated various potential methods to solve the problems mentioned, particularly in anaerobic di-

gestion process of fruit and vegetable waste (H. Bouallagui *et al.*, 2003 [8]; H. Bouallagui *et al.*, 2004a [9]; 2004b [10]; H. Bouallagui *et al.*, 2005 [11]; H. Bouallagui *et al.*, 2009 [12]). Those studies focused on the optimal conditions for biogas conversion from fruit and vegetable waste, and developed the high rate anaerobic reactors for digester performance.

1.2 Prior Studies

Anaerobic digestion is the most complex process which has substantial chance for it to become unstable and eventually break down. It is important to understand the kinetics of anaerobic digestion process, which enables us to predict the operation of digester. It can also contribute to the understanding of the biodegradation process [1]. Mathematical modeling and simulation can be an excellent tool for these purposes.
In general, mathematical model can be categorized into two groups of types; dynamic or non-dynamic and white-box, grey-box or black-box model. Dynamic models are capable of making continuous predictions in time, while the non-dynamic models only predict one time-independent variable. Dynamic models consist of several ordinary differential equations (ODE's), based on mass balance principles. Non-dynamic white-box models link up substrate to products on stoichiometric grounds. The difference between white-box, grey-box and black-box models is the amount of a prior selective data. White-box models use a prior selective data for description of the biochemical reactions occurring during digestion process. Black-box models empirically relate the input directly to the output without including any prior knowledge of the physical and chemical reactions. Grey-box models are those in which the parameters have a physical representation that is adjustable, for example, a parameter estimation procedure [13].
A general application of a mathematical model consists of six steps. The first step is model selection. A choice has to be made between data-driven or mechanistically influenced models. The second step is parameter selection for calibration. This selection should be based on an evaluation of the identifiability of the specific parameters. The third step is data collection *i.e.* experimental measurements. The fourth step is parameter estimation. Various cost functions or objective functions have been used for this step, such as least squares, least-modulus, or maximum likelihoods. Also, a large number of minimization algorithms have been applied: the Gauss-Newton method, the steepest descent method, the Levenberg-Marquardt method (combination of Gauss-Newton method and steepest descent method), and genetic algorithms. The fifth step is accuracy estimation, where the uncertainty is determined. Confidence intervals for the estimated parameters can be developed. If the estimated uncertainty is too large, additional data should be introduced. At the final step, outcomes should be subjected to a validation procedure the calibration and individually obtained data. A visual review, in which the tendency of the predictions is compared with the measurements, can also be helpful [13].

1.2.1 Modeling of Anaerobic Digestion Processes

Several mathematical models of anaerobic digestion process have been developed. The first modeling was motivated by the demand for efficient operation of processes in the 1940's (Monod, 1949[14]; Contois, 1959[15]; Chen-Hashimoto, 1978[16]; Hill and Barth, 1977[17]; Hill, 1982[18]). Monod suggested the non-linear relation between specific growth

rate and limited substrate concentration. The specific growth rate increases strongly for low substrate concentration and slowly for high substrate concentration, until a saturation of bacteria is reached. Contois proposed that the specific growth rate was considered as a function of the growth-limiting nutrient in both input and effluent substrate concentration. On this groundwork, Chen-Hashimoto developed kinetic models for substrate utilization and methane production. They suggested that the Contois model would be more suitable than the Monod model to predict digester performance. Hill and Barth (1977) considered anaerobic digestion process as a multi-step process where one slower step controls the global rate. Nonetheless this model was unable to adequately depict the process performance, particularly under transeunt conditions. Hill (1982) considered the concentration of volatile fatty acids as the key parameter, incorporating acidogenesis and acetogenesis into modeling.
Further studies led to a comprehensive model detailing the production and degradation pathways of intermediates (Angelidaki *et al.*, 1993[19], 1999[20]; Siegrist *et al.*, 1993[21]; Vavilin *et al.*, 1994[23], 1995[24]; Batstone *et al.*, 2000[25]). Those models involved additional processes and species, to detailed kinetics with inhibition and many kind of substrates. Nevertheless, practical applications have been limited by diversity and complexity of the models [26].
The Anaerobic Digestion Model No.1 (ADM1), was recently introduced by the IWA Task Group for Mathematical Modeling of Anaerobic Digestion Processes for the reason described above. It formulates dynamics of 24 species and involves 19 bioconversion processes. However, large number of parameters and difficulties in their identifiability were the major drawbacks of ADM1. Also, despite the adequate representation of relevant physical processes by the ADM1 model, many reactions can occur quickly without the overall process dynamic [27].

1.2.2 Modeling Anaerobic Digestion of Fruit and Vegetable Waste

Only few studies have been reported modeling and simulation of anaerobic digestion process of fruit and vegetable waste. Mata Alvarez *et al.* (1993) [28] reviewed several models for methanization of a batch two-phase anaerobic digestion of fruit and vegetable wastes. Three selected and tested models in the investigation were the first-order model, the Monod model, and the Chen-Hashimoto model. Sosnowski *et al.* (2008) developed a mathematical model based on the first-order model and the Monod model for kinetic investigations of co-fermentation processes [29]. A simplified model of the first-order was developed by Siles *et al.* (2008)[30] for studying the anaerobic digestion of wastewater derived from the orange rind as by-product of orange juice production. Mathematical models based on the Monod model was also developed by Vavilin (2010)[31] for description of methane yield from organic waste. The first-order model was also proposed for formulation of biogas generation *eg.* Li Chen *et al.* (2011)[32] and Kafle *et al.* (2014)[33]. Most of those models reported above were based on the Monod model and the first-order model, and well performed in practical applications.

1.3 Scope of the Thesis

As explained above, many mathematical models are developed and available in literature in order to obtain an optimal anaerobic digestion process. Nevertheless, applicability of the models may be limited by diversity and complexity of anaerobic processes. Complexity of the models is not necessarily equated to accuracy. The general objective of the thesis is to develop a simple mathematical models which is suitable for description of anaerobic digestion process of fruit and vegetable waste. The scopes of the thesis are as follow:

1. Laboratory scale experiment to generate data of accumulated methane concentration and biogas volume obtained from fruit and vegetable waste as substrates and sludge from biogas plant as inoculum;
2. Introduction of experimental outcomes into inverse problems of methane generation process;
3. Numerical solution of the inverse problems and estimation kinetic parameters from proposed models;
4. Simulation of a methane co-fermentation process with outcomes form the analysis.

The contents of the thesis are organized as follows:

Chapter 1 contains the motivation behind the study and key aspects of mathematical modeling and numerical simulation of anaerobic digestion process.

In Chapter 2, selected basic considerations which are the most relevant to the biotechnological processes are presented.

Chapter 3 encompasses laboratory scale experiment, as well as chemical analysis and mathematical model used.

Chapter 4 presents the results of experiment of biogas from fruit and vegetable wastes, mathematical modeling of anaerobic digestion process, and results of simulation based on the proposed models. Limitation of both experimental and numerical results are described.

Chapter 5 presents the general conclusion concerning findings and contributions of this study, together with important outlines to be considered for further studies.

Chapter 2

Background and Selected Basic Considerations

2.1 Anaerobic Digestion of Fruit and Vegetable Wastes

2.1.1 Introduction

The estimated fruit and vegetable wastes percentage for each commodity group in each food supply chain, according to Food and Agricultural Organization (FAO), are 15, 9, 25, 10 and 7% in agricultural production, post-harvest handling and storage, processing and packaging, distribution and consumption respectively in South and South-East Asia [34]. In general, solid waste generated in traditional markets including fruit and vegetable wastes are disposed in municipal landfill and caused environmental problems [11].
The central food distribution market in Barcelona produces fruit and vegetable waste around 90 tonnes per day during 250 days per year. In Tunisia, fruit and vegetable waste collected from the market has been measured and estimated to be 180 tons per month. In India, fruit and vegetable waste constitute about 5.6 million tonnes annually and currently those wastes are disposed by dumping on the periphery of the cities [11].
Mostly, anaerobic digestion process with fruit and vegetable wastes contains volatile solids (VS) about 80-90%, and water content about 75-95%. Those are the main cause of heavy odor and plenty of leachate during collection, transportation and landfill of this wastes. Anaerobic digestion has been suggested as an alternative for handling organic waste for recovery of renewable energy. Some research groups have developed different anaerobic digestion processes for fruit and vegetable wastes [11].
The anaerobic degradation of cellulose-poor wastes like fruit and vegetable waste is limited by methanogenesis rather than by hydrolysis, although anaerobic digestion of particulate form is more likely limited by hydrolysis. One of the causes is a rapid acidification of wastes in the reactor, and another is a larger volatile fatty acids production, which inhibits the activity of methanogenic bacteria [12].
Fruit and vegetable wastes must undergo some pre-treatments before being loaded to the digesters. They are shredded to small particles and homogenized to facilitate optimal digestion process, and then diluted for decrease of the concentration of organic matter. Some studies also soggste buffer of wastes by the addition of sodium hydroxide solutions for increase of pH of fruit and vegetable wastes. Those wastes are also pre-treated at high temperature for improvement of the efficiency [11].

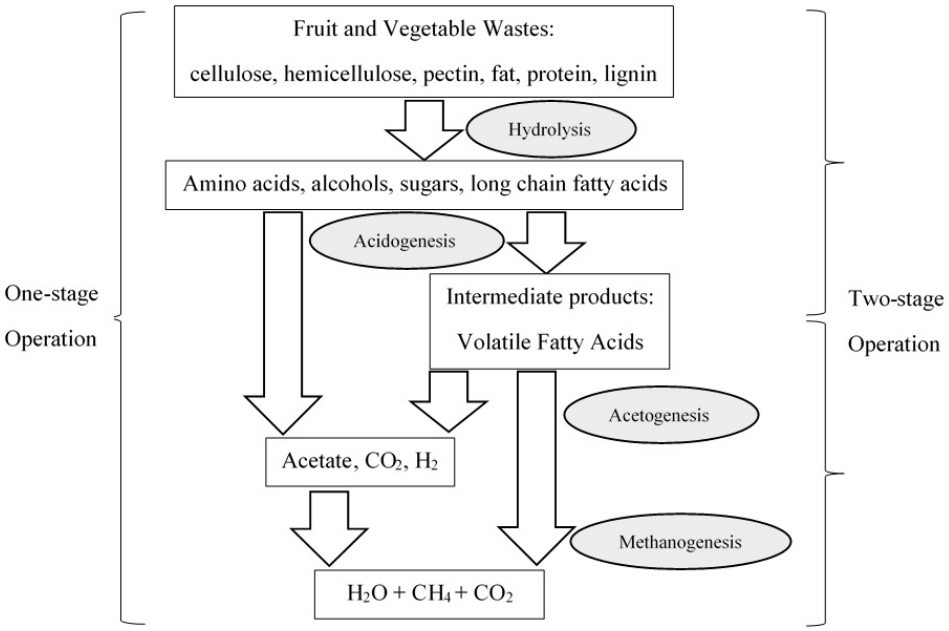


Figure 2.1: Reaction Scheme of Anaerobic Digestion of Fruit and Vegetable Waste (Based on H. Bouallagui *et al.* 2005 [11])

2.1.2 Stages of Anaerobic Digestion Process

Biomethanation of fruit and vegetable wastes process is accomplished by four stages of biochemical transformation. Those stages are hydrolysis, acidogenesis, acetogenesis, and methanogenesis. The whole processes is shown on Fig. 2.1.

Hydrolysis

Hydrolysis is the splitting of a compound with water [35]. Generally, the main components of organic matter are carbohydrates, fats, and proteins [36]. In this stage, undissolved compounds of carbohydrates, proteins, and fats are broken down into monomers (water-soluble fragments) by exoenzymes (hydrolase) of facultative and obligatorily anaerobic bacteria [37]. The hydrolysis of carbohydrates takes place within a few hours, while the hydrolysis of proteins and lipids within a few days. Lignocellulose and lignin are degraded only slowly and incompletely [37].
Particulate organic materials of fruit and vegetable waste such as cellulose, hemicellulose, pectin and lignin, must undergo liquefaction by extracellular enzymes before being absorbed by acidogenic bacteria. The rate of this stage is limited by several factors, such as pH, temperature, composition and particle size of the substrate and high concentrations of intermediate products [11].

Acidogenesis

Soluble organic components including products from hydrolysis are converted into organic acids, alcohols, hydrogen, and carbon dioxide by acidogens, and then further converted into acetic acids, hydrogen, and carbon dioxide. Those products results from hydrolysis. They are discharged into the medium inside the digester, and degraded by a large diversity of facultative anaerobes and anaerobes through many fermentative process. The degradation of those compounds results in the production of carbon dioxide, hydrogen gas, alcohols, organic acids, some organic-nitrogen compounds, and some organic-sulfur compounds [11, 35].
For example, the degradation process of monosaccharides, important products and stoichiometric reactions from glucose degradation are presented as follows [56].

$$\text{Acetate:} \quad C_{16}H_{12}O_6 + 2H_2O \longrightarrow 2CH_3COOH + 2CO_2 + 4H_2 \tag{2.1}$$

$$\text{Butyrate:} \quad C_{16}H_{12}O_6 \longrightarrow CH_3CH_2CH_2COOH + 2CO_2 + 2H_2 \tag{2.2}$$

Acetogenesis

Acetogenesis occurs in the acid-forming stage. Many of acids and alcohols, including butyrate, propionate, and ethanol, are produced during acid-forming stage, and they are degraded to acetate. The acetate is generated through the activity of acetogenic or acetate-forming bacteria. Acetate is an essential substrate used by methane-forming bacteria to produce biogas. Besides, carbon dioxide and hydrogen can be converted directly to acetate or methane. The conversion of large soluble organic compounds to small soluble organic compounds results in little change in the organic strength of the compounds. Some of those compounds are converted into organic acids and alcohols, and some others are converted to new bacterial cells [35].
For example, the conversion reactions of propionate and butyrate are presented as follows [56].

$$\text{Propionate:} \quad CH_3CH_2COOH + 2H_2O \longrightarrow CH_3COOH + 3H_2 + CO_2 \tag{2.3}$$

$$\text{Butyrate:} \quad CH_3CH_2CH_2COOH + 2H_2O \longrightarrow 2CH_3COOH + 2H_2 \tag{2.4}$$

A key feature in the anaerobic digestion process is the syntrophic relation between obligate hydrogen producing acetogens (OHPA) and the methane-generate bacteria. The OHPA are only capable of growing in environments with low concentration of their metabolic product H_2. Such environments can exist if a H_2 consuming species, such as methane-generate bacteria is present. On the other hand, methane-generate bacteria are inhibited by high fatty acids (substrates of OHPA) concentrations. As a result, any significant increase in either of those substrates will eventually lead to the inhibition of both groups of bacteria. Furthermore, in a stable anaerobic digester, consumption of H_2 leads to the production of acetate as main residual acid. Hence, high concentrations of propionate and butyrate are indicative of reactor failure [57, 58].

Methanogenesis

In this stage, methane is produced by methanogenic bacteria from acetic acids, hydrogen, and carbon dioxide as well as from other organic compounds, such as formic acid and methanol. All other fermentative products, such as acids, alcohols, and organic nitrogen compounds must be converted by methane-forming bacteria. If these compounds are not degraded, they would be accumulated in the digester, which would be responsible for high organic strength or carbonaceous biochemical oxygen demand (cBOD) of the supernatant. As long as the "Working Velocity" of acid-producing bacteria and methane-forming bacteria are roughly the same, metabolic activity of the methanogenic stage is safeguarded [35, 11].
The main reactions related to methanogenesis are presented as follows [56].

$$CH_3COOH \longrightarrow CH_4 + CO_2 \tag{2.5}$$

$$4H_2 + CO_2 \longrightarrow CH_4 + 2H_2O \tag{2.6}$$

2.2 Co-digestion process

Co-digestion is the term used to describe the combined treatment of several wastes with complementary characteristics. It is being one of the main advantages of the anaerobic technology, which has some benefits for treatment of different kind of wastes in one process: (1) increasing methane production yield due to supply of additional nutrients from co-substrates; (2) highly efficient for equipments utilization; and (3) low cost needed for processing multiple waste in a single facility [38, 39].
Several studies concerning the utilization of co-digestion, such as co-digestion of organic fraction of municipal solid wastes and agricultural residues, organic solid wastes and sewage sludge, or more specific wastes, have been documented. The recent works concerned the synergisms among co-digested substrates. Their results include the optimization of the carbon to nitrogen ratio for co-digestion process of municipal wastes and sewage sludge, which improved the buffer capacity in methane production. Co-digestion process of fish waste, abattoir waste water, and waste activated sludge were also conducted. The unbalance nutrients for each waste characterised by a low C/N ratio were regarded as an important limitation factor to anaerobic digestion of these organic wastes. By applying this technology, limitation of anaerobic digestion process of fruit and vegetable wastes can be reduced [12, 38].

2.3 Biochemical Reactions of Substrates and Products

In chemical reactions, there are reactants (chemical compounds) and products. During reactions, reactants change and often release energy to the environment. The changes that occur to the reactants result in the formation of products. Chemical reactions that occur inside bacterial cells are known as biochemical reactions. During some biochemical reactions, intermediates products are formed. They are usually short-lived and not

accumulated. However, change of specific environmental or operational conditions, such as change in pH or temperature, may cause the accumulation of intermediates. Initial substrates for microbes in anaerobic digesters include carbohydrates, lipids, and proteins. Those substrates are found as particulates, such as the carbohydrate cellulose and as colloids such as proteins [35].

2.3.1 Carbohydrates

Carbohydrates are synthesized in the green leaves of plants by the conversion of carbon dioxide into glucose during photosynthesis. Within the digester all carbohydrates are degraded inside the cell of facultative anaerobes and anaerobes. Carbohydrates too large to enter the cell, that is, in an insoluble or complex soluble form, must be hydrolyzed into smaller, soluble sugars outside the cell through the use of exoenzymes. Once hydrolyzed, the smaller, soluble sugars enter the cell, where they are degraded by endoenzymes [35].

2.3.2 Lipids

Lipids are usually found in animal and plant tissues, which do not dissolve in water. They are extracted from animal and plant tissues with non-polar organic solvents such ether. Lipids often disposed to a municipal anaerobic digester include fats and oil. Animal fats and vegetable fats or oils are the most plentiful lipids in nature. Butter and lard are examples of animal fats, while corn, olive, peanut, soybean, and sunflower oils are examples of vegetable oils [35].
In anaerobic digesters fats undergo degradation through two main steps. The first steps, the fats are hydrolyzed to glycerol and fatty acids. Lipase enzymes are used by bacteria to hydrolyze the fats. Glycerol is degraded, and the fatty acids released through hydrolysis are degraded into two carbon units at a time [35].

2.3.3 Proteins

Proteins are complex, high molecular-weight compounds, which have a relatively large surface and do not dissolve in wastewater. They can be classified into two, simple or conjugated according to their chemical composition. Simple proteins release only amino acids and no other compounds on hydrolysis. Blood serum albumin is an example of a simple protein. Conjugated proteins are more common than simple proteins and release amino acids and non-protein substances on hydrolysis. Glycoproteins that contain carbohydrates and lipoproteins that contain lipids are examples of conjugated proteins [35].

2.3.4 Volatile Fatty Acids

Volatile fatty acids appear as substrates and products in the anaerobic digester. Many serve as substrate for methane-generate bacteria, and they are the products of the fermentative activities of facultative anaerobes and anaerobes. Production of volatile fatty acids in an anaerobic digester will result in the production of methane. As wastes are degraded, new bacterial cells or sludge are produced. The cellular growth or amount of sludge produced is expressed as net biomass yield. The higher the volatile solids feed to

the digester, the larger amount of volatile fatty acids formed in the digester. The larger amount of volatile fatty acids in the digester, the greater the impact of volatile fatty acids on digester alkalinity and pH [35].

2.4 Biogas

Numerous gases are produced in an anaerobic digester. The gases produced largest quantities are methane and carbon dioxide. By volume, methane is 60% to 65%, and carbon dioxide is 35% to 40%. When anaerobic digestion of sludges and wastewaters is interrupted by changes in operational conditions, numerous insoluble and volatile compounds are produced. Those compounds may be released wherever anaerobic digestion of organic compounds is interrupted. Many of these compounds are malodorous and often are released in sewage, secondary clarifier sludge blanket, thickener, or anaerobic digester [35]. The inorganic compounds include methane and volatile organic compounds (VOC). The VOC contain volatile fatty acids, nitrogen-containing compounds, and volatile sulphur compounds (VSC). The production of VOC and VSC is due to the degradation of proteinaceous wastes [35].
Hydrogen sulfide (H_2S) is the most undesirable inorganic gas produced during an anaerobic digestion process. If biogas contains too much hydrogen sulfide, digester equipment may be damaged. This gas can be scrubbed from biogas, but scrubbing is cost-prohibitive for small wastewater treatment plants. Inorganic gases molecular nitrogen (N_2) and nitrous oxide (N_2O) are produced through anoxic respiration (denitrification) in the anaerobic digester. Anoxic respiration can occur with the transfer of nitrate ions (NO_3^-) to the digester with suldges. Nitrate-containing compounds such as sodium nitrate ($NaNO_3$) are also one of the causes for anoxic respiration to occur [35].

2.5 Environmental Factor Influencing the Anaerobic Digestion Process of Fruit and Vegetable Waste

Microbial metabolic processes are dependent on various factors. Furthermore, environmental requirements of the hydrolysis and acid-forming bacteria of the substrates differ from those of the methane-forming bacteria. It is important to consider and control parameters affecting those processes, so that an anaerobic process can be run in an optimum condition [37].

2.5.1 Temperature

Temperature is the most important environmental factor for bacterial growth. There are three temperature ranges associated with anaerobic digestion process. Those ranges are referred to as psychrophilic, mesophilic, and thermophilic [57, 59]. Methane-forming bacteria are the ones that are more sensitive to temperature conditions compared to other bacteria involved in a process. The relative growth rate of methane-bacteria under different temperature conditions is presented in Fig. 2.2 [59].
Generally, methane-forming bacteria belong to the mesophilics (32-42^{o}C)[37]. The anaerobic digestion under mesophilic range is more stable and requires a smaller energy. Castillo

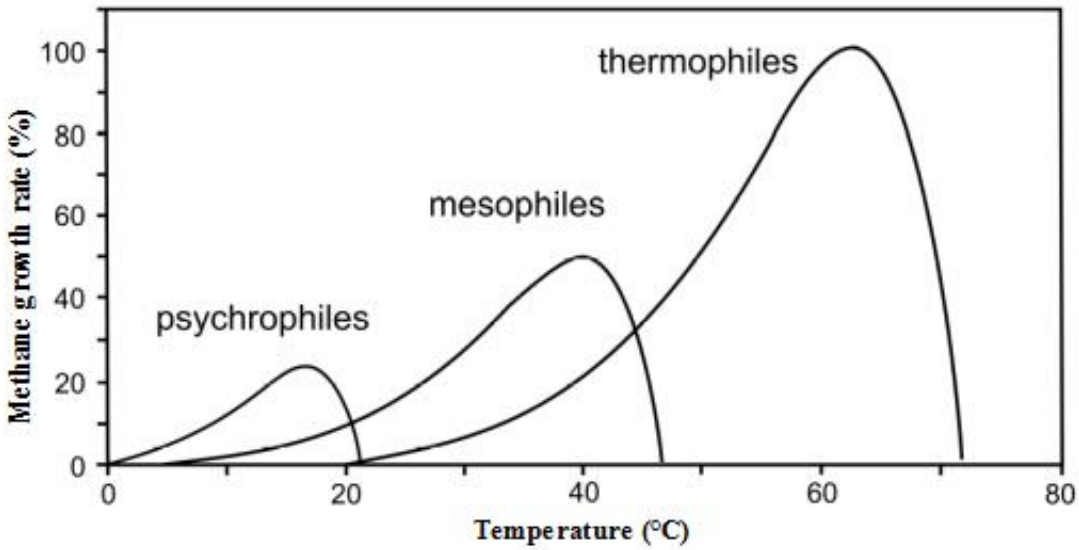


Figure 2.2: Relative growth rate of methane-forming bacteria under different temperature conditions (Based on a result by Grady *et al.* 2011 [59])

et al. (2006) found that the best operational temperature was 35^{o}C for 18 day experimental period, while a little fluctuation in temperature from 35^{o}C to 30^{o}C caused a reduction in the rate of biogas production [40].
In some cases, methane can be produced at a low temperature (0.6-1.2^{o}C), e.g. on the surface of the permafrost soils. In laboratory tests, methane formation could be proven also with temperatures below freezing, i.e. down to -3^{o}C [37]. However, this condition is less efficient to produce methane [57].
Anaerobic digestion under thermophilic condition has advantages, such as high metabolic rates and destruction of pathogens. Bouallagui et al. (2004) [10] investigated the performance of anaerobic digestion of fruit and vegetable waste under thermophilic (55^{o}C), psychrophilic (20^{o}C), and mesophilic (35^{o}C). They found that biogas production from the experimental thermophilic was higher than from psychrophilic and mesophilic. However, thermophilics has some drawbacks regarding to stability compared to mesophilics. Furthermore, energy requirements of thermophilics are higher than mesophilics [10]. Other studies also showed that anaerobic digestion was easily inhibited and less stable at thermophilic condition than at mesophilic temperatures [41].

2.5.2 pH

Stability of anaerobic digestion process is closely related to pH value. It is important to maintain a suitable pH in order to accomplish an efficient process. During the fermentation, carbon dioxide is continuously evolved and escapes into air. With falling pH value, more carbon dioxide is dissolved in the substrate as uncharged molecules. With a high pH value, dissolved carbon dioxide generates ionized carbonic acid. A drop in the pH value and a rise of carbon dioxide in the process is an indication of disturbance in an anaerobic digestion process. A first sign of the acidification is the rise of the propionic

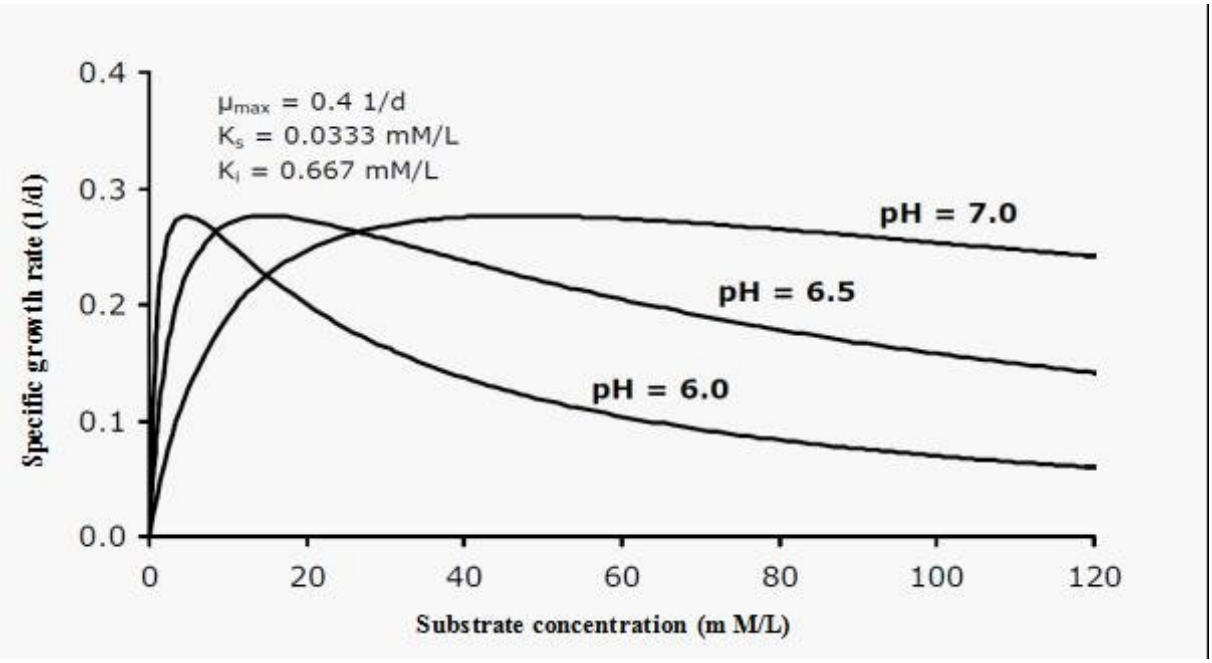


Figure 2.3: Effect of pH on maximum specific growth rate of bacteria (Based on a result by Gerber *et al.* 2008 [36])

acid concentration. If the pH value drops below pH 6.5, then the production of organic acids leads to a further decrease of the pH value by the hydrolytic bacteria and possibly lead to digester failure. Effect of pH on maximum specific growth rate of bacteria is shown on Fig. 2.3 [36, 37].
pH values suitable for anaerobic digestion were reported by various researchers. Ward *et al.* (2008) found that a pH range of 6.8-7.2 was ideal for anaerobic digestion. Lee *et al.* (2009b) reported that methanogenesis in an anaerobic digester occurs efficiently at pH 6.5-8.2, while hydrolysis and acidogenesis occurred at pH 5.5 and 6.5, respectively [40].
Some researchers investigated that pH affects the growth of microorganism during treatment of wastes with high concentration of Total Ammonia Nitrogen (TAN). Process instability due to ammonia often results in volatile fatty acids accumulation, which leads to a decrease of pH and thereby declining concentration of free ammonia [41]. The interaction between free ammonia, volatile fatty acids, and pH may lead to a condition where the process is running stably but with a lower methane yield [19, 42].

2.6 Experimentation Mode

2.6.1 Batch Operation

Batch operation is a biological process without interchange of mass with the environment, *i.e.* there are no input or output flow (except for the gas stream). In such a system, all fruit and vegetable wastes are loaded in a digester at the beginning of the reaction cycle, and allowed to undergo all degradation steps sequentially. A whole processes has clear separation between the first stage, where acidification proceeds much faster than

methanogenesis, and the second stage where acids are transformed into biogas [11].
Several advantages have been stated with regards to the use of batch tests for kinetic parameter estimation in anaerobic digestion. Those include: (1) the possibility to record the time evolution of several variables; (2) the relatively short time span as compared to continuous operations; and (3) simplicity and popularity as reflected in a wide acceptance [43]. Some investigations showed a good results on anaerobic digestion process of fruit and vegetable waste tested on batch systems. Converti *et al.* (1999) showed that some of fruit and vegetable wastes, for both under mesophilic and thermophilic, quickly degraded [11].
The specific features of batch processes are simple design and process control, robustness towards coarse and heavy contaminants, and lower investment costs [11]. However, the main drawback, stems from the lack of input excitation which result in a lack of parameter sensitivity. Bouallagui *et al.* (2001) and Marouani *et al.* (2002) found that the anaerobic digestion of fruit and vegetable waste with 8% Total Solids in a batch digester was inhibited by the VFA accumulation and irreversible decreasing of pH problems [11]. This can be reduced by using different sets of initial conditions and by determining a proper range substrate and biomass (S/X) ratio [43].

2.6.2 Single-stage VS Two-Stage Operation

In a single-stage operation, four stages of anaerobic digestion take place simultaneously in one reactor. This operation has advantages being simple and easy to operate, and requires low investment costs. About 90 percent of the full scale plants in Europe, which treated organic fraction of municipal solid wastes and biowastes, rely on continuous one-stage operations. Many studies concerned waste treatment in this operation. Mata Alvarez *et al.* (1992) investigated the performance of mesophilic one-stage in a completely stirred digester for treatment of organic fraction of waste from food market. Further, Mata Alvarez *et al.* (1990) mentioned that this waste was more biodegradable, which led to a larger and faster volatile fatty acid production. Those components inhibit the methane-generating bacteria. Lane (1979) reported that overloading of digester with fruit and vegetable waste resulted in a decrease of pH and gas yield, and an increase in the carbon dioxide content [11]. The configuration of one-stage anaerobic digester is shown on Fig. 2.4 [35].
In single-stage anaerobic digestion of solid wastes, problems may occur if the substrate is easily biodegradable. There is possibility for the accumulation of biomass within the anaerobic reactor of solid waste, and slow growing methane-generating bacteria may appear, which lead to high loading rates of organic components [11].
In two or multi-stage operation, the reactions take place sequentially in at least two reactors. The idea is to separate hydrolysis stage and methanogenesis stage in space and time, with the intention of decreasing the overall retention time and making the operation optimal. By combining acidogenesis and methanogenesis in one stage, uniform conditions are imposed on the both groups of bacteria [11].
The advantages of a two stage anaerobic digestion process are [35]:
1. Improvement in process control.
2. Disposal of excess acidogenic sludge without losing slow growing methanogens.
3. Attenuation of a toxic material in the more robust first stage.
4. Precise pH control in each reactor.
5. Higher methane content in the biogas from the methanogenic stage.

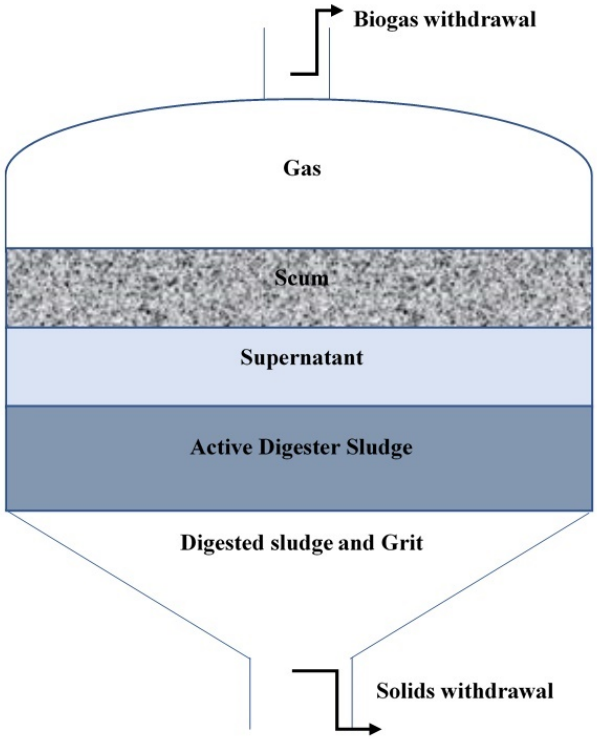


Figure 2.4: One-stage anaerobic digester (Based on results by Gerardi, 2003 [35])

6. Increased loading rate possible for the methanogenic stage.
The two-stage anaerobic digestion of a mixture of fruit and vegetable waste was studied in different works by Rajeshwari *et al.* (2001) and Ruynal *et al.* (1998). Those studies have led to a conclusion that phase-separated digesters may offer the best choice for high efficiency concerning both depuration rates and energy recovery [11]. The configuration of two-stage anaerobic digester is shown on Fig. 2.5 [35]

2.7 Operational Conditions of Anaerobic Digester

The rate-limiting reaction in anaerobic digestion process is usually the conversion of volatile fatty acids to methane. Methane-generating bacteria obtain very little energy from the degradation of volatile fatty acids. Most of the energy released from the volatile fatty acids is transferred to the methane. Because of the low energy yield obtained from volatile fatty acid by methane-generate bacteria, their growth rate is restricted, that is, the amount of substrate utilization per unit organism is high. Therefore, bacterial growth or sludge production is low and optimum operational conditions must be maintained for satisfactory rates of solids destruction and methane production. Those factors are responsible for the rate-limiting reaction of the conversion of volatile fatty acids to methane. However, if the substrates were mostly slowly degrading particulate materials, then the rate-limiting reaction would be the hydrolysis of the particulate materials [35].

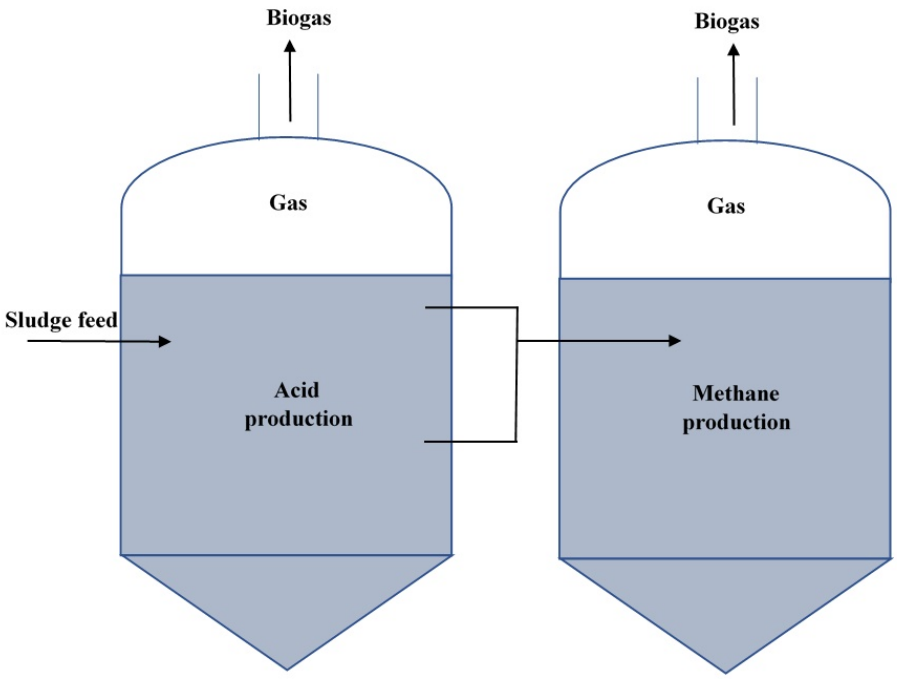


Figure 2.5: Two-stage anaerobic digester (Based on results by Gerardi, 2003 [35])

Start-up phase

Methane-generate bacteria are strict anaerobes and are extremely sensitive to changes in alkalinity, pH, and temperature. Therefore, operational conditions in the digester must be periodically monitored and maintained within optimum ranges. The successful operation of an anaerobic digester requires the activity of an abundant and diverse population of methane-generate bacteria, and cultivation in the digester that is heated to 35^{o}C with fresh inoculum may be helpful. Seeding with inoculum can be practiced during start-up or when the efficiency of the digester is deteriorates, for example, when the digester pH drops. During start-up, loading the digester should proceed slowly. Careful monitoring and control of pH and alkalinity are essential, especially when an efficient inoculum is not provided. One should proceed slowly in start-up period slowly with approximate time of about one month to achieve a steady state condition or an efficient operation [35].

Retention times

There are two significant retention times in an anaerobic digester. Those are solids retention time (SRT) and hydraulic retention time (HRT). The SRT is the average time during which microbes (solids) are in the anaerobic digester. The HRT is the time during which the sludge is in the anaerobic digester. Because the regeneration time required for a population of methane-generate bacteria to double in size is relatively long compared to aerobic bacteria and facultative anaerobic bacteria, typical SRT for anaerobic digesters are more than 12 days. Detention times for less than ten days are not recommended, because there is significant methane forming bacteria wash-out may occur [35].

High SRT values are advantageous for anaerobic digesters. This values maximize the removal capacity, reduce the required digester volume, and provide the buffering capacity for protection against the effects of shock loadings and toxic compounds in sludges. High SRT values also help permit biological acclimation of toxic compounds [35].
HRT values affect the rate and extent of methane production. This values control the conversion of volatile solids to gaseous products in an anaerobic digester. The design of the HRT is an essential factor for the final disposition of the digested sludge. The HRT may be relatively high or low, if the digested sludge is to be disposed with landfill or incineration. However, increases in detention time more than 12 days do not contribute significantly to the increase of volatile solids destruction [35].

2.8 Basic Models: Kinetics of Biogas Formation from Fruit and Vegetable Waste

It is important to considers growth of microbes, degradation of substrate, and formation of products to investigate the kinetics of biogas formation. The substrate balance of a continuous or a discontinuous process can be expressed by

$$\frac{\mathrm{dS}}{\mathrm{dt}}(\text{accumulation}) = \mathrm{D.S_0}(\text{input}) - \mathrm{D.S}(\text{output}) + \frac{\mathrm{dS}}{\mathrm{dt}}(\text{reaction}) \quad , \tag{2.7}$$

where D is dilution rate (flow per volume of digester, in 1/h) and S is substrate concentration [36]. The reaction rate is proportional to product formation and depending on the cell concentration. The kinetics of bacterial growth is strongly depending on growth requirements and the medium. The balance of microbial cells can be expressed as follow [36].

$$\frac{\mathrm{dX}}{\mathrm{dt}}(\text{accumulation}) = \mathrm{D.X_0}(\text{input}) - \mathrm{D.X}(\text{output}) + \mu\mathrm{X}(\text{growth}) + \mathrm{k_d X}(\text{death}) \quad , \tag{2.8}$$

where X is bacterial concentration (g/L), and μ is specific growth rate of bacteria. The bacterial growth depends on the specific growth rate, which influences by several factors, including substrate concentration, temperature, and pH.
The dilution rate for batch reactor is zero. The mass balances for bacteria and substrate can be expressed by

$$\frac{\mathrm{dS}}{\mathrm{dt}} = -\frac{\mathrm{dS}}{\mathrm{dt}} = -\mathrm{r_s X} \quad , \tag{2.9}$$

$$\frac{\mathrm{dX}}{\mathrm{dt}} = \mu\mathrm{X} + \mathrm{k_d X} \quad , \tag{2.10}$$

where r_s is consumption rate of substrate [44].

2.8.1 Kinetics of Microbial Growth: Monod Model

Monod discovered a non-linear relation between specific growth rate and substrate concentration, which the investigation was based on the work by Michaelis-Menten theory.

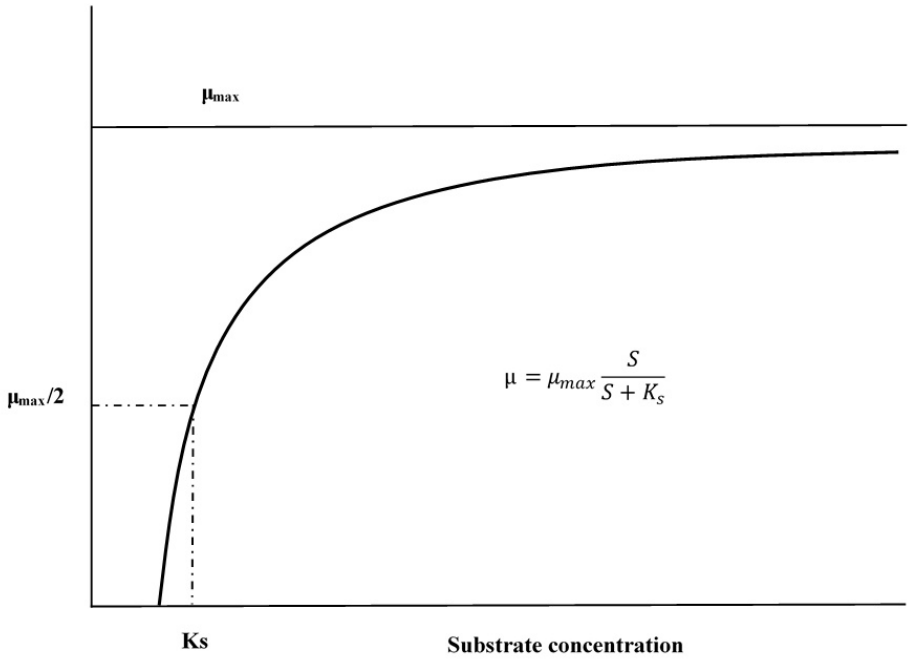


Figure 2.6: Specific growth rate of bacteria depending on substrate concentration (Based on work by Monod, 1949 [36])

The formulation of microbial growth can be expressed by

$$\mu = \mu_{\max} \frac{\mathrm{S}}{\mathrm{K_S + S}} \quad , \tag{2.11}$$

where μ_{max} is the limit of the specific bacteria growth rate when the saturation is reached, K_S is monod-constant describes substrate concentration at 50% of the maximum specific growth rate of bacteria [45].
The substrate concentration is the limiting factor, in which the component limits the specific growth rate due to its concentration. K_S is the affinity of bacteria to the limiting substrate. If the substrate concentration is greater than K_S, the specific growth rate is approximately linear, and K_S is always greater than zero. Thus, the values of $\frac{S}{K_S+S}$ is always less than one, and the values of the specific growth rate is less than μ_{max}. If the substrate is not the limited factor due to a high enough concentration of substrate, the maximum specific growth rate can be reached (Fig. 2.6) [36].
The Monod model for homogeneous and simple substrates is highly accurate. However, it is not appropriate for heterogeneous and complex substrate, such as degradation of municipal solid wastes. Furthermore, the lag phase is not included in the model [36].

2.8.2 Kinetics of Substrate Degradation

The substrate degradation can be calculated based on the specific growth rate of microbes. They need substrate to synthesis new cell material, to produce products such as acetic

acid or methane, and to supply requires maintenance and growth energy. The whole degradation process of substrate can be formulated as the sum of three terms [36].

$$\frac{\mathrm{dS}}{\mathrm{dt}_{\,\mathrm{r}}} = \frac{\mathrm{dS}}{\mathrm{dt}_{\,\mathrm{x}}} + \frac{\mathrm{dS}}{\mathrm{dt}_{\,\mathrm{e}}} + \frac{\mathrm{dS}}{\mathrm{dt}_{\,\mathrm{c}}} \tag{2.12}$$

Synthesis new cell material

The substrate degradation to biomass can be described stoichiometrically. The relation for acid forming bacteria use glucose as substrate, which can be expressed as follows [60].

$$\mathrm{C_6H_{12}O_6 + 1.2NH_3 \longrightarrow 1.2C_5H_7NO_2 + 3.6H_2O} \tag{2.13}$$

According to this reaction, 1.2 mol acid-forming bacteria are formed by 1 mol glucose. Considering the molar mass of glucose and biomass and assuming that the $C_5H_7NO_2$ represents 92% of the dry biomass, the yield coefficient of glucose to acid-forming bacteria Y_x is 0.82 g/g [60]. Based on this condition, substrate degradation due to biomass formation depends on the change of cell concentration of microbes can be expressed by [36].

$$\frac{\mathrm{dS}}{\mathrm{dt}_{\,\mathrm{x}}} = -\frac{1}{\mathrm{Y_x}}\frac{\mathrm{dX}}{\mathrm{dt}} = -\frac{\mu \mathrm{X}}{\mathrm{Y_x}} \tag{2.14}$$

Energy supply of bacteria

Bacteria need energy for their living to the synthesize cell ingredients, which are degraded continuously to sustain the concentration gradient between cell interior and exterior. The energy demand can be divided into growth energy and maintenance energy. The required energy is provided by substrate [36].
The maintenance energy coefficient is specified by Moletta & Albagnac (1984) with 0.0169 mol ATP per g biomass and per hour. Assuming that degradation of 1 mol glucose to 3 mol acetic acid will result in 6 ATP, the maintenance energy rate K_{mx} is 12.1 g glucose per g active biomass and per day.[36]
According to Moletta et al. (1986)[60], the substrate degradation for energy supply is expressed by

$$\frac{\mathrm{dS}}{\mathrm{dt}_{\,\mathrm{e}}} = \mathrm{K_{sx}X}\mu + \mathrm{K_{mx}X}\frac{\mathrm{S}}{\mathrm{K_S + S}} \tag{2.15}$$

The first term on the right side is the substrate degradation for growth energy supply and the second term is the substrate degradation for maintenance energy supply.

Conversion of substrate

The conversion of substrate to products can be considered stoichiometrical as well, for example, the degradation of acetic acid to methane is expressed by

$$\mathrm{CH_3COOH \longrightarrow CH_4 + CO_2} \quad , \tag{2.16}$$

The degradation of 1 mol acetic acid results in 1 mol methane. Using the molar mass of acetic acid and methane, the yield coefficient of acetic acid to methane, Y_s, is 0.27 g/g.

Using the calculated yield coefficient, the substrate degradation due to product formation is determined by the expression (2.17) [36].

$$\frac{\mathrm{dS}}{\mathrm{dt}}_{\mathrm{c}} = \frac{1}{\mathrm{Y_s}} \left(\frac{\mathrm{dP}}{\mathrm{dt}} \right)_{\mathrm{p}} \tag{2.17}$$

2.8.3 Kinetics of Product Formation

The end product of fermentation process is biogas. Nevertheless, a lot of intermediates are also very important products. The kinetics of product formation can be calculated by principles based on the kinetics of substrate degradation and of bacterial growth, respectively. Gaden (1959) investigated fermentation processes and classified products into three types [36]:
Type I products, which result from primary energy metabolism. The product is produced at the same time as substrate is degraded.

$$\frac{\mathrm{dP}}{\mathrm{dt}} = \mathrm{Y_{p1}} \mu \mathrm{X} = \mathrm{Y_{p1}} \frac{\mathrm{dX}}{\mathrm{dt}} \tag{2.18}$$

Type II products, which result from energy metabolism indirectly. The product is produced at side reactions or following interactions of direct metabolic products. Therefore, the product formation is delayed and two maxima appear in substrate degradation and bacterial growth.

$$\frac{\mathrm{dP}}{\mathrm{dt}} = \mathrm{Y_{p1}} \mu \mathrm{X} + \mathrm{Y_{p2}} \mathrm{X} = \mathrm{Y_{p1}} \frac{\mathrm{dX}}{\mathrm{dt}} + \mathrm{Y_{p2}} \mathrm{X} \tag{2.19}$$

Type III products, which do not result from energy metabolism. The formation of complex molecules (biosynthesis), such as the formation of antibiotics is the example of this process. Energy metabolism is practically complete while the complex product accumulates.

$$\frac{\mathrm{dP}}{\mathrm{dt}} = \mathrm{Y_{p2}} \mathrm{X} \tag{2.20}$$

Chapter 3

Material and Methods

3.1 Lab-Scale Digester

The laboratory-scale digesters used for experiments were a batch single-stage anaerobic digester. The reactors used were 50 L made of aluminium, 500 and 2000 ml batch digesters made of glass. The temperature for 500 and 2000 ml were maintained at 30-45 °C using water bath, while temperature for 50 L digester was not maintained (under ambient temperature). Digester set up for each digester is shown in Figure 3.1 and Figure 3.2.

3.2 Substrate and inoculum

The substrate used in the experiments were fruit and vegetable waste, while inoculum used in the experiments were horse dung and sludge from biogas plant located in Tsuyama and Kojima. The preparation was as follows: sorting of wastes, homogenizing the size of wastes by blender, and adding water before the wastes mixed with inoculum, finally adding inoculum and mixed into digester.

3.3 Chemical Analysis Methods

Several analysis was done for the experimental results of biogas production from fruit and vegetable waste, including biogas volume and concentration, elemental analysis (CHN), COD (chemical oxygen demand), TS (total solids), and water content.
Biogas volume was measured by displacement method. Elemental analysis (CHN) was performed with The PerkinElmer 2400ii CHN Elemental Analyzer. Concentration of methane and carbon dioxide was analyzed with gas chromatograph Shimadzu GC-8A1F (FID). COD and TS was performed in accordance to Standard Methods [48].

3.3.1 COD analysis - Potassium permanganate acidic method

Sample Pre-treatment

Samples were collected and separated from solid compounds by centrifuging.

Analysis

Samples were poured into a 300ml conical flask and the volume was increased to 100 ml by adding water (dilution of the sample should be enough so that about a half of potassium permanganate solution can remain after the next procedure). Once 10 ml sulfuric acid (1+2) and 10ml potassium permanganate were added, the flask was stirred and immediately put into a boiling water bath to heat up for 30 minutes, under temperature 60 - 80^{o}C. After the flask was taken out from the bath, 10 ml sodium oxalate was added and subsequently, the flask was stirred. Titration was conducted until the color of the solution turns slightly red due to the potassium permanganate solution (wait for 30 seconds at this point). In parallel, 100 ml of distilled or deionized water was poured into a conical flask, and the same procedure was conducted. The oxidation of permanganate and oxidation-reduction potentiometric titration are expressed as below

$$2KMnO_4 + 3H_2SO_4 \longrightarrow K_2SO_4 + 2MnSO_4 + 3H_2O + 5O \tag{3.1}$$

$$2KMnO_4 + 5Na_2C_2O_4 + 3H_2SO_4 \longrightarrow K_2SO_4 + 5Na_2SO_4 + 10CO_2 + 3H_2O \tag{3.2}$$

CODMn (mgO/l) is calculated by following equation:

$$\mathrm{COD}(mgO_2/l) = (a - b)f\frac{1000}{V} \tag{3.3}$$

where a is volume of potassium permanganate solution required in titration (ml) used for samples, b is volume of potassium permanganate solution required in titration (ml) used for blank, f is factor of potassium permanganate solution, and V is volume of the water sample (ml), respectively.

3.3.2 Total Solids

Total solids is the amount of solid present in the sample after the water is vaporised. The sample, approximately 10 gr, is taken and poured into combustion boat and dried to a constant weight at about 105^{o}C in furnace. Total solids in mg/L is formulated as follows.

$$\mathrm{TS}(g/L) = \frac{(A - B)x100}{V} \tag{3.4}$$

where A is weight of dried residue and combustion boat (g), B is weight of combustion boat (g), and V is volume of sample (ml).

3.4 Mathematical model of anaerobic digestion of fruit and vegetable waste

The modeling of biogas formation from fruit and vegetable waste were based on several models, including the first-order model, the Monod model, the Sosnowski model, and the Grau model. Those models are based on the microbial growth and substrate consumption rates which depend on a growth-limiting substrate concentration. Nutrients were assumed to be substrates that are supplied in excess [45, 51].

3.4.1 Monod Model

The model involves two stages of anaerobic digestion: first hydrolysis-acidogenesis stage, and second acetogenesis-methanogenesis stage. First stage, soluble organic compounds are converted into acids and then into volatile fatty acids and carbon dioxide by acidogenic and acetogenic bacteria. As the hydrolysis is not the limiting step in the anaerobic process of fruit and vegetable waste, the hydrolysis was assumed to be done in a very short time. Fruit and vegetable waste tends to rapid acidogenesis and acetogenesis because the low pH of this wastes [12]. The reaction is described below:

$$\mathrm{FVW(sol)} + \mathrm{bacteria} \longrightarrow \mathrm{VFA} + \mathrm{CO_2} \tag{3.5}$$

Second stage, volatile fatty acids and carbon dioxide produced from the previous stage are converted into methane and carbon dioxide by methanogenic bacteria. The reaction is described below:

$$\mathrm{VFA} + \mathrm{bacteria} \longrightarrow \mathrm{CH_4} + \mathrm{CO_2} \tag{3.6}$$

Effect of temperature on microbial growth was considered in this study. This is similar to the observation of enzyme activity, where an increase of temperature up to a certain point increases the bacterial activity while the decrease of temperature decreases the activity [44].

$$\mathrm{k} = \mathrm{k_{max}}\exp\left(-\frac{\mathrm{Ea}}{\mathrm{RT}}\right) \tag{3.7}$$

where k is rate constant, Ea is constant of activation energy ($Jmol^{-1}$), T is temperature (K), and R is constant of gas molar ($JK^{-1}mol^{-1}$). This equation has been applied to a various parameters, such as specific growth rate (Siegrist *et al.*, 2002 [22]), the maximum specific growth rate (Angelidaki *et al.*, 1993 [19]), the saturation constant (Siegrist *et al.*, 2002 [22]), the hydrolysis rate, the death rate (Siegrist *et al.*, 2002 [22]), the dissociation constant (Angelidaki *et al.*, 1993 [19]), etc. For an empirical description, the temperature dependence implied by Arrhenius may be adapted [36].
The formulation of temperature as the influence factor for microbial growth is expressed by a principle based on the adaptation of Arrhenius law *cf.* Bergter (1983) and Sinclair and Kristiansen (1993) [36, 47].

$$\mu_{\mathrm{max}}(\mathrm{T}) = \mathrm{k_1}\exp\left(-\frac{\mathrm{Ea}}{\mathrm{RT}}\right) - \mathrm{k_2}\exp\left(-\frac{\mathrm{Ea}}{\mathrm{RT}}\right) \tag{3.8}$$

where k_1,k_2 are rate constant (day^{-1}), Ea is constant of activation energy ($Jmol^{-1}$), T is temperature (K), and R is constant of gas molar ($JK^{-1}mol^{-1}$).
According to the equations explained above, the mass balance of substrate degradation, growth of bacteria, and biogas formation for batch is formulated [47]:

$$\frac{\mathrm{dS}}{\mathrm{dt}} = \frac{1}{\mathrm{Y}}\mu\mathrm{X} \quad , \tag{3.9}$$

$$\frac{\mathrm{dX}}{\mathrm{dt}} = \mu\mathrm{X} \quad , \tag{3.10}$$

$$\frac{\mathrm{dP}}{\mathrm{dt}} = \mathrm{Y_p}\mu\mathrm{X} \quad , \tag{3.11}$$

where P is concentration of biogas formation, Y and Y_p are yield coefficient for substrate and biogas, respectively. By substituting Eq.(3.8) into Eq. (3.9 - 3.11), system of equations (3.12), (3.13), (3.14) as follow [47].

$$\frac{\mathrm{dS}}{\mathrm{dt}} = \frac{1}{\mathrm{Y}}\frac{\mu_{\max}\mathrm{XS}}{\mathrm{K_s} + \mathrm{S}} \quad , \tag{3.12}$$

$$\frac{\mathrm{dX}}{\mathrm{dt}} = \frac{\mu_{\max}\mathrm{XS}}{\mathrm{K_s} + \mathrm{S}} \quad , \tag{3.13}$$

$$\frac{\mathrm{dP}}{\mathrm{dt}} = \mathrm{Y_p}\frac{\mu_{\max}\mathrm{XS}}{\mathrm{K_s} + \mathrm{S}} \quad , \tag{3.14}$$

where P represents the concentration of methane and carbon dioxide, which can be expressed by

$$\frac{\mathrm{dCH_4}}{\mathrm{dt}} = \mathrm{Y_p}\frac{\mu_{\max}\mathrm{XS}}{\mathrm{K_s} + \mathrm{S}} \quad , \tag{3.15}$$

$$\frac{\mathrm{dCO_2}}{\mathrm{dt}} = \mathrm{Y_p}\frac{\mu_{\max}\mathrm{XS}}{\mathrm{K_s} + \mathrm{S}} \quad . \tag{3.16}$$

3.4.2 P. Sosnowski Model

P. Sosnowski *et al.* (2008) proposed a model for 2 stages of anaerobic digestion process, hydrolysis stage and methanogenesis stage. The hydrolysis stage and the methanogenesis stage were modeled by first-order kinetic and Monod-like reaction, respectively [48].

Hydrolysis Stage

P. Sosnowski assumed that the process of anaerobic digestion of complex substrate of particulate material, such as solid waste and sewage sludge, were limited by the hydrolysis stage. Particulate organic material must be decomposed to solutes capable of being actively or passively transported across cell membranes for microbial metabolism. There are two mechanisms to describe the hydrolytic process [51]:
1. The microbes secrete enzymes into the culture medium for extracellular degradation for size reduction.
2. The microbes adhere to a particle and consume soluble products released by reactions that are catalyzed by enzymes locally produced.
The first-order model also used to describe hydrolysis stage of organic polymer, as the enzymatic activity is not influenced by bacterial growth. The formulation of first-order model can be expressed by

$$\mathrm{r_s} = \mathrm{K_h}.\mathrm{S} \tag{3.17}$$

where K_h is hydrolytic constant [45].

Methanogenesis Stage

Methanogenesis is assumed to be the rate-limiting step of the anaerobic digestion of complex substrates. Modeling efforts often view methanogenesis as the limiting step because methanogenic bacteria have the higher sensitivity and lower growth rates compared to those in the non-methanogenic group bacteria. When substrates were overloaded in one digester, high volatile fatty acid product concentrations would inhibit polymer hydrolysis and acidogenesis and resulted in methanogenesis, as opposed to hydrolysis, and methanogenesis stage would become the overall anaerobic digestion rate-limiting step [51, 52].
Based on the stages described, ordinary differential equations of the process can be formulated by

$$\frac{\mathrm{dS}}{\mathrm{dt}} = -\mathrm{kS} \quad , \tag{3.18}$$

$$\frac{\mathrm{dV}}{\mathrm{dt}} = \mathrm{Y_{VFA/S}kS} - \mathrm{V_{VFA}X_0}\frac{\mathrm{VFA}}{\mathrm{K_s + VFA}} \quad , \tag{3.19}$$

$$\frac{\mathrm{dCH_4}}{\mathrm{dt}} = \mathrm{Y_{CH4/VFA}V_{VFA}X_0}\frac{\mathrm{VFA}}{\mathrm{K_s + VFA}} \quad , \tag{3.20}$$

$$\frac{\mathrm{dCO_2}}{\mathrm{dt}} = \mathrm{Y_{CO2/S}kS} + \mathrm{Y_{CO2/VFA}V_{VFA}X_0}\frac{\mathrm{VFA}}{\mathrm{K_s + VFA}} \quad , \tag{3.21}$$

where k is constant of first-order kinetic (d^{-1}), and VFA is concentration of volatile fatty acids [48]. $Y_{VFA/S}$, V_{VFA}, $Y_{CH4/VFA}$, $Y_{CO2/S}$, and $Y_{CO2/VFA}$ represent as yield factor of VFA from substrate S, maximum specific utilization of VFA, yield factor of methane from VFA, yield factor of carbon dioxide from substrate S and from VFA, respectively.

3.4.3 Grau Model

The mechanism of substrate utilization by a bacterial cell can be generally described as a sequence of three complex processes: contact of a cell with the molecule of substrate; transport of the molecule into the cell; and intermediate metabolism of the substrate. On the basis of the described general mechanism it is practical to classify various types of substrates into three main groups: (a) single component substrate, which are directly transportable into the cell;(b) multicomponent substrates, which are mixtures of several single substrates; and (c) complex substrates, which have to be changed externally prior to the transportation into the cell [50].
Grau *et al.* (1975) considered multicomponent substrate degradation rate, for which the concept was based on the linear degradation by Monod. The decrease of removal rate caused by a reduced number of components and thus the decrease in total substrate concentration with time must also be explained. To reflect this, Grau [50] used an exponent similar to the order of chemical reactions, which can be formulated.

$$\frac{\mathrm{dS}}{\mathrm{dt}} = -\mathrm{k_s X}\frac{\mathrm{S}}{\mathrm{S_0}}^{\mathrm{n}} \quad . \tag{3.22}$$

For the first order $n = 1$, becomes

$$\frac{dS}{dt} = -k_s X \frac{S}{S_0} \quad , \tag{3.23}$$

where k_s is Monod kinetic constant, and S_0 is the initial concentration of substrate. Integrating Eq. 3.21, the equation would be eventually become:

$$\frac{S}{S_0} = \exp\left(-\frac{k_{1s} X_0 t}{S_0}\right) \tag{3.24}$$

The Grau model was modified to describe degradation of substrate, microbial growth rate, and methane production, which are formulated by [49]

$$\frac{dS}{dt} = -\frac{1}{\alpha}\gamma X \frac{S}{S_0} \quad , \tag{3.25}$$

$$\frac{dX}{dt} = \gamma X \frac{S}{S_0} - \delta X \quad , \tag{3.26}$$

$$\frac{dP}{dt} = \beta\gamma X \frac{S}{S_0} \quad , \tag{3.27}$$

where α, β, γ, and δ are kinetic parameters represent yield coefficient for substrate and methane, maximum specific growth rate, and decay coefficient, respectively.

3.5 Analysis Method

Initial value problems from the proposed model (Monod and P. Sosnowski model) were solved with the Adams-Bashforth-Moulton-Predictor-Corrector method in conjunction with the Runge-Kutta method to generate values of numerical solutions at the first three steps. the Adams-Bashforth-Moulton-Predictor-Corrector is a fourth order method, which numerical values of the solution at previous four steps are used to generate the value at the next step. It is essential to generate numerical values at the first three steps, and Runge-Kutta method was applied for that purpose [47, 48].
Kinetic parameters from proposed model were determined by trial error method. An interval was set for each parameters, which divided into sub-intervals of equal length. The values of parameters was sought among those sets of parameters values, which minimizes the error between experimental results and numerical results [47, 48]. The initial value problem of proposed models were solved numerically using the values of the parameters, and the error between the experimental results and numerical results were evaluated [47][48].

3.5.1 The Monod Model

The formulation of trial-error methods for parameter estimation of Monod model is described as follows [47].

There are eight parameters
$k_1 : [k1_{min}, k1_{max}]$
$k_2 : [k2_{min}, k2_{max}]$
$E_1 : [E1_{min}, E1_{max}]$
$E_2 : [E2_{min}, E2_{max}]$
$K_s : [Ks_{min}, Ks_{max}]$
$Y : [Y_{min}, Y_{max}]$
$Y_p : [Yp_{min}, Yp_{max}]$
An interval was set for each of those parameters. Each of those intervals were divided into 50 subintervals of equal length. The values of the parameters which minimizes the error.

$$\sum \sqrt{(CH_4\text{experiment} - CH_4(\text{numerical results}))^2 + (CO_2\text{experiment} - CO_2(\text{numerical results}))^2} \tag{3.28}$$

was sought among the sets of the parameter values
$k_1 = k1_{min} + i\triangle k_1$
$k_2 = k2_{min} + i\triangle k_2$
$E_1 = E1_{min} + i\triangle E_1$
$E_2 = E2_{min} + i\triangle E_2$
$K_s = Ks_{min} + i\triangle K_s$
$Y = Y_{min} + i\triangle Y$
$Y_p = Yp_{min} + i\triangle Y_p$
with i = 0,1,...,n
where

$$\begin{aligned}
\triangle k_1 &= \frac{k1_{max} - k1_{min}}{n}, \\
\triangle k_2 &= \frac{k2_{max} - k2_{min}}{n}, \\
\triangle E_1 &= \frac{E1_{max} - E1_{min}}{n}, \\
\triangle E_2 &= \frac{E2_{max} - E2_{min}}{n}, \\
\triangle Y &= \frac{Y_{max} - Y_{min}}{n}, \\
\triangle Y_p &= \frac{Yp_{max} - Yp_{min}}{n}, \\
\triangle K_s &= \frac{Ks_{max} - Ks_{min}}{n}
\end{aligned} \tag{3.29}$$

3.5.2 The P. Sosnowski Model

The formulation of trial-error methods for parameter estimation of P. Sosnowski model is described as follows [48].
There are five parameters.
$k : [kmin; kmax]$

$YVFA/S : [Yvfamin; Yvfamax]$
$VVFA : [Vvfamin; Vvfamax]$
$YCH4/VFA : [Ych4vmin; Ych4vmax]$
$YCO2/VFA : [Yco2vmin; Yco2vmax]$
An interval was set for each of those parameters. Each of those intervals were divided into 10 subintervals of equal length. The values of the parameters which minimizes the error for all five experimental points for each methane and carbon dioxide:

$$\sum \sqrt{(CH_4exp - CH_4num)^2 + (CO_2exp - CO_2num)^2} \tag{3.30}$$

The best values of parameters were sought among the sets of the parameter values
$k = k_{min} + i\triangle k$
$Y_{VFA/S} = Y_{VFA/S} + i\triangle Y_{VFA/S}$
$V_{VFA} = V_{VFA} + i\triangle V_{VFA}$
$Y_{CH4/VFA} = Y_{CH4/VFA} + i\triangle Y_{CH4/VFA}$
$Y_{CO2/VFA} = Y_{CO2/VFA} + i\triangle Y_{CO2/VFA}$
with i = 0,1,...,n
where

$$\begin{aligned}
\triangle k &= \frac{k_{max} - k_{min}}{n}, \\
\triangle Y_{VFA/S} &= \frac{Yvs_{max} - Yvs_{min}}{n}, \\
\triangle V_{VFA} &= \frac{Vvfa_{max} - Vvfa_{min}}{n}, \\
\triangle Y_{CH4/VFA} &= \frac{Ych4v_{max} - Ych4v_{min}}{n}, \\
\triangle Y_{CO2/VFA} &= \frac{Yco2v_{max} - Yco2v_{min}}{n},
\end{aligned} \tag{3.31}$$

for $n = 12000$.

3.6 Inverse Problem

An inverse problem considered here based on the modified Grau model, where α, β, γ, and δ represent as kinetic parameters. An additional information were obtained from experimental data of biogas production (concentration of methane), which then used for the estimation of kinetic parameters. The initial concentration of methane (P_0) equals zero, while initial concentration for substrate (S_0) and bacteria (X_0) are known. Denote by $S(t, \alpha, \beta, \gamma, \delta)$, $X(t, \alpha, \beta, \gamma, \delta)$, and $P(t, \alpha, \beta, \gamma, \delta)$, the solutions of Eqs. (3.25), (3.26), (3.27) that satisfies the initial conditions [48].

$$\mathrm{S}(0, \alpha, \beta, \gamma, \delta) = \mathrm{S}_0 \tag{3.32}$$

$$\mathrm{X}(0, \alpha, \beta, \gamma, \delta) = \mathrm{X}_0 \tag{3.33}$$

$$\mathrm{P}(0, \alpha, \beta, \gamma, \delta) = 0 \tag{3.34}$$

Suppose at $t_0 = 0$, and P_1, P_2, ... , P_m are the values of P at time $t = t_1$, t_2, ... , t_m, respectively. Let

$$\mathrm{P}(\mathrm{t_i}, \alpha, \beta, \gamma, \delta) = \mathrm{P_i}; (\mathrm{i} = 1, 2, ..., \mathrm{m}) \tag{3.35}$$

The residuals between P and P_i is formulated as follow.

$$\mathrm{g_i}(\alpha, \beta, \gamma, \delta) = \mathrm{P_i} - \mathrm{P}(\mathrm{t_i}, \alpha, \beta, \gamma, \delta); (\mathrm{i} = 1, 2, ..., \mathrm{m}) \tag{3.36}$$

where P_i and $P(t_i, \alpha, \beta, \gamma, \delta)$ represent as measured methane concentration and results obtained from simulation, respectively. Solutions of the inverse problem are values of the parameters that satisfy

$$\mathrm{g_i}(\alpha, \beta, \gamma, \delta) = 0; (\mathrm{i} = 1, 2, ..., \mathrm{m}) \tag{3.37}$$

which form a system of nonlinear equations for unknown parameters α, β, γ, and δ [48]. The estimation of the kinetic parameters is based on the minimization of the ordinary least squares given by

$$\mathrm{S} = \frac{1}{2} \sum_{\mathrm{i}=1}^{\mathrm{m}} \left[\mathrm{g_i}\left(\alpha, \beta, \gamma, \delta\right)\right]^2 \tag{3.38}$$

where S is sum of squares error [48].

3.7 Description of Numerical Schemes

The proposed model of Monod and P. Sosnowski were implemented in C program, while modified Grau model was implemented in Matlab using $nlinfit$ function, which uses Levenberg-Marquardt method to solve nonlinear least square problem.

3.7.1 Levenberg-Marquardt

The Levenberg Marquardt method is a standard technique for solving nonlinear least square problems. Least square problems arise in the context of fitting a parameterized function to a set of measured data points by minimizing the sum of the square errors between the data points and the values of function. Nonlinear least square methods iteratively reduce the sum of the squares of the errors between the values of function and the measured data points through a sequence of updates to parameter values.(references)
The Levenberg Marquardt method is a combination of two minimization methods: the Gradient descent method and the Gauss Newton method. In the gradient descent method, the sum of the squared errors is reduced by updating the parameters in the steepest descent direction. In the Gauss Newton method, the sum of the squared errors is reduced by assuming the least squares function is locally quadratic, and finding the minimum of the quadratic function. The Levenberg Marquardt method acts as a gradient descent method when the parameters are far from their optimal value, and acts as the Gauss Newton

method when the parameters are close to their optimal value. (references)
Gradient descent method is a general minimization method which updates parameter values in the downhill direction. This method converges well for problems with simple objective functions. For problems with thousands of parameters, this methods are sometimes the only viable choice.(references)
The Gauss Newton method is a method for minimizing a sum-of-squares objective function. It presumes that the objective function is approximately quadratic in the parameters near the optimal solution(reference). For moderately sized problems the Gauss Newton method typically converges much faster than gradient descent method(reference).

Iterative Procedure

To minimize the least squares given by Eq. (3.37), the partial derivatives of S with respect to each of the unknown kinetic parameters α, β, γ, and δ should be equal to zero at the solution. Specifically, the kinetic parameters need to be find so that the proposed model behave similarly to the data. The Levenberg-Marquardt method [48] was applied for was applied for computational analysis of the nonlinear least square problem of methane generation. The recurrence formula of the Levenberg-Marquardt method is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left(J_n^T J_n + \lambda \mathbf{I}\right)^{-1} J_n^T \mathbf{g}_n \tag{3.39}$$

where $\mathbf{x_{n+1}}$ and $\mathbf{x_{n+1}}$ are vector of estimated parameter values, which can be written as

$$\mathbf{x}_{n+1} = \begin{bmatrix} \alpha_{n+1} \\ \beta_{n+1} \\ \gamma_{n+1} \\ \delta_{n+1} \end{bmatrix}, \quad \mathbf{x}_n = \begin{bmatrix} \alpha_n \\ \beta_n \\ \gamma_n \\ \delta_n \end{bmatrix},$$

$\mathbf{g_n}$ is the residuals

$$\mathbf{g}_n = \begin{bmatrix} g_{1,n} \\ g_{2,n} \\ g_{3,n} \\ g_{4,n} \end{bmatrix},$$

and J_n is Jacobian matrix

$$J_n = \begin{bmatrix} a_{1,n} & b_{1,n} & c_{1,n} & d_{1,n} \\ a_{2,n} & b_{2,n} & c_{2,n} & d_{2,n} \\ a_{3,n} & b_{3,n} & c_{3,n} & d_{3,n} \\ a_{4,n} & b_{4,n} & c_{4,n} & d_{4,n} \end{bmatrix}.$$

Here λ represents as positive parameter called damping parameter, and $\mathbf{I}$ represent as identity matrix. Large values of damping parameter lead to the gradient descent method, while small values of damping parameter result in the Gauss-Newton method, which is formulated by [48]

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left(J_n^T J_n\right)^{-1} J_n^T \mathbf{g}_n \quad . \tag{3.40}$$

Numerical Implementation

The Levenberg Marquardt method was implemented in matlab, which has a built-in algorithm called $nlinfit$. The systems of ordinary differential equations were coded and the kinetic parameters were declared. The experimental data need to be fit also supplied. The systems of differential equations were solved by combining $ode15s$ and $interp1$. The input and output arguments were

$$\text{beta} = \text{nlinfit}(\text{X}, \text{Y}, \text{modelfun}, \text{beta0}) \tag{3.41}$$

where X is a matrix of predictor (independent variable) values, Y is the observed output (dependent variable), and $modelfun$ is the nonlinear regression model function [61]. $modelfun$ is a function, which accepts two inputs: an array of coefficient vector, and an array of X. The first four input arguments must be provided with non-empty initial guess of the coefficients $beta0$ matrices. Y and X must be the same size as the vector (or matrix) returned by fun. A structure containing estimation algorithm options $options$ can be set using $statset$. Following Matlab compatible options are recognized [61]
1. $TolFun$ Minimum fractional improvement in objective function in an iteration (termination criteria). Default setting is 1e-8.
2. $MaxIter$ Maximum number of iterations allowed. Default setting is 100.
3. $DerivStep$ Step size factor. The default is $eps^{(1/3)}$ for finite differences gradient calculation.
4. Display String indicating the degree of verbosity. Default is set to be "off". Currently only supported values are "off" (no messages) and "iter" (some messages after each iteration).
5. Optional Name, Value pairs can be provided to set additional options. Estimation algorithm specified as a structure using $statset$.
The algorithms of $nlinfit$ are described as follows [61].
1. $nlinfit$ treats NaN values in Y or modelfun(beta0,X) as missing data, and ignores the corresponding observations.
2. For non robust estimation, $nlinfit$ uses the Levenberg-Marquardt nonlinear least squares algorithm, while for robust estimation uses an iterative re-weighted least squares algorithm.

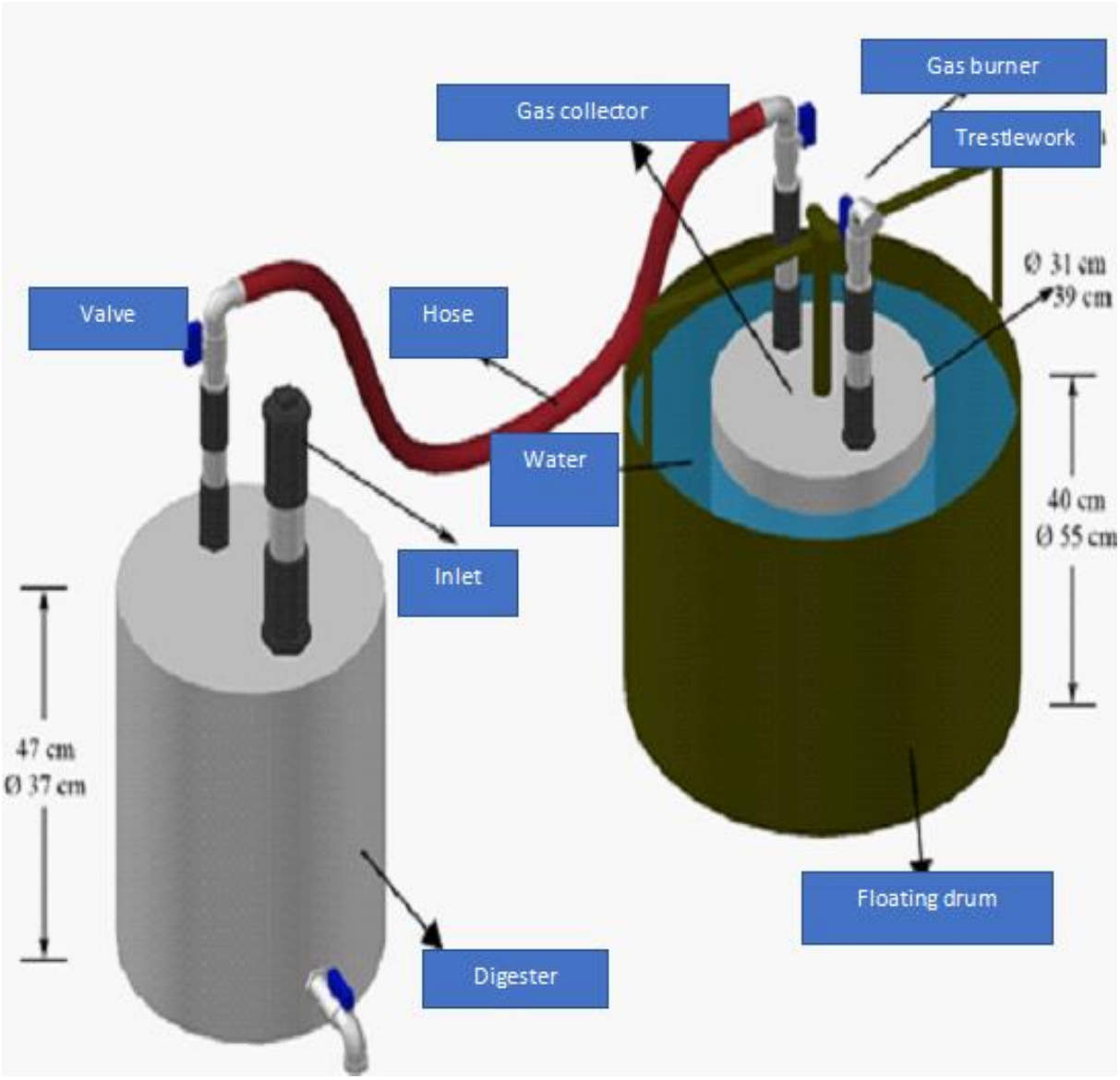


Figure 3.1: Experimental set up for 50 L digester [46]

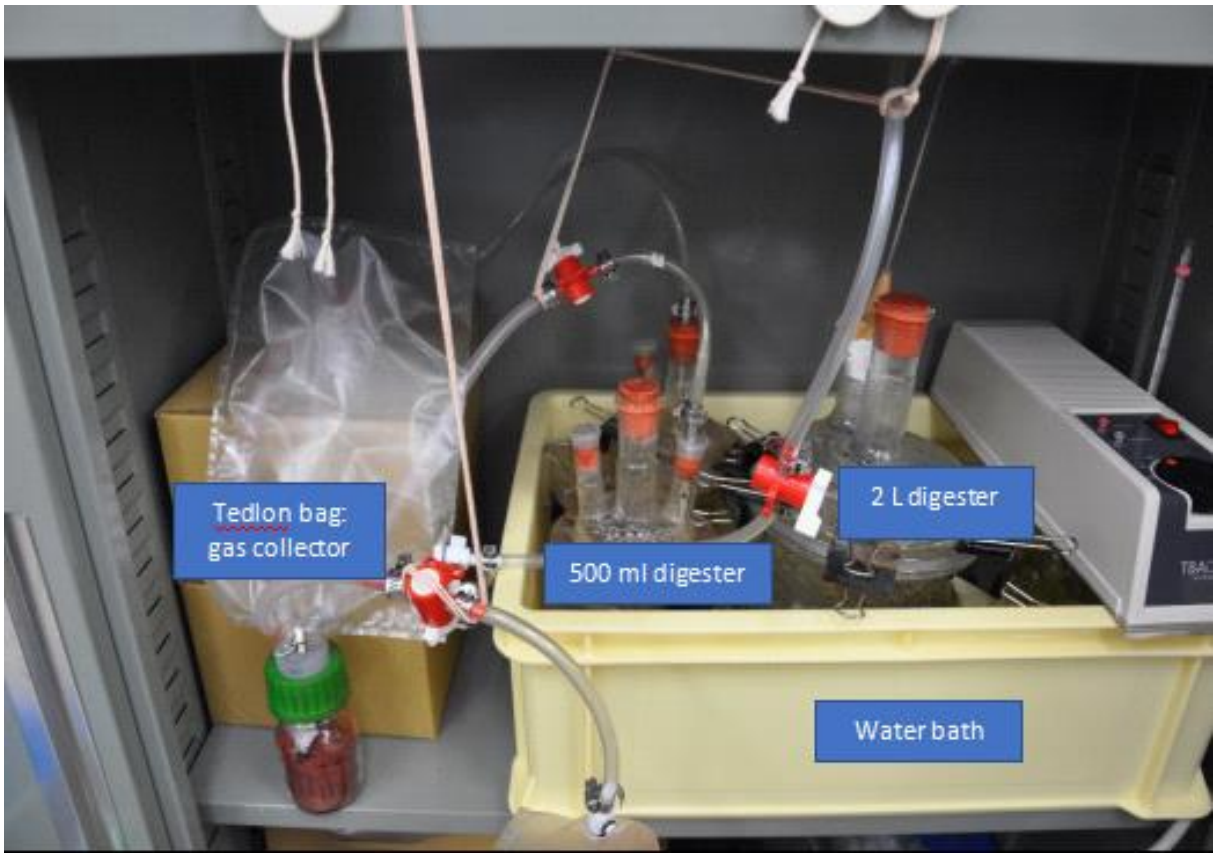


Figure 3.2: Experimental set up for 500ml and 2000 ml digester

Chapter 4

Results and Discussion

4.1 Experimental Results

The experiments of biogas production from fruit and vegetable waste as substrate were performed in a single batch type digester under mesophilic-range temperature. The inoculum used for the experiment were horse dung and sludge from biogas plant located at Tsuyama, Okayama, Japan, and also from a biogas plant located in Kojima, Okayama, Japan. Experiments were performed with three type of digester, including 50 L floating-drum digester, 500 and 2000 ml digester made of glass.

First experiment was done using 50 L batch-floating drum type digester under ambient temperature (27 - 29^{o}C), with fruit and vegetable waste as substrate and horse dung as inoculum. The experiment was done for 30 days. Elemental analysis for C/N and water content was done before experiment start, which each values were 91.96% and 28.73, respectively. pH and temperature was recorded every five days, while biogas volume was recorded every day during experiment. Figures 4.1, 4.2 and 4.3 show the measured biogas volume, value of pH and temperature during experiment, and concentration of methane and carbon dioxide, respectively.

Second experiment was performed with a 500 ml reactor with nine days retention time, which fruit and vegetable waste and horse dung as the substrate and inoculum, respectively. The temperature was set 40^{o}C with water bath. The initial C/N was around 13.58 - 17.87. The value of pH was around 5.9 - 6 during the experiment. Total biogas production was 200 ml and analyzed by a gas chromatography. The results were Nitrogen 86.8%, Carbon dioxide 13.15%, and 0.05% for unknown gas, respectively.

The next experiments were done in a 500 ml reactor under conditions: temperature was set at the range of 30, 35, 40, and 45^{o}C; fruit and vegetable waste was used as substrate, while horse dung and sludge from the biogas plant were used as inoculum. Each experiment was done for 15 days. The amount of each substrate and inoculum was set to be: horse dung 50 ml, and fruit and vegetable waste 300 ml for temperature 30^{o}C; horse dung 100 ml, and fruit vegetable waste 400 ml for temperature 40^{o}C; horse dung 100 ml, fruit and vegetable waste 400, and sludge from biogas plant 100 ml for 45^{o}C. Total biogas volume was 540 ml (30^{o}C), 280 ml (35^{o}C), 250 ml (40^{o}C), and 160 ml (45^{o}C), respectively. Figure 4.3 shows the concentration of biogas for each experiment.

The last experiment was performed with 2 L reactor under mesophilic range (36-38^{o}C) for 30 days retention time, with fruit and vegetable waste and sludge from biogas plant as substrate and inoculum, respectively. Analysis of elemental (CHN) analysis, COD, and

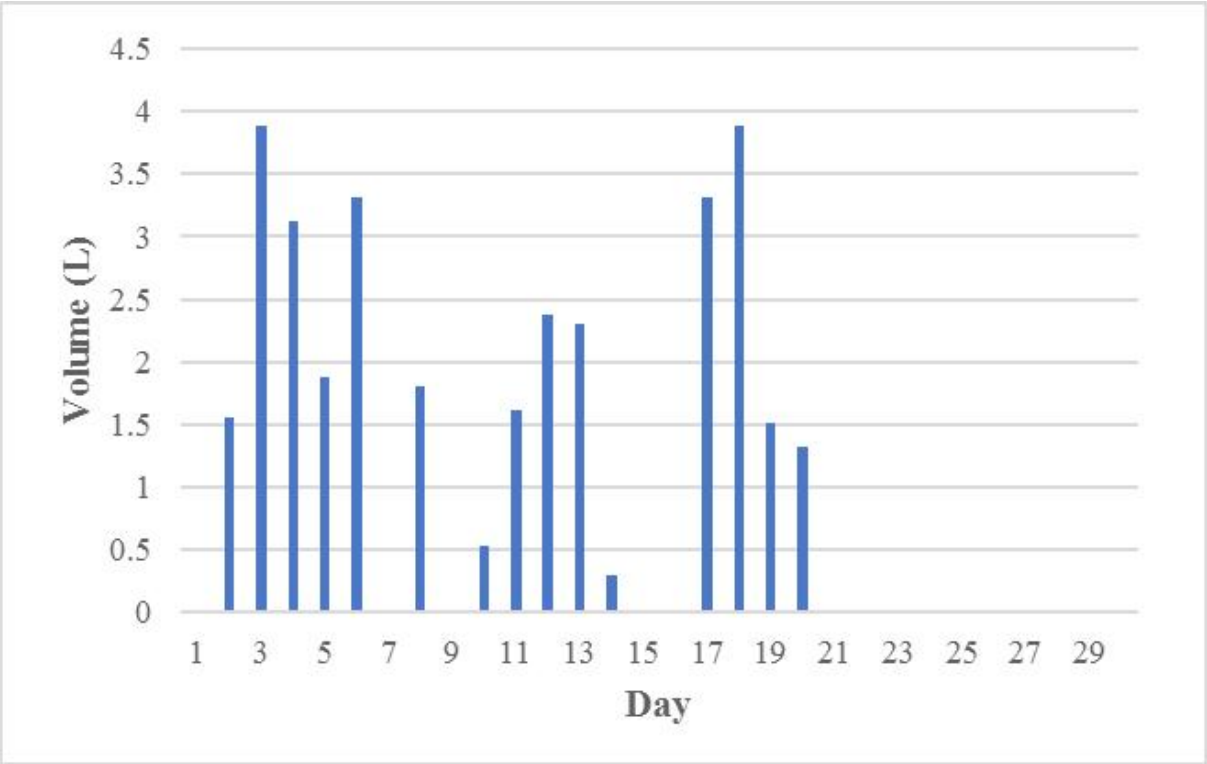


Figure 4.1: Biogas production of 50 L batch digester with fruit and vegetable waste and horse dung as substrate and inoculum, respectively. Daily measurement of biogas volume was recorded (in L) [47]

Table 4.1: Experimental Results under Different Temperature Condition

Temperature	Biogas volme (ml)	Methane (%)	Carbon dioxide (%)
30 ^{o}C	540	0.224	59.122
35 ^{o}C	280	0.093	48.763
40 ^{o}C	250	0.023;1.205	20.95;19.31
45 ^{o}C	160	1.205;2.833	2.216;7.339

TS, was performed before the experiment started, while biogas volume, and concentration of methane and carbon dioxide was performed every six days. Initial COD and TS measured was 12 g/L and 0.107 g/L, respectively. Figures 4.4 and 4.5 show the concentration of methane and carbon dioxide, and volume of biogas measured during experiment, respectively.

Biogas production is the best way to monitor the anaerobic process, which the methane production is a parameter of how well anaerobic process is performed. Figures 4.1 - 4.5 show the performance of biogas production of fruit and vegetable waste with horse dung. Mostly, the production reached maximum on day 19 and slowly decreased until it stopped. The growth rate of methane-generating bacteria is low with regeneration about 5-16 days, while the growth rate of acid bacteria is shorter than methane-generating bacteria with regeneration in about 24-90 h. The low growth rate of the methane-generate bacteria means that a long start-up phase is required, because the amount of inoculating sludge necessary to start the anaerobic digester at full capacity immediately is mostly unavail-

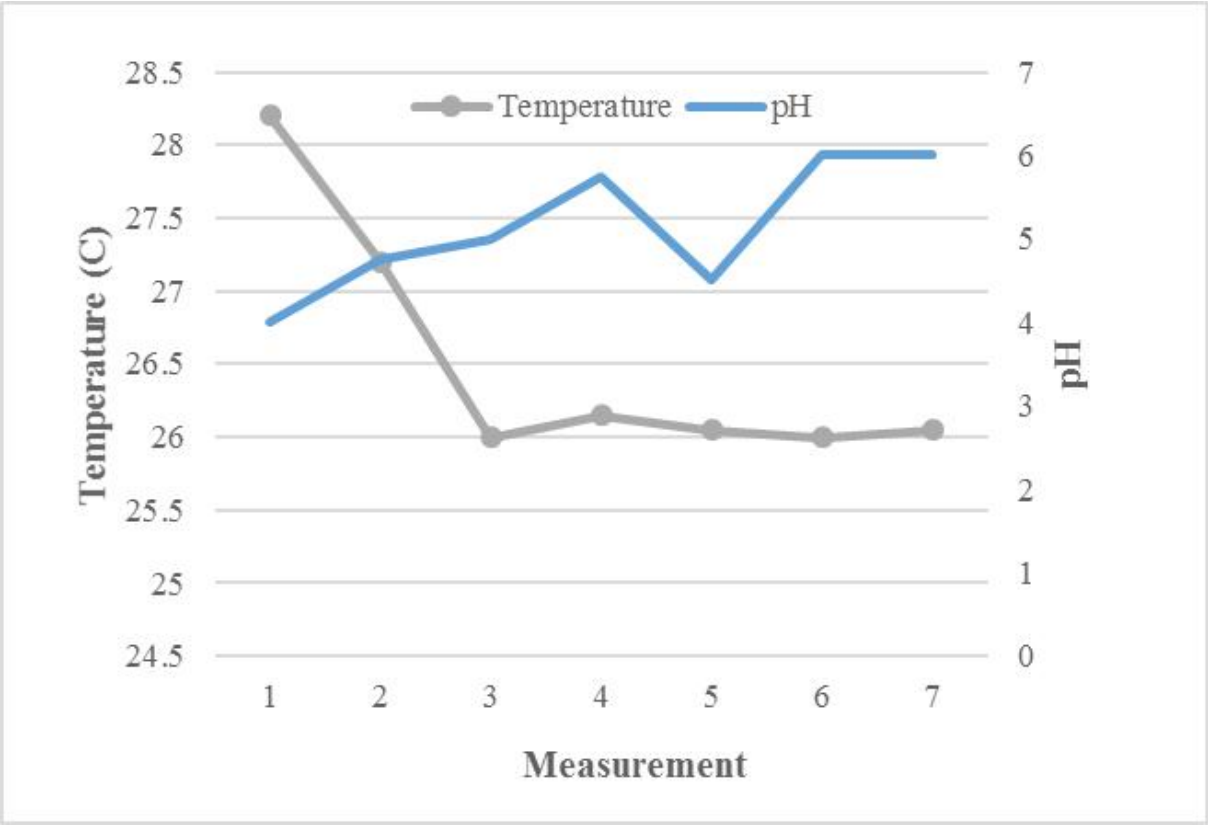


Figure 4.2: pH and temperature recorded from experiment with 50 L digester

able and has to be built up in the starting phase. Figures 4.4 and 4.5 show that the concentration of methane and carbon dioxide during first five days were low, while biogas volume was high. The experimental result suggest that the production of methane still not stable, and that the condition of anaerobic digester including pH and temperature need to be maintained properly during this phase [47, 48].

The inorganic gases nitrogen (N_2) was detected in the biogas about 40-48% (Figure 4.4). Generally, biogas containing nitrogen and oxygen could be introduced when the ventilation is switched on to or the gas pipes are not fully tight. Those gases also can be produced through anoxic respiration (denitrification) in the anaerobic digester. Anoxic respiration can occur during transfer of nitrate ions (NO_3^-) to the digester with sludges or the addition of nitrate-containing compounds such as sodium nitrate ($NaNO_3$) to increase digester alkalinity [35, 37].

Several studies showed that the optimal temperature for biogas production is under mesophilic range [10, 39, 40]. Biogas production and concentration of methane and carbon dioxide were varied under different temperature ranges (30-45^{o}C). The highest biogas volume was produced under 30^{o}C (Figure 4.4).

Generally, anaerobic digestion process improve with longer time of experiment. Toward the end of the residence time, the content of methane increases disproportionately, especially as soon as the carbon dioxide is released, and the hydrolysis stage become deactivated [37]. Figure 4.6 show that there was increase in methane production in comparison with the experiments with shorter retention time (Fig. 4.4 and Fig. 4.5).

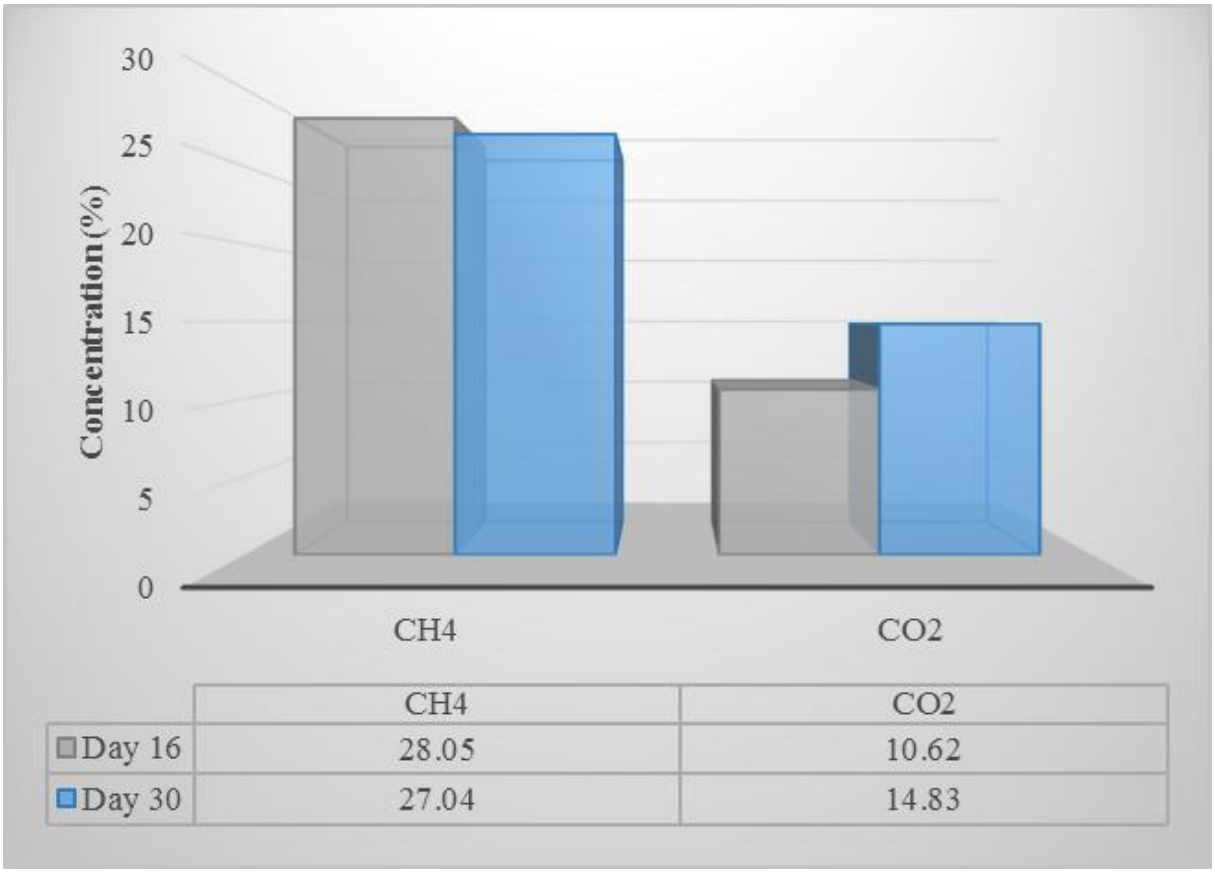


Figure 4.3: Concentration of methane and carbon dioxide measured on day 16 and day 30, performed with 50 L digester

Concentration of methane and carbon dioxide under temperatures between 30 and 35^{o}C were higher than under temperatures between 40 and 45^{o}C (Figures 4.3 and 4.4). A study showed that methane-generating bacteria under temperatures between 40^{o}C and 50^{o}C are easily inhibited [35, 41].
The fluctuations of temperature from 30 to 35^{o}C and 40 to 45^{o}C caused a reduction of biogas volume (Figures 4.3 and 4.4). Fluctuations in temperature affect the activity of methane-forming bacteria to a greater extent than the operating temperature. Therefore, it is necessary to maintain the temperature fluctuations in digester as small as possible, that is, less than 1^{o}C per day for thermophilic and 2-3^{o}C per day for mesophilic [35].
The pH during experiment was around 5 to 6 (Figure 4.1). Fruit and vegetable waste has low pH value which caused rapid acidification and decreased pH in the reactor. If the pH drops below 6.5, methane-generate bacteria will be inhibited. The decrease in pH will also increase the production of volatile fatty acid, which leads to digester failure and decrease in biogas production [12, 35]. Generally, a properly operating anaerobic digester at pH between 6.8 and 7.2 produces volatile acids and then converted to methane and carbon dioxide [35].
Ratio of CN is also one of important parameters need to be considered in the anaerobic digestion of fruit and vegetable waste. CN ratio during experiments was 28-33. Generally, the ratio of CN for anaerobic process should be in the range of 20-30. It is important to

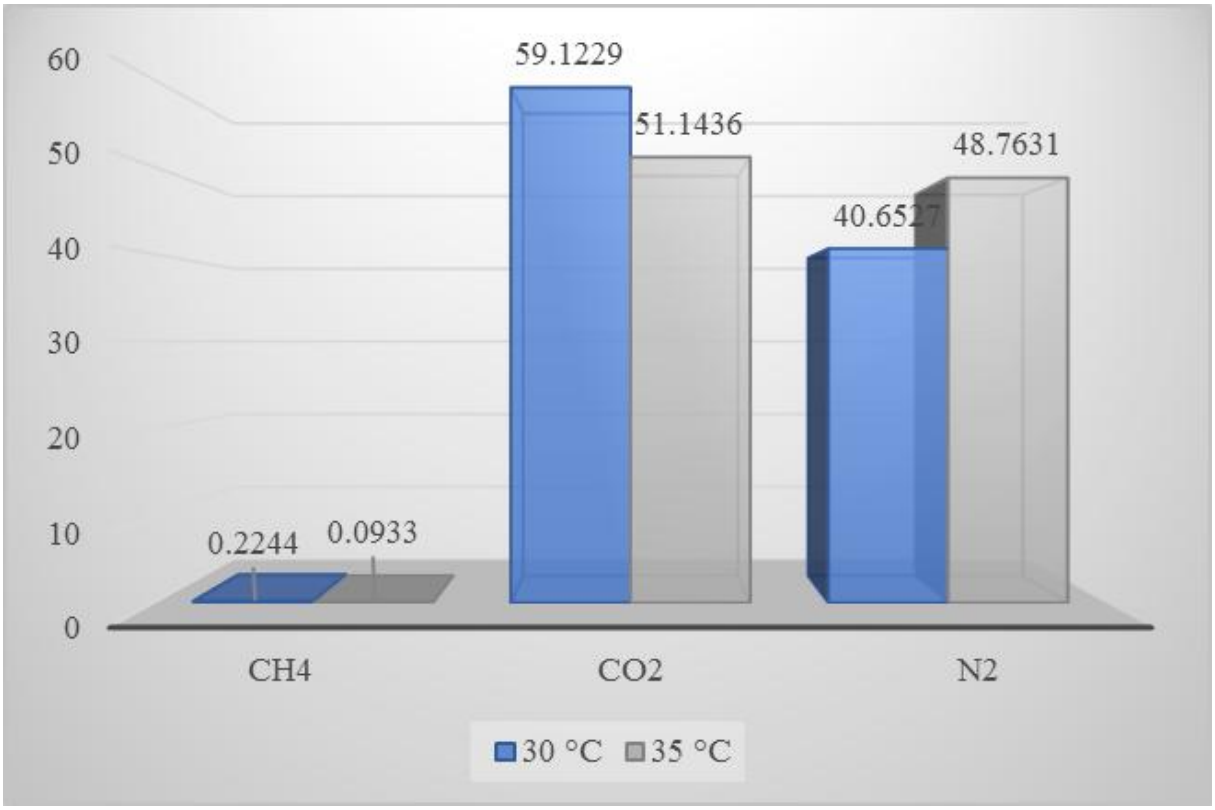


Figure 4.4: Concentration of methane, carbon dioxide, and nitrogen performed with 500 ml digester

maintain an optimal CN ratio, because the most of methane-generating bacteria utilize carbon 25-30 times higher than nitrogen. If the ratio is too high, the nutrients deficiency for bacterial growth will occur and will decrease biogas production [48].

4.2 Numerical Results

4.2.1 The Monod Model

The initial value problem of ordinary differential equations (ODEs) from the model indicate which model by the equation number were solved numerically using the values of the parameters indicate what values of the parameters by the equation number, and the error between the experimental results and numerical results were evaluated. The intervals used in parameters and the optimum values of the parameters are shown in Table 4.2.
The equations were solved numerically with Adams-Bashforth-Molton predictor-corrector methods in conjunction with the Runge-Kutta method to generate the values of numerical solutions at the first three steps [47]. The coded model was to simulate biogas volume, and data fitting has been carried out by trial error methods to find the unknown kinetic parameters values. Figure 4.8 shows the performance of biogas simulation and fit the experimental results.
Methane and carbon dioxide concentration is shown in Figure 4.11, while comparison

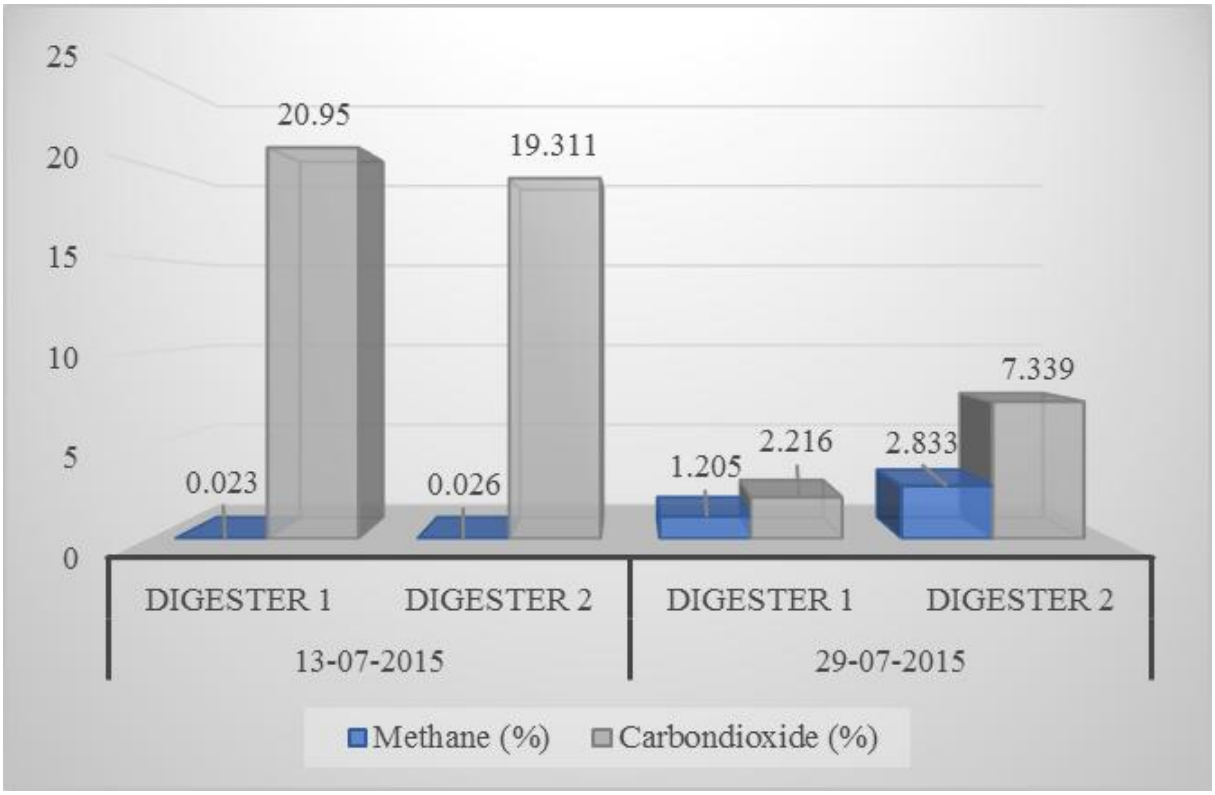
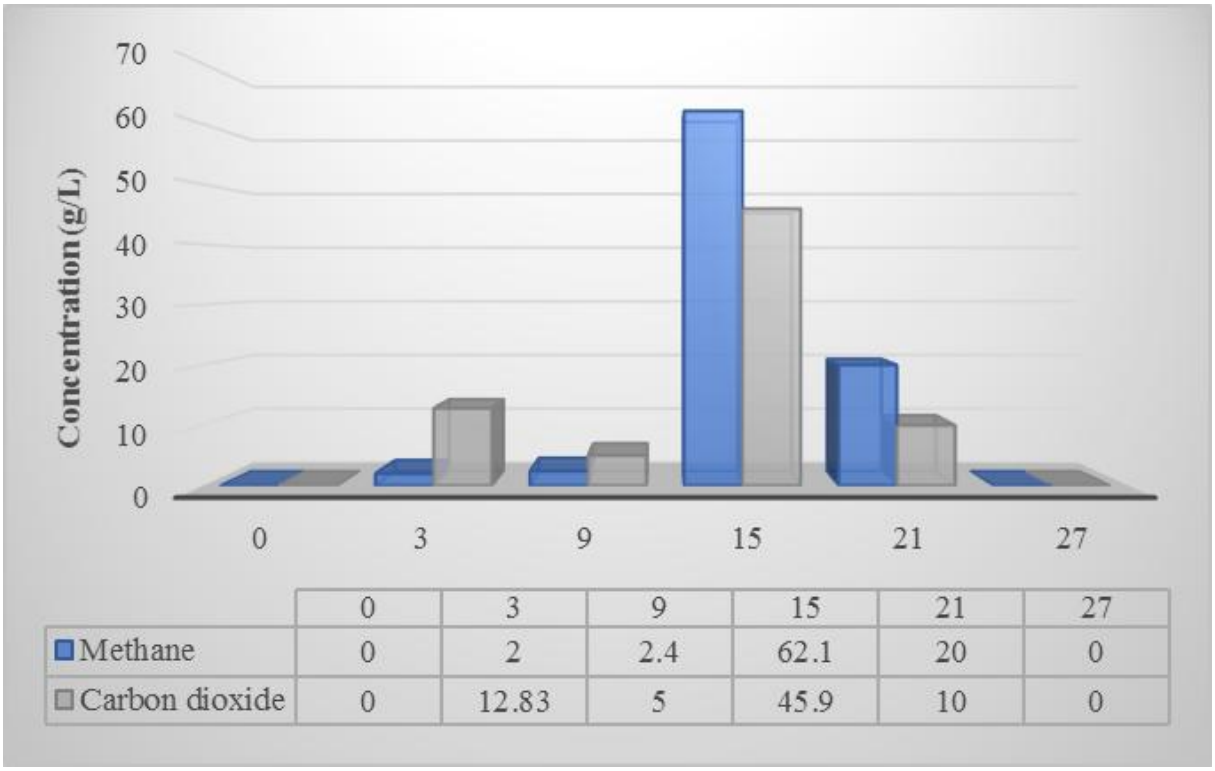


Figure 4.5: Concentration of methane and carbon dioxide performed with 500 ml digester under temperature 40 and 45^{o}C, respectively

between numerical results and experimental results are shown in Figures 4.9 and 4.10, respectively. It shows that numerical results did not fit the experimental results. Practically, there is no reaction that is fully completed, while the models are based on ideal condition of anaerobic digestion without any inhibition. The models need to be improved in order to make the simulation match with the experimental results. The numerical results of biogas production were satisfactory results short period of experiment, however it was unsatisfactory for long period of experiment. It is important to validate the simulation with a good experimental data in order to give best results [47].

Table 4.2: Results of Parameter Estimation Based on The Monod Model [47]

Parameters	Interval	Optimum Value	Units
$E1$	[0.4-0.8]	0.533	J/mol
$E2$	[0.2-1.0]	0.467	J/mol
Ks	[0.01-50.0]	33.337	1/day
Yp	[0.1-10.0]	5.3	g microbial/g substrate
Y	[0.01-20.0]	13.337	g microbial/g substrate
$k1$	[0.1-1.0]	1.0	1/day
$k2$	[0.1-1.0]	1.0	1/day



	0	3	9	15	21	27
Methane	0	2	2.4	62.1	20	0
Carbon dioxide	0	12.83	5	45.9	10	0

Figure 4.6: Concentration of methane and carbon dioxide performed with 2 L digester

4.2.2 The P. Sosnowski Model

The Monod model and first-order model were used for description of the anaerobic digestion processes of fruit and vegetable waste and sludge from biogas plant. Initial value problems of those models were solved numerically. The error between experimental results and numerical results were evaluated, while the kinetic parameters values were determined by trial error methods. Table 4.3 shows the results of parameter estimation. Figure 4.12 shows anaerobic digestion process, and results of comparison between numerical results and experimental results [48].

Figure 4.13 illustrates the performance of the numerical results towards experimental results. The numerical results did not fit the experimental results well. The experimental data showed the generation of methane and carbon dioxide started very slowly from the beginning of experiments, however the numerical results showed biogas generation rapidly increased until it reached the maximum value, and then stopped at the end of the process. This explains that the Monod kinetic is incapable of describing the anaerobic digestion of fruit and vegetable waste [48].

The comparison for both numerical results and experimental results revealed the incapability of the models for description of the processes of complex and heterogeneous substrates, although there were some successful cases [30]. Those models also failed in description of the process under inhibitions. Fruit and vegetable wastes tend to be inhibited in methanogenic phase rather than hydrolysis phase, because of the low pH values of substrates. This fact led to pH decrease in digester and accumulation of volatile fatty acids production, which inhibit the methane-generate bacteria [48, 12].

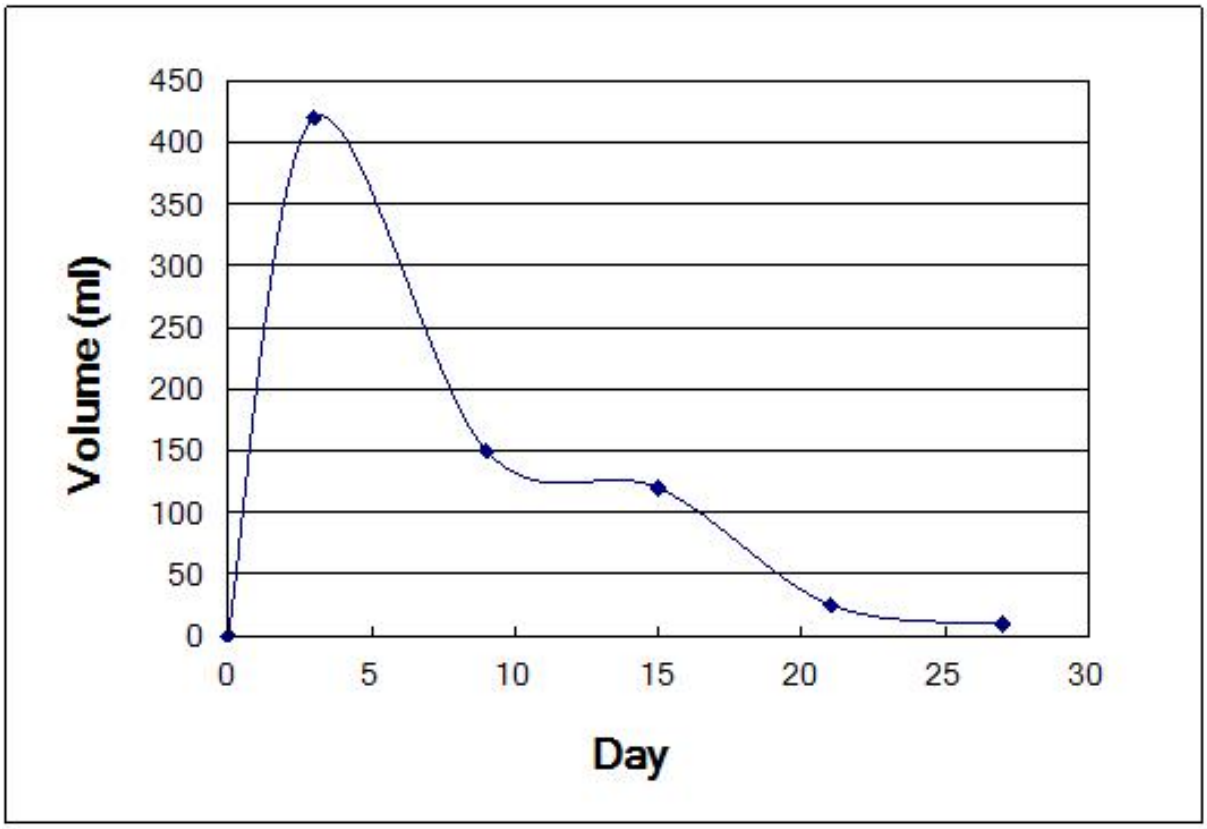


Figure 4.7: Biogas volume performed with 2 L digester [46]

Table 4.3: Description of Kinetics Parameter Used in the P . Sosnowski Model [48]

Kinetics Parameters	Value	Unit
k	0.000208	day^{-1} (* [48])
K_S	11.25	g/l (** [52])
$Y_{VFA/S}$	35.3	- (* [48])
$Y_{CH4/VFA}$	0.46	- (* [48])
V_{VFA}	0.00307	day^{-1} (* [48])
$Y_{CO2/S}$	0.29	- (** [52])
$Y_{CO2/VFA}$	0.34	- (* [48])

4.2.3 The Grau Model

The initial values for the Grau model were obtained numerically, and kinetic parameter estimation was performed. The Levenberg-Marquardt method was applied for estimation, and data from a batch digester experiment were introduced into the estimation. Initial approximations for the parameters were determined, and Table 4.4 shows the results of parameter estimation [49].

The initial concentration of substrate $S(0)$ and initial concentration of bacteria $X(0)$ were expressed as COD and TS, while initial concentration of methane $P(0)$ was zero.

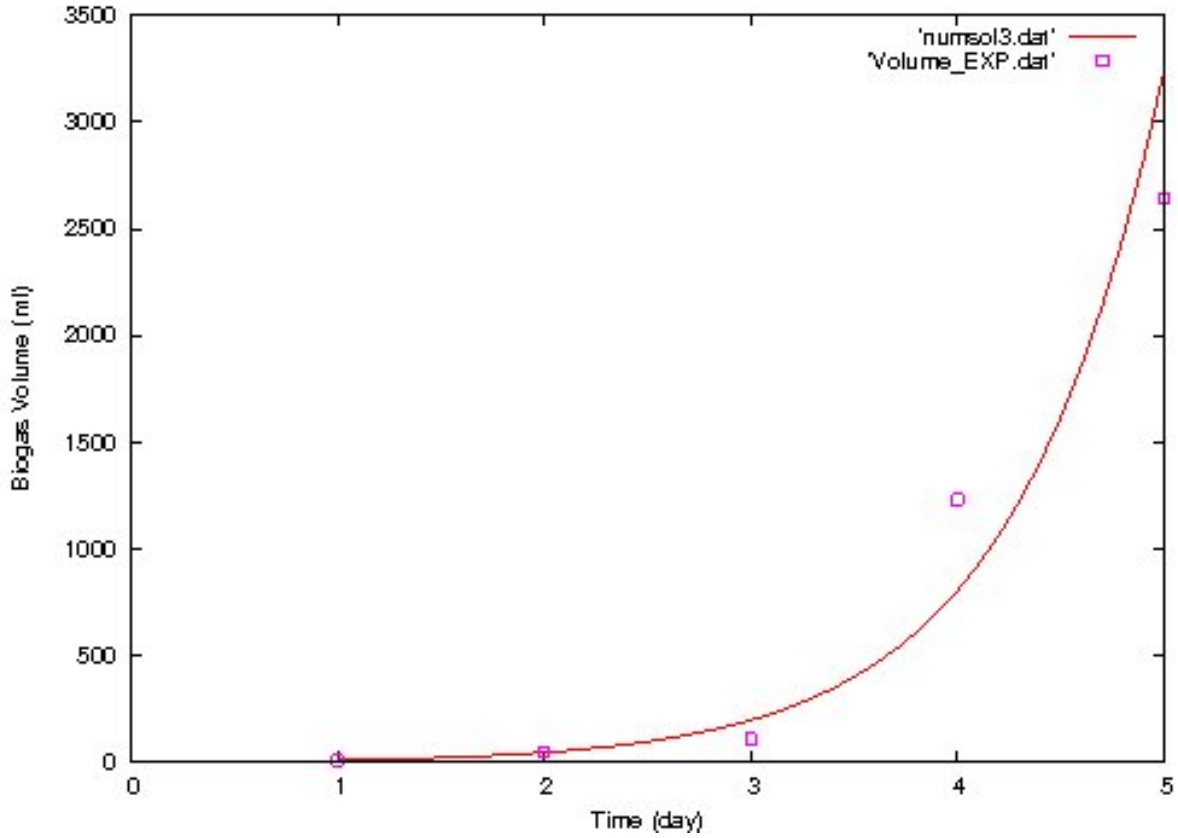


Figure 4.8: Biogas volume performed with 50 L digester: Comparison between numerical results (line) and experimental results (dots) [47]

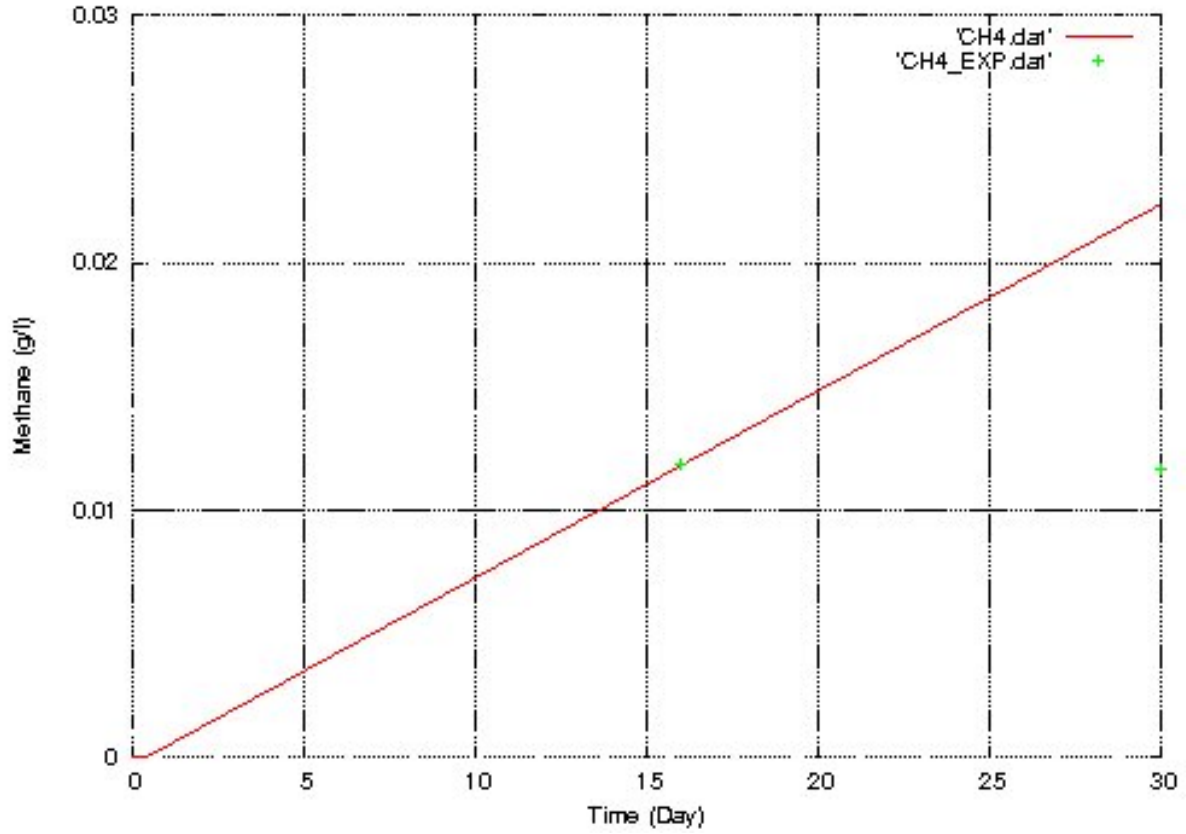


Figure 4.9: Concentration of methane performed with 50 L digester: Comparison between numerical results (line) and experimental results (dots) [47]

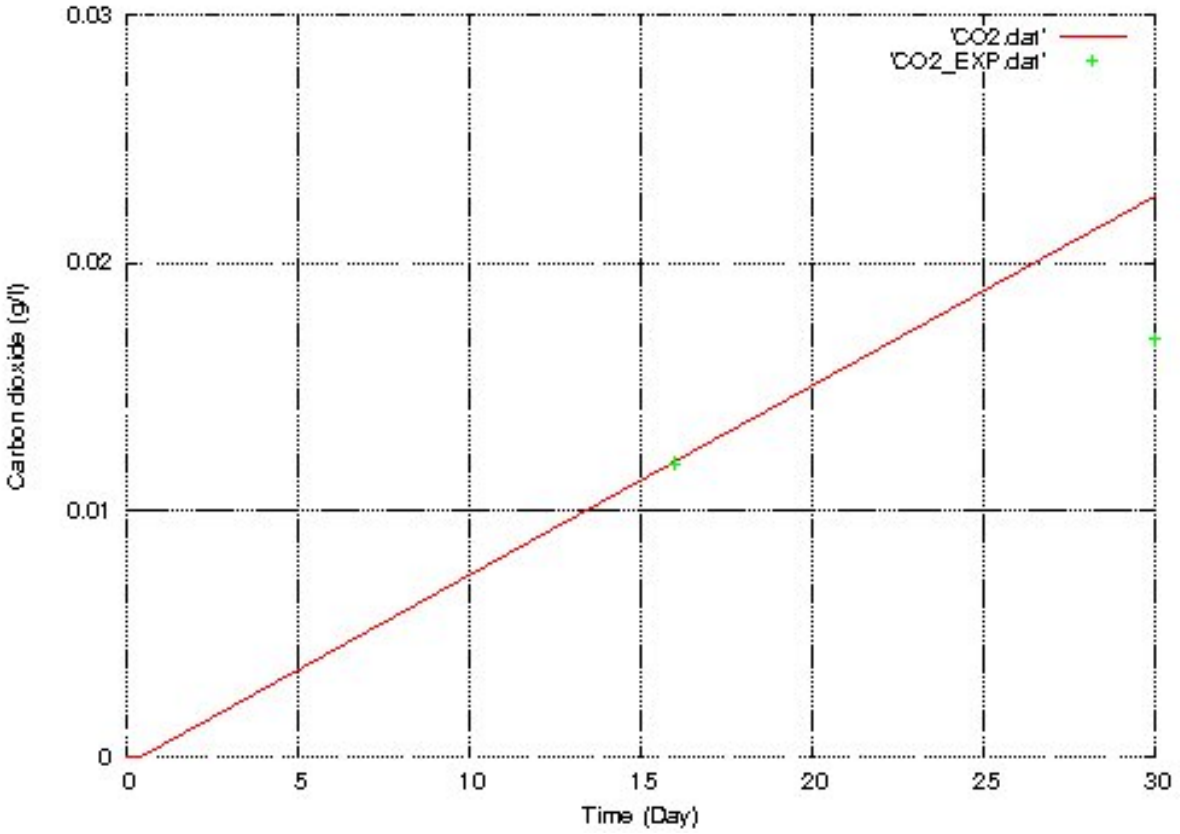


Figure 4.10: Concentration of carbon dioxide performed with 50 L digester: Comparison between numerical results (line) and experimental results (dots) [47]

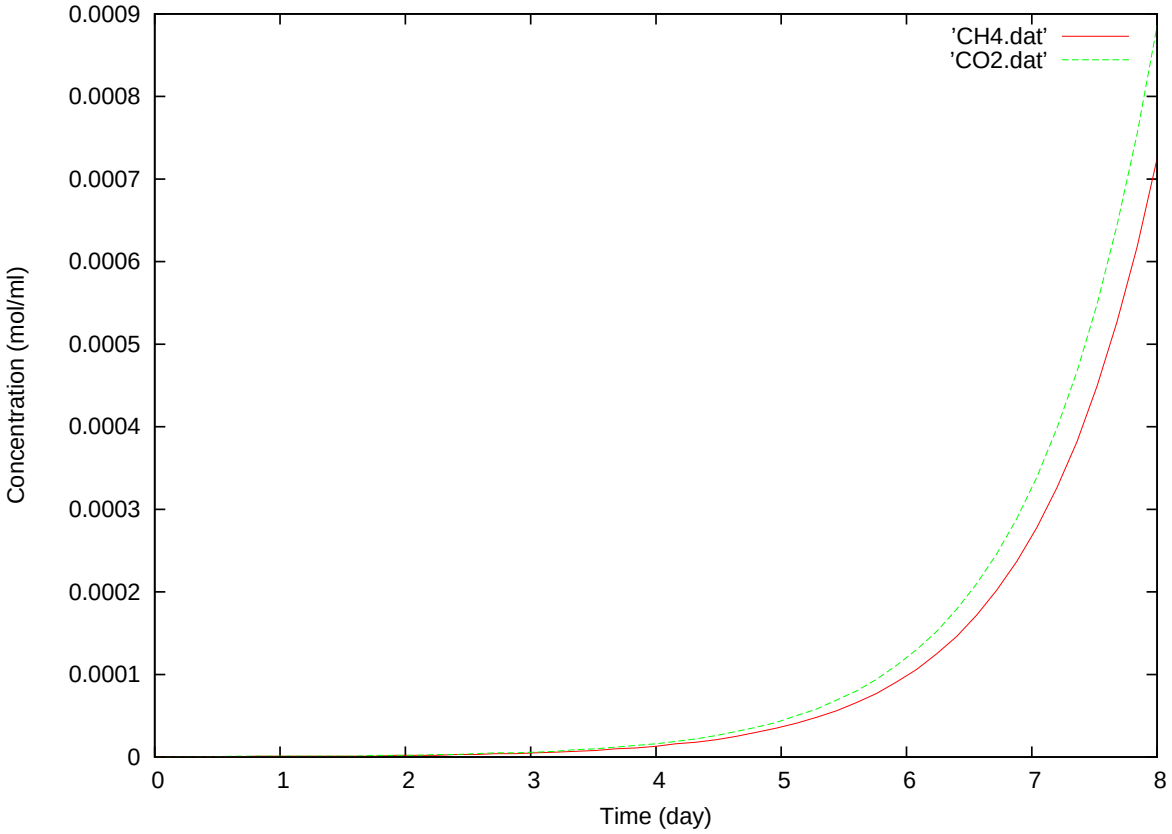


Figure 4.11: Numerical results of methane and carbon dioxide concentration based on the Monod model [47]

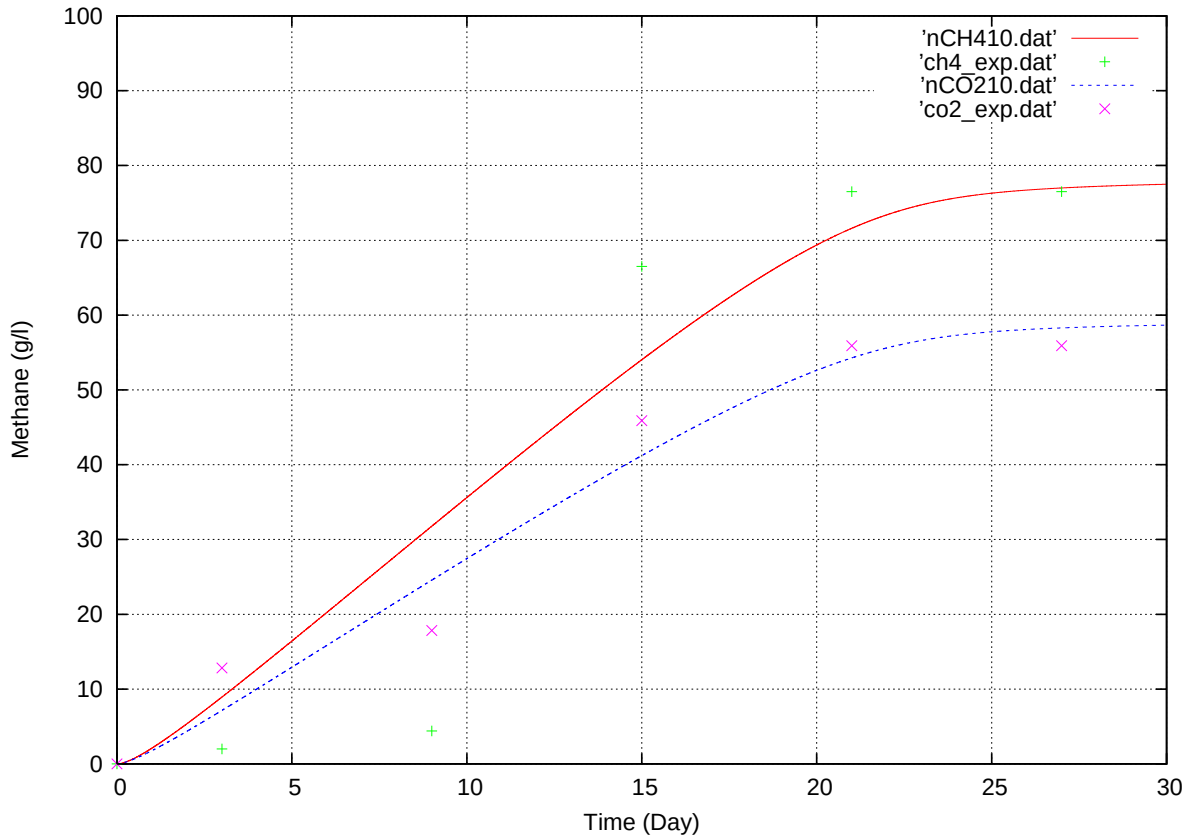


Figure 4.12: Comparison between numerical results and experimental results based on the P. Sosnowski model (line: simulation, dots: experimental [48]

Four parameters $\alpha(Y_S)$, $\beta(Y_P)$, $\gamma(\mu_{max})$, and $\delta(h)$ were introduced as the unknown parameters needed to be estimated. All computations were performed by using MATLAB. The Levenberg-Marquardt method demonstrated good performances for nonlinear least squares approximations. Figure 4.14 shows comparison between experimental and numerical results, while Figure 4.16 shows experimental results [49].

Biogas production slowly increased from day 10 to day 20, which are shown by both

Table 4.4: Results of Parameter Estimation Based on The Grau Model [49]

Parameter	Value	Initial Guess
α	1269.615	90.0
β	0.073	0.3
γ	2.135	2.0
δ	1.266	0.001

experimental results and the numerical results. In this phase, anaerobic digestion process was still in start-up phase. During the start-up, anaerobic digestion was not fully reached, and cultivation of methane-generate bacteria needed longer time for regeneration of methane bacteria and for production of methane. Methane-generate bacteria slowly grow, because the regeneration period of those bacteria is about 14 days [49].

The growth of methane bacteria consists of several phases. The first phase is lag phase,

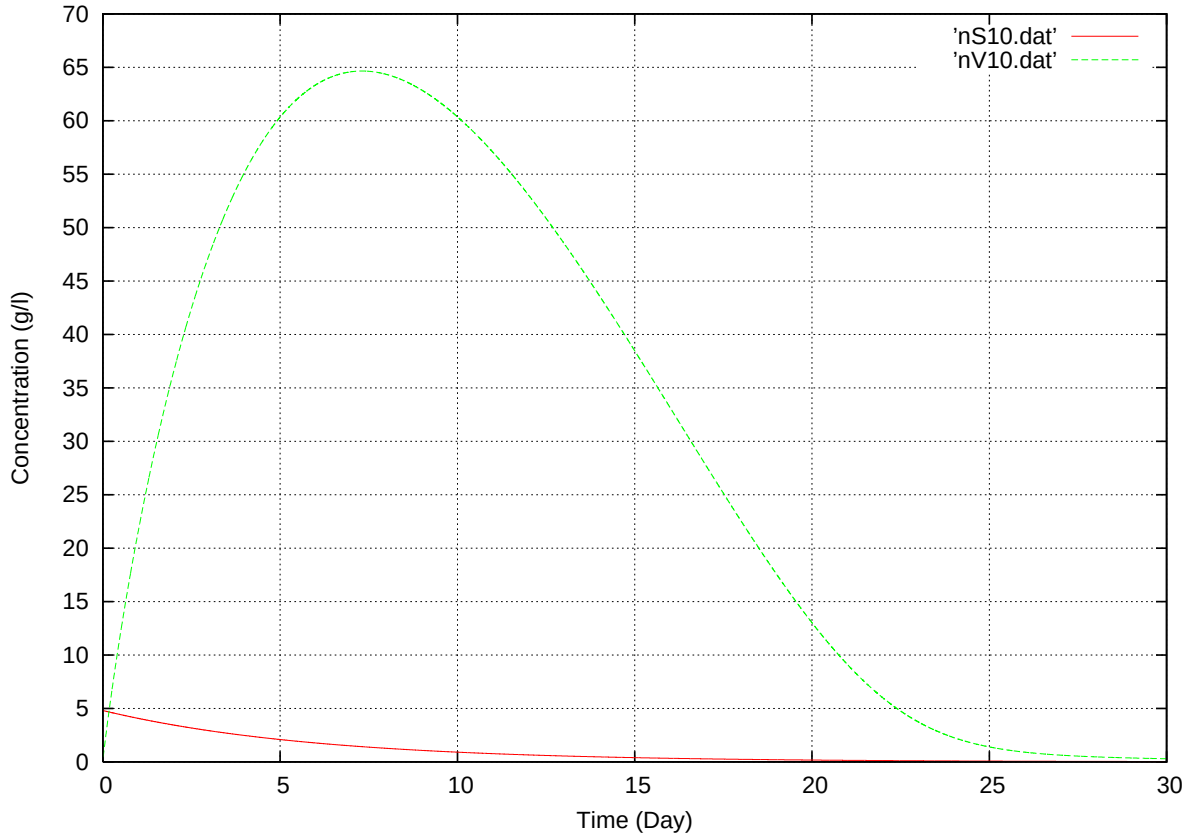


Figure 4.13: Numerical results based on the P. Sosnowski model: Degradation of substrate (red) and volatile fatty acid (green) [48]

where there is no growth of methane bacteria on the first five days (Fig. 4.15). The inoculum of methane bacteria was provided from a biogas plant, where the substrate was different from the one used in the experiment. The difference in the substrates led to condition where the contact rate of methane bacteria to new substrate decreased, and the acceleration growth of methane bacteria also decreased. Thus, the methane production was small in the lag phase [49].
The second phase is the exponential phase. In this phase, the rate of bacterial growth is in a steady state. The rate will change if nutrients are depleted, toxic products are accumulated, and the pH value changes due to substrate degradation. This phase appeared during day 6 to day 16, where the maximum growth of methane bacteria and methane generation was reached during day 14 to day 15 [49].
The last two phases are the retardation phase and the stationary phase. During the retardation phase, is decreased as the substrate reduces. Thus, methane still produced in a small amount during the retardation phase of day 15 to day 20. Further, there was no methane generation due to decay of methane bacteria and depletion of substrate day 20 to day 30 (Fig. 4.15) [49].
The major limitation of fruit and vegetable wastes is the rapid acidification due to lower pH value. This will lead to digester failure and decrease in biogas production, because methane-generate bacteria tend to be inhibited by pH below 5. It is important to maintain the optimal pH for anaerobic process, so that inhibition of methane-generate bacteria

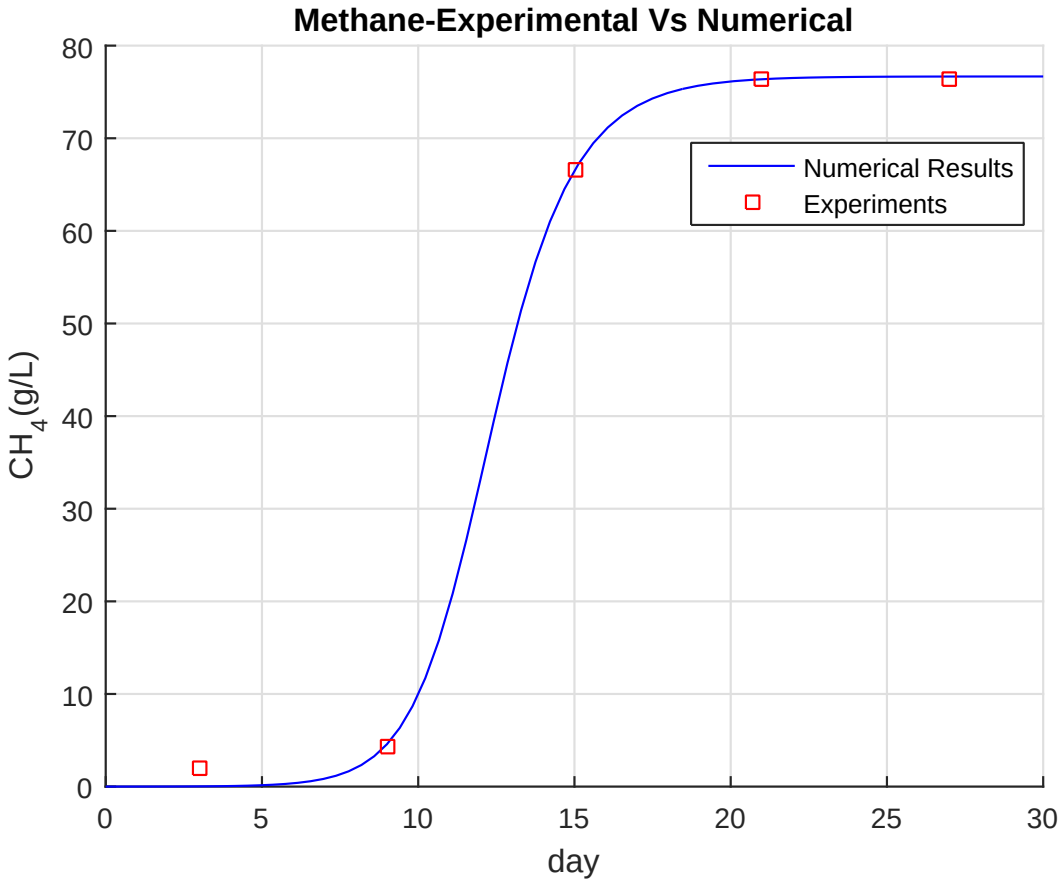


Figure 4.14: Numerical results based on the Grau model: comparison of numerical results with experimental results [49]

could be prevented [49].

4.3 Discussions

4.3.1 Limitation of Experiments

Alkalinity and pH

Alkalinity play an important role as a buffer that prevents rapid change in pH. The composition and concentration of the substrate influence the alkalinity and pH of the digester. Generally, fruit and vegetable wastes have a low pH value and they tend to be acidified rapidly. When organic compounds are degraded, carbon dioxide is released, which results in production of carbonic acid, bicarbonate alkalinity, and carbonate alkalinity. The equilibrium of those components are a function of digester pH. However, the degradation of organic compounds of fruit and vegetable wastes produces organic acids that destroy alkalinity, which is not recovered until methane generation occurs in the digester [35, 11]. The pH during experiments was around 4 to 6, while the optimal pH range for methane generation was within range of 6.8 to 7.2. pH values below 6 are toxic to methane-generate

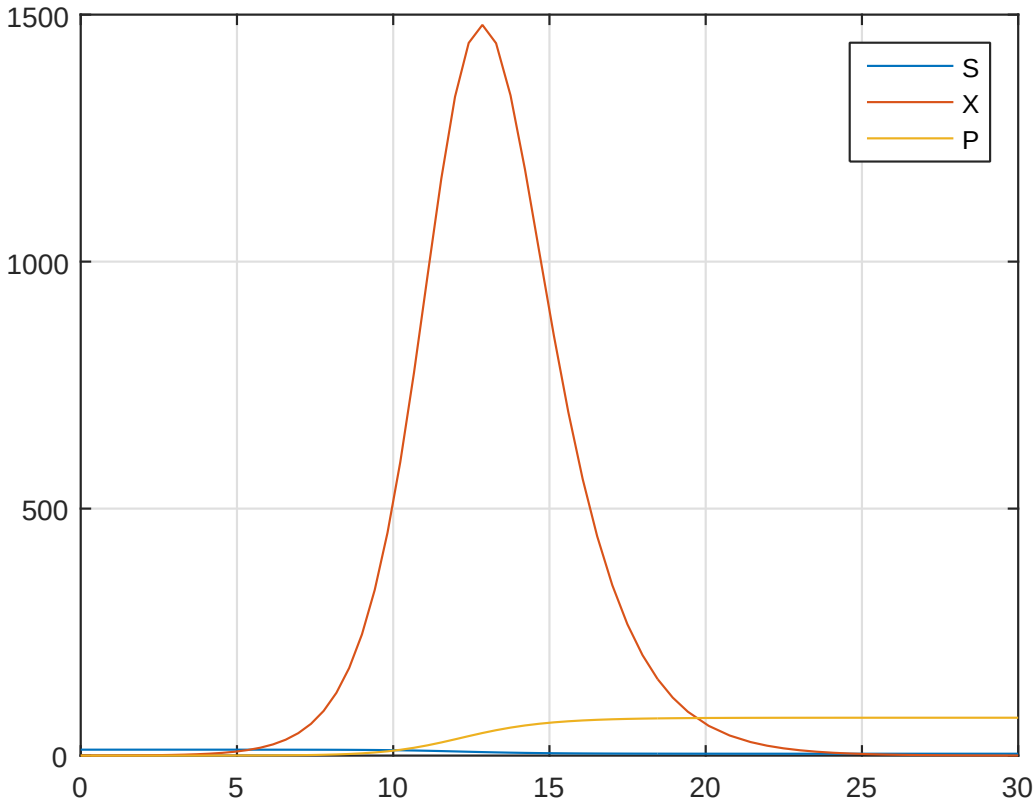


Figure 4.15: Numerical results based on the Grau model: S denotes as Substrate, X denotes as bacteria, and P denotes as methane [49]

bacteria. It is important to maintain the level of alkalinity for stabilization of pH in the digester. Sodium bicarbonate and potassium bicarbonate are preferred for pH level adjustment in an anaerobic digester, because methane-generate bacteria require bicarbonate alkalinity as the primary carbon source. Besides, sodium and potassium are the least toxic for bacteria [35].
The pH value of medium in the anaerobic digester is controlled by the concentration of carbon dioxide in the gas phase, and the concentration of bicarbonate-alkalinity in the liquid phase. If the concentration of carbon dioxide in the gas phase remains constant, the possible addition of bicarbonate-alkalinity can increase the pH [55]. The experimental results concerning biogas production from fruit and vegetable waste with horse dung showed that carbon dioxide concentration increased at day 30, while the concentration of methane decrease (Fig. 4.3). The pH value during experiment continuously decreased until the end of experiment (Fig. 4.2). A drop of pH value and rise of concentration of carbon dioxide in the biogas is an indication of disturbance against the anaerobic digestion process, which was caused by increasing of acid in the digester [37].

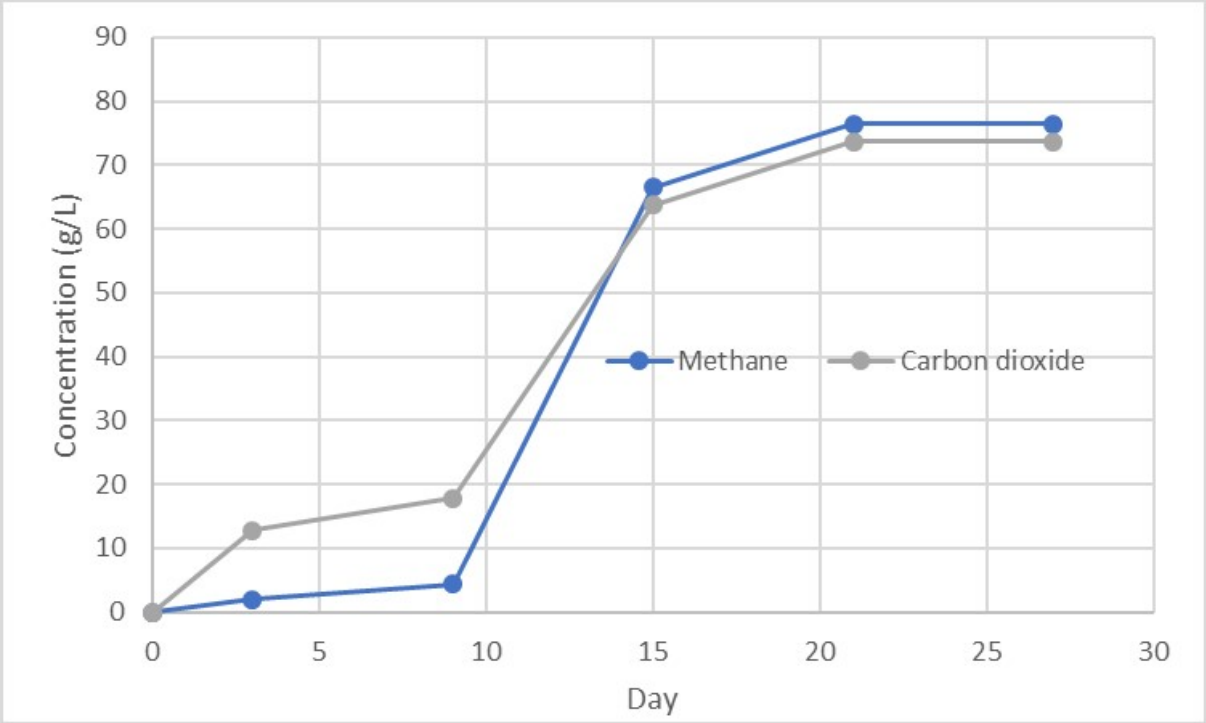


Figure 4.16: Accumulated methane and carbon dioxide concentration generated from 2 L batch anaerobic digester with fruit and vegetable waste and sludge from biogas plant as substrate and inoculum, respectively [49]

Ammonia and volatile fatty acids production

Ammonia is produced by the degradation of organic nitrogen compounds, such as amino acids and proteins. The quantity is affected from a complex substrate, such as co-digestion process of fruit and vegetable waste with horse dung and sludge from biogas plant. Ammonium ion (NH_4^+) and free ammonia (NH_3) are the two essential forms of inorganic ammonia nitrogen in an anaerobic digester [35, 41]. Ammonium ion is required as a nutrient source for bacteria in an anaerobic digester, while free ammonia is the main cause of inhibition [35]. When ammonia inhibition occurs in a digester, the toxic effect on methane-generate bacteria will cause accumulation of volatile fatty acid, which leads to a decrease of pH value and reduces the concentration of free ammonia [41, 42].
The production of volatile fatty acids during anaerobic digestion process tends to reduce the pH value. This reduction is countered by the activity of the methane-generate bacteria, which also produce alkalinity in the form of carbon dioxide, ammonia and bicarbonate [55]. The relationship between free ammonia, volatile fatty acid, and pH will leads to a condition where the anaerobic digestion process is stable with a low methane production [41, 42]. It is appropriate to conclude that it is essential to maintain the concentration of free ammonia and volatile fatty acid in anaerobic digester for prevention of process failure.

Performance of batch single-stage digester

A batch reactor is a typical operation of anaerobic digestion process that is used extensively due to simple design, simple process control, and low investment costs. However, some studies reported that batch systems inhibited by the accumulation of inhibitory products and decreasing pH [11]. Application of sequencing batch reactor technology, or multiple-stage reactor technology can be applied for anaerobic digestion of fruit and vegetable waste. Due to drawback of fruit and vegetable waste samples described in the above, some studies reported that those technologies are more applicable [11].
The digester was not equipped with a proper mixing, which is important for anaerobic digestion with particulate matters. Mixing minimizes the floating of particles and reduces the accumulation of undesirable materials, which reduces the digester performance. Undesirable materials appear when gas bubbles are entrapped in a liquid. The surface tension of the liquid or the sludge is reduced, which results in accumulation of solids over entrapped gas bubbles. An adequate mixing must be ensured to prevent the accumulation of undesirable materials [35].
Mixing also provides efficient hydrolysis of wastes and production of organic acid by acid-generating bacteria. However, it is important to provide a proper mixing in the digester, because it may influence the methane-generating bacteria. The continuous rapid mixing may disturb the methane-generating bacteria in the digester. Scheduled mixing of digester is more applicable than continuous mixing, because it is costly and requires a specific facility [35].

4.3.2 Limitation and Applicability of Proposed Models

Anaerobic digestion process is a multi-step process which involves multiple microbes. This process usually contains a particular step which called rate-limiting step. The rate-limiting step causes process failure to occur under stressed conditions [53]. Determination of a rate-limiting step is critical for anaerobic digestion process design treating a specific substrate, establishment of a stable process performance, and management of anaerobic digestion process [54].

The Monod model

The rate-limiting step for the Monod model was assumed to be hydrolysis stage, acidogenesis stage and methanogenesis stage. The Monod model describes substrate concentration as limiting factor, which limits the growth of microbes due to its concentration [36]. In the acidogenesis stage and methanogenesis stage, the concentration of volatile fatty acid was assumed to be the limiting factor of the biogas production. Volatile fatty acid is produced as the end products of bacterial metabolism of complex substrates containing fat, protein, and carbohydrate [48]. The overloading volatile fatty acid in one digester increase the production of volatile fatty acid which inhibits the process of hydrolysis, acidogenesis, and methanogenesis, and later the entire process becomes the overall anaerobic digestion rate-limiting step [51].
This model also incorporates the influenced of temperature to the growth of bacteria. The model was based on the pronciple of Arrhenius law modified by Bergter (1983) and Sinclair & Kristiansen (1993) [36]. However, the Arrhenius law was not valid for describing

the anaerobic process of fruit and vegetable waste. The applicability of the model implies for an empirical description only [36].

The P. Sosnowski model

The rate-limiting step for the P. Sosnowski model was assumed to be hydrolysis stage and methanogenesis stage, on which the first order model and the Monod model are based [48]. The hydrolysis stage as rate-limiting step takes place the degradation of complex organic particulate, such as sewage sludge. In this stage, complex substrates should be decomposed to soluble compounds before they are metabolized by microbes [51]. The numerical results showed the poor applicability of the P. Sosnowski model to simulation of the anaerobic digestion of fruit and vegetable waste. The arduous task of determining kinetic data for simulation of the anaerobic digestion process in the acetogenesis stage and methanogenesis stage conversion has frustrated the implementation of this model.
The hydrolysis stage as the limiting-rate failed description of the anaerobic digestion process with fruit and vegetable as substrate (Fig. 4.12). Fruit and vegetable waste is an easily biodegradable waste which has low pH, and it tends to be limited by methanogenesis than hydrolysis, although the anaerobic digestion with particulate sample tends to be inhibited by hydrolysis.

The Grau model

The Grau model incorporates substrate degradation rate of multi-component substrate based on the linear degradation proposed by Monod. This model was used for description of anaerobic digestion process of fruit and vegetable waste with sludge from biogas plant. The results (Fig. 4.14 and Fig. 4.15) showed that this model demonstrated a good numerical agreement with the experimental data. However, the model failed in simulation of the start-up phase of anaerobic digestion process. Fruit and vegetable wastes are easily degradable substrates. Figure 4.14 shows that biogas already produced during the first five days.
The methanogenesis stage as the rate-limiting step of the production of methane from fruit and vegetable waste was assumed. Several studies reported that anaerobic digestion of fruit and vegetable waste is limited by methanogenesis stage rather than hydrolysis [11]. Fruit and vegetable waste has a low pH value which tends to inhibit the methane-generate bacteria, because those bacteria have the highest sensitivity to the low pH and lowest growth rate compared to other bacteria involved in the process [51]. However, the applicability of the models is limited by the difficulty due to determination of kinetic data for anaerobic bacteria on the acidogenesis stage to methanogenesis stage [51].
The model with rate-limiting approaches assumes that biogas production can be predicted by one single step during an anaerobic digestion process, which leads to simple and readily applicable models [54]. However, simplicity of the models may not be appropriate for description of anaerobic digestion of complex substrates, such as co-digestion process of fruit and vegetable waste with horse dung and sludge from biogas plant. Influences of ambient conditions such as pH and temperature, inhibition by volatile fatty acid and ammonia cannot be determined by the models. It is difficult to determine those parameter when only a rate-limiting step or single bacteria are accounted for the whole process of anaerobic digestion [51].

Trial error methods for parameter estimation

Inverse problems for methane generation from fruit and vegetable waste were formulated. Parameter estimation were performed by a trial-error method. Parameter fitting was carried out by minimizing the error between experimental data and numerical data. The initial concentration of substrates, microbes, and biogas were required for the Monod model and the P. Sosnowski model. Table 4.2 and Table 4.3 shows the numerical results. Figures 4.8 - 4.13 show the numerical results with experimental data.
The results showed that the applicability of this method for estimation of kinetics parameter. However, this method is very time consuming and does not provide any information about the uncertainty and uniqueness associated with the parameter values. In order to avoid tedious trial-error approach, optimization algorithms were proposed. Those techniques approach the optimum parameter values by optimizing an objective function.

The Levenberg-Marquardt method performances

The modified Grau model was proposed for description of the generation of methane from anaerobic digestion of fruit and vegetable waste. The experimental data were introduced into the model. There are four unknown kinetic parameters. It is important to choose appropriate values of the parameters in order for the numerical results match to the experimental data. The Levenberg Marquardt method was applied for least square approximation between numerical results and experimental data.
In this method, it is important to choose appropriate initial approximations. Figure 4.14 demonstrates an appropriate fit between the numerical results and experimental results, and Table 4.3 shows the estimated parameter values with values of the sum squares error (SSE) reached 3.9 after 150 iteration. This values could explain that the model was not consistent with the experimental data. The value of SSE closer to zero indicates that the model has a smaller random error component, and that the fit of the model will be more applicable for prediction of anaerobic digestion process.
Nonlinear least square problems can have objective functions with multiple local minima. Fitting algorithms make approximate solutions converge to different local minima depending upon values of the initial guess, the measurement noise, algorithmic parameters. In the absence of physical insights, reasonable initial guess may be found by coarsely griding the parameter space, and finding the best combination of parameter values. In this method, accurate starting point of initial guess is needed for best performance. However, this method involves a gradient-descent method and thus iteration can get stuck in local minima.

Chapter 5

Conclusion

This study focuses on mathematical models and numerical simulations of anaerobic digestion process with fruit and vegetable waste. Those models performed with different kinetics model describing the degradation rate of substrate, the growth of microbes, and methane generation rate. Laboratory scale experiments were carried out to record accumulated methane concentration, carbon dioxide concentration and biogas volume. Those outcomes from the experiments were introduced to inverse problems of methane generation. Major conclusions obtained from the studies are summarized.

5.1 Modeling Anaerobic Digestion of Fruit and Vegetable Waste

In recent years, anaerobic digestion process has become the promising method to treat solid waste, because of many advantages such as production of renewable energy and valuable bio-products. It has been applied to many types of waste such as municipal organic waste, fruit and vegetable waste, agricultural waste, etc. With a growing attention to this process, variety of anaerobic digestion systems have been developed. Mathematical modeling and numerical simulation provide an excellent tools for enhancement of anaerobic digestion process.

Along with the development of mathematical models since 1940's to 2000, interest in anaerobic digestion and its simulation, especially in solid waste has been rapidly developing. However, the limited studies in modeling of anaerobic digestion with fruit and vegetable waste as substrate, and the unsteady characteristics of the substrates proposed challenging problems.

Important aspects and outcomes of this study are:

1. According to experimental results, anaerobic digestion of fruit and vegetable waste are more sensitive to pH inhibition due to the low pH values of the wastes. It is important to pretreat waste by adding buffer to increase the pH, and to maintain pH value during experiments.
2. Batch single stage reactor for fruit and vegetable waste tends to be sensitive to inhibition during an anaerobic process due to accumulation of inhibitory product, such as volatile fatty acid. The problem can be solved by introducing two or multiple stage reactor to separate the acidification stage and methane-generation stage.
3. Monod model and first-order model were applied for simulation of anaerobic digestion

process. Inverse problems were analyzed and experimental results were compared. The results showed that those models were incapable of describing the process under inhibitions.
4. The modification of Grau model was proposed, and it was implemented in Matlab. An acceptable agreement between numerical results and experimental results demonstrated the applicability of the model.
5. The trial error method was used to determined the kinetic parameters of proposed models. The results showed that this method is time consuming. In order to avoid tedious trial-error approach, the Levenberg-Marquardt method was applied. The Levenberg-Marquardt method demonstrated an excellent performances for the parameter estimation.

5.2 Further Studies

The anaerobic digestion of fruit and vegetable wastes were easily inhibited by the ambient condition, such as pH and temperature. Besides, the low pH value of fruit and vegetable waste tends to produce more acid during process which leads to the decrease of pH. The production of ammonia and volatile fatty acids are also influences by the pH values. Based on the limitation described, there is a need for further research and improvement of the mathematical models to establish the link between pH and inhibition by volatile fatty acid and ammonia. With additional knowledge, the inhibition can be predicted and the performance of biogas production assessed. Integrated models will promote their applications in full-scale plant design, operation and optimization.

Appendix A

Monod Model

```
1   #include <stdio.h>
2
3   #include <math.h>
4
5   double F1(double, double, double, double);
6   double F2(double, double, double, double);
7   double F3(double, double, double, double);
8
9
10    int           n,i;
11    double        S, X, P;
12    double        X_0;
13    double        h,xn,t16,t30,t0,y0_,x0=0.0;
14    double        tTEMP,STEMP,XTEMP,PTEMP;
15    double        ff10,ff11,ff12,ff13,ff14;
16    double        ff20,ff21,ff22,ff23,ff24;
17    double        ff30,ff31,ff32,ff33,ff34;
18
19    double        dSdt,dXdt,dPdt;
20    double        Ks,k1,k2,Y,Yp,E1,E2;
21    double        k11,k12,k13,k14;
22    double        k21,k22,k23,k24;
23    double        k31,k32,k33,k34;
24    double        eta10,eta11;
25    double        eta20,eta21;
26    double        eta30,eta31;
27    double        eta40,eta41;
28    double        X_0=4.0;
29    double        XINIT=4.0;
30    double        SINIT=4.8;
31    double        PINIT=0.0;
32    double        R=8.314,T=300.0;
33
34  #define number 400
```

```
35
36  int main(void)
37  {
38    int j, j1, j2, j3, j4, j5, j6, i16,
39      jks,jk1,jk2,jy,jyp,je1,je2,
40        num_ks = 3, num_k1 = 3, num_k2 = 3, num_y = 3, num_yp = 3,
41         num_e1 = 3, num_e2 = 3;
42    double      y1[number],y2[number],y3[number],yy[number];
43    double      time;
44    double      tTEMP, STEMP, XTEMP, PTEMP;
45    double      tTEMP2, STEMP2, XTEMP2, PTEMP2;
46    double      tTEMP3, STEMP3, XTEMP3, PTEMP3;
47    double      ch4_day16 = 11.883, ch4_day30 = 11.624;
48    double      error, errortemp;
49    double      SMIN, XMIN, PMIN;
50    double      ks_min = 0.1, ks_max = 1.0, ks_best;
51    double      k1_min = 0.1, k1_max = 1.0, k1_best;
52    double      k2_min = 0.1, k2_max = 1.0, k2_best;
53    double      y_min = 0.1, y_max = 1.0, y_best;
54    double      yp_min = 0.1, yp_max = 1.0, yp_best;
55    double      e1_min = 0.1, e1_max = 1.0, e1_best;
56    double      e2_min = 0.1, e2_max = 1.0, e2_best;
57    double      delta_ks, delta_k1, delta_k2, delta_y,
58                delta_yp, delta_e1, delta_e2;
59    FILE *fopen(), *cdat1, *cdat2, *cdat3;
60    //k=0.17, YCH4_V=0.66, Vv=0.66, Ks=11.25, YV_S=0.7, YCO2_S=0.29, YCO2_V=0.19;
61
62    delta_ks = (ks_max - ks_min)/num_ks;
63    printf("delta_ks = %f\n",delta_ks);
64
65    delta_k1 = (k1_max - k1_min)/num_k1;
66    printf("delta_k1 = %f\n", delta_k1);
67
68    delta_k2 = (k2_max - k2_min)/num_k2;
69    printf("delta_k2 = %f\n", delta_k2);
70
71    delta_y = (y_max - y_min)/num_y;
72    printf("delta_y = %f\n",delta_y);
73
74    delta_yp = (yp_max - yp_min)/num_yp;
75    printf("delta_yp = %f\n",delta_yp);
76
77    delta_e1  = (e1_max - e1_min)/num_e1;
78    printf("delta_e1 = %f\n",delta_e1);
79
80    delta_e2  = (e2_max - e2_min)/num_e2;
81    printf("delta_e2 = %f\n",delta_e2);
```

```
82
83      time = 0.0;
84
85      //  printf("\n enter the upper limitation of t coordinate tn=");
86      //  scanf("%lf",&tn);
87      //  printf("\n enter the number of division of the interval n=");
88      //  scanf("%d",&n);
89      t0 = 0.0;
90      t16 = 16.0;
91      t30 = 30.0;
92      n = 3000;
93
94      h=(t30-t0)/n;  printf("\n h = %f\n",h);
95      i16 = (t16 - t0)/h;
96      printf("i16 = %d\n",i16);
97
98      error = 0.0;
99
100     for (je2=0; je2<=num_e2; je2++){
101       YCO2_V = yco2v_min + jyco2v*delta_yco2v;
102
103     for (je1=0; je1<=num_e1; je1++){
104       E1 = e1_min + je1*delta_e1;
105
106     for (jyp=0; jyp<=num_yp; jyp++){
107       Yp = yp_min + jyp*delta_yp;
108
109     for (jy=0; jy<=num_y; jy++){
110       Y = y_min + jy*delta_y;
111
112     for (jk2=0; jk2<=num_k2; jk2++){
113       k2 = k2_min + jk2*delta_k2;
114
115     //start loop for k1
116     for (jk1=0; jk1<=num_k1; jk1++){
117       k1 = k1_min + jk1*delta_k1;
118
119     //start loop for Ks
120     for(jks=0;jks<=numk;jks++){
121       //    printf("j = %d, error = %f\n",j,error);
122       Ks = ks_min + jks*delta_ks;
123       //    printf("Ks = %ff\n",Ks);
124
125     tTEMP = t0; STEMP=SINIT, XTEMP=XINIT, PTEMP=PINIT;
126
127        ff10 = F1(tTEMP,STEMP,XTEMP,PTEMP);
128        ff20 = F2(tTEMP,STEMP,XTEMP,PTEMP);
```

```
129         ff30 = F3(tTEMP,STEMP,XTEMP,PTEMP);
130         ff40 = F4(tTEMP,STEMP,XTEMP,PTEMP);
131
132
133       /* Runge Kutta */
134       for(i=1; i<=3; i++)
135         {
136         tTEMP = t0 + (i-1)*h;
137         k11 = h*F1(tTEMP,STEMP,XTEMP,PTEMP);
138         k21 = h*F2(tTEMP,STEMP,XTEMP,PTEMP);
139         k31 = h*F3(tTEMP,STEMP,XTEMP,PTEMP);
140         tTEMP2 = tTEMP + h/2.0;
141         STEMP2 = STEMP + k11/2.0;
142         XTEMP2 = XTEMP + k21/2.0;
143         PTEMP2 = PTEMP + k31/2.0;
144         k12 = h*F1(tTEMP2,STEMP2,XTEMP2,PTEMP2);
145         k22 = h*F2(tTEMP2,STEMP2,XTEMP2,PTEMP2);
146         k32 = h*F3(tTEMP2,STEMP2,XTEMP2,PTEMP2);
147         tTEMP3 = tTEMP + h/2.0;
148         STEMP3 = STEMP + k12/2.0;
149         XTEMP3 = XTEMP + k22/2.0;
150         PTEMP3 = PTEMP + k32/2.0;
151         k13 = h*F1(tTEMP3,STEMP3,XTEMP3,PTEMP3);
152         k23 = h*F2(tTEMP3,STEMP3,XTEMP3,PTEMP3);
153         k33 = h*F3(tTEMP3,STEMP3,XTEMP3,PTEMP3);
154         tTEMP4 = tTEMP + h;
155         STEMP4 = STEMP + k13;
156         XTEMP4 = XTEMP + k23;
157         PTEMP4 = PTEMP + k33;
158         k14 = h*F1(tTEMP4,STEMP4,XTEMP4,PTEMP4);
159         k24 = h*F2(tTEMP4,STEMP4,XTEMP4,PTEMP4);
160         k34 = h*F3(tTEMP4,STEMP4,XTEMP4,PTEMP4);
161         STEMP = STEMP     + (k11+2.0*k12+2.0*k13+k14)/6.0;
162         XTEMP = XTEMP     + (k21+2.0*k22+2.0*k23+k24)/6.0;
163         PTEMP = PTEMP     + (k31+2.0*k32+2.0*k33+k34)/6.0;
164
165
166         if(i == 1){
167          ff11 = F1(tTEMP,STEMP,XTEMP,PTEMP);
168          ff21 = F2(tTEMP,STEMP,XTEMP,PTEMP);
169          ff31 = F3(tTEMP,STEMP,XTEMP,PTEMP);
170       }
171
172       else if(i == 2){
173         ff12 = F1(tTEMP,STEMP,XTEMP,PTEMP);(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
174         ff22 = F2(tTEMP,STEMP,XTEMP,PTEMP);(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
175         ff32 = F3(tTEMP,STEMP,XTEMP,PTEMP);(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
```

```
176         ff42 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
177       }
178
179       else if(i == 3){
180         ff13 = F1(tTEMP,STEMP,XTEMP,PTEMP);
181         ff23 = F2(tTEMP,STEMP,XTEMP,PTEMP);
182         ff33 = F3(tTEMP,STEMP,XTEMP,PTEMP);
183       }
184
185        }//end loop for Runge Kutta
186
187       for (i=4; i<=n; i++){
188        tTEMP = t0 + i*h;
189          /* Predictor */
190         STEMP = STEMP + (55.0*ff13-59.0*ff12+37.0*ff11-9.0*ff10)/24.0;
191         XTEMP = XTEMP + (55.0*ff23-59.0*ff22+37.0*ff21-9.0*ff20)/24.0;
192         PTEMP = PTEMP + (55.0*ff33-59.0*ff32+37.0*ff31-9.0*ff30)/24.0;
193
194         /* Evaluate */
195         ff14 = F1(tTEMP,STEMP,XTEMP,PTEMP);
196         ff24 = F2(tTEMP,STEMP,XTEMP,PTEMP);
197         ff34 = F3(tTEMP,STEMP,XTEMP,PTEMP);
198
199         /* Corector */
200         STEMP = STEMP + (9.0*ff14+19.0*ff13-5.0*ff12+ff11)/24.0;
201         XTEMP = XTEMP + (9.0*ff24+19.0*ff23-5.0*ff22+ff21)/24.0;
202         PTEMP = PTEMP + (9.0*ff34+19.0*ff33-5.0*ff32+ff31)/24.0;
203
204         /* Evaluate */
205         ff14 = F1(tTEMP,STEMP,XTEMP,PTEMP);
206         ff24 = F2(tTEMP,STEMP,XTEMP,PTEMP);
207         ff34 = F3(tTEMP,STEMP,XTEMP,PTEMP);
208
209         ff10 = ff11;  ff20 = ff21;  ff30 = ff31;
210         ff11 = ff12;  ff21 = ff22;  ff31 = ff32;
211         ff12 = ff13;  ff22 = ff23;  ff32 = ff33;
212         ff13 = ff14;  ff23 = ff24;  ff33 = ff34;
213
214
215
216        if(i == i16-1){
217          errortemp = sqrt((PTEMP - ch4_day16)*(PTEMP - ch4_day16));
218                 if(jks == 0 && jk1 == 0 && jk2 == 0 && jy == 0 &&
219                    jyp == 0 && je1 == 0  && je2 == 0){
220                     error = errortemp;
221                     ks_best = Ks;
222                     k1_best = k1;
```

```
223                     k2_best = k2;
224                     y_best  = Y;
225                     yp_best = Yp;
226                     e1_best = E1;
227                     e2_best = E2;
228 
229                          }
230                 else if(errortemp < error){
231                     printf("Check\n");
232                                 error = errortemp;
233                                 ks_best = Ks;
234                                 k1_best = k1;
235                                 k2_best = k2;
236                                 y_best  = Y;
237                                 yp_best = Yp;
238                                 e1_best = E1;
239                                 e2_best = E2;
240 
241                 }
242 
243 
244       }
245 
246       }//end loop for predictor-corector
247 
248     }
249     // end loop for k
250 
251     }
252     // end loop for ych4v
253 
254     }
255 
256     }
257 
258     }
259 
260     }
261 
262     }
263 
264   printf("ks_best = %f, k1_best = %f, k2_best = %f, y_best = %f, yp_best = %f,
265 
266     Ks = ks_best;
267     k1 = k1_best;
268     k2 = k2_best;
269     Y  = y_best;
```

```
270     Yp = yp_best;
271     E1 = e1_best;
272     E2 = e2_best;
273     printf("Ks = %f, k1 = %f, k2 = %f, Y = %f, Yp = %f, E1 = %f, E2 = %f\n",
274
275     cdat1=fopen("./S.dat","w");
276     cdat2=fopen("./X.dat","w");
277     cdat3=fopen("./P.dat","w");
278
279     tTEMP = t0; STEMP=SINIT, XTEMP=XINIT, PTEMP=PINIT;
280     printf("t = %f, S = %f, X = %f, P = %f\n",tTEMP, STEMP, XTEMP, PTEMP);
281
282     fprintf(cdat1,"%f %f\n",tTEMP, STEMP);
283     fprintf(cdat2,"%f %f\n",tTEMP, XTEMP);
284     fprintf(cdat3,"%f %f\n",tTEMP, PTEMP);
285
286
287        ff10 = F1(tTEMP,STEMP,XTEMP,PTEMP);
288        ff20 = F2(tTEMP,STEMP,XTEMP,PTEMP);
289        ff30 = F3(tTEMP,STEMP,XTEMP,PTEMP);
290
291
292     for(i=1; i<=3; i++)
293       {
294       tTEMP = t0 + (i-1)*h;
295       k11 = h*F1(tTEMP,STEMP,XTEMP,PTEMP);
296       k21 = h*F2(tTEMP,STEMP,XTEMP,PTEMP);
297       k31 = h*F3(tTEMP,STEMP,XTEMP,PTEMP);
298       tTEMP2 = tTEMP + h/2.0;
299       STEMP2 = STEMP + k11/2.0;
300       XTEMP2 = XTEMP + k21/2.0;
301       PTEMP2 = PTEMP + k31/2.0;
302       k12 = h*F1(tTEMP2,STEMP2,XTEMP2,PTEMP2);
303       k22 = h*F2(tTEMP2,STEMP2,XTEMP2,PTEMP2);
304       k32 = h*F3(tTEMP2,STEMP2,XTEMP2,PTEMP2);
305       tTEMP3 = tTEMP + h/2.0;
306       STEMP3 = STEMP + k12/2.0;
307       XTEMP3 = XTEMP + k22/2.0;
308       PTEMP3 = PTEMP + k32/2.0;
309       k13 = h*F1(tTEMP3,STEMP3,XTEMP3,PTEMP3);
310       k23 = h*F2(tTEMP3,STEMP3,XTEMP3,PTEMP3);
311       k33 = h*F3(tTEMP3,STEMP3,XTEMP3,PTEMP3);
312       tTEMP4 = tTEMP + h;
313       STEMP4 = STEMP + k13;
314       XTEMP4 = XTEMP + k23;
315       PTEMP4 = PTEMP + k33;
316       k14 = h*F1(tTEMP4,STEMP4,XTEMP4,PTEMP4);
```

```
317       k24 = h*F2(tTEMP4,STEMP4,XTEMP4,PTEMP4);
318       k34 = h*F3(tTEMP4,STEMP4,XTEMP4,PTEMP4);
319       STEMP = STEMP     + (k11+2.0*k12+2.0*k13+k14)/6.0;
320       XTEMP = XTEMP     + (k21+2.0*k22+2.0*k23+k24)/6.0;
321       PTEMP = PTEMP + (k31+2.0*k32+2.0*k33+k34)/6.0;
322
323     fprintf(cdat1,"%f %f\n",tTEMP+h, STEMP);
324     fprintf(cdat2,"%f %f\n",tTEMP+h, XTEMP);
325     fprintf(cdat3,"%f %f\n",tTEMP+h, PTEMP);
326     if(i == i16-1) printf("t = %f\n",tTEMP+h);
327
328
329     if(i == 1){
330       ff11 = F1(tTEMP,STEMP,XTEMP,PTEMP);
331       ff21 = F2(tTEMP,STEMP,XTEMP,PTEMP);
332       ff31 = F3(tTEMP,STEMP,XTEMP,PTEMP);
333     }
334
335     else if(i == 2){
336       ff12 = F1(tTEMP,STEMP,XTEMP,PTEMP);
337       ff22 = F2(tTEMP,STEMP,XTEMP,PTEMP);
338       ff32 = F3(tTEMP,STEMP,XTEMP,PTEMP);
339     }
340
341     else if(i == 3){
342       ff13 = F1(tTEMP,STEMP,XTEMP,PTEMP);
343       ff23 = F2(tTEMP,STEMP,XTEMP,PTEMP);
344       ff33 = F3(tTEMP,STEMP,XTEMP,PTEMP);
345     }
346
347      }
348
349
350     for (i=4; i<=n; i++){
351        /* Predictor */
352       tTEMP = t0 + i*h;
353       STEMP = STEMP + (55.0*ff13-59.0*ff12+37.0*ff11-9.0*ff10)/24.0;
354       XTEMP = XTEMP + (55.0*ff23-59.0*ff22+37.0*ff21-9.0*ff20)/24.0;
355       PTEMP = PTEMP + (55.0*ff33-59.0*ff32+37.0*ff31-9.0*ff30)/24.0;
356
357       /* Evaluate */
358       ff14 = F1(tTEMP,STEMP,XTEMP,PTEMP);
359       ff24 = F2(tTEMP,STEMP,XTEMP,PTEMP);
360       ff34 = F3(tTEMP,STEMP,XTEMP,PTEMP);
361
362       /* Corector */
363       STEMP = STEMP + (9.0*ff14+19.0*ff13-5.0*ff12+ff11)/24.0;
```

```
364       XTEMP = XTEMP + (9.0*ff24+19.0*ff23-5.0*ff22+ff21)/24.0;
365       PTEMP = PTEMP + (9.0*ff34+19.0*ff33-5.0*ff32+ff31)/24.0;
366
367       /* Evaluate */
368       ff14 = F1(tTEMP,STEMP,XTEMP,PTEMP);
369       ff24 = F2(tTEMP,STEMP,XTEMP,PTEMP);
370       ff34 = F3(tTEMP,STEMP,XTEMP,PTEMP);
371
372     fprintf(cdat1,"%f %f\n",tTEMP+h, STEMP);
373     fprintf(cdat2,"%f %f\n",tTEMP+h, XTEMP);
374     fprintf(cdat3,"%f %f\n",tTEMP+h, PTEMP);
375
376
377       ff10 = ff11;  ff20 = ff21;  ff30 = ff31;
378       ff11 = ff12;  ff21 = ff22;  ff31 = ff32;
379       ff12 = ff13;  ff22 = ff23;  ff32 = ff33;
380       ff13 = ff14;  ff23 = ff24;  ff33 = ff34;
381
382     }
383
384      fclose(cdat1);
385      fclose(cdat2);
386      fclose(cdat3);
387
388   return (0);
389 }
390
391
392 double F1(double t, double S, double X, double P)
393 {
394   return ((1/Y)*(X_0*S/Ks+S)*(k1*exp(-E1/RT)-k2*exp(-E2/RT)));
395 }
396
397 double F2(double t, double S, double X, double P)
398 {
399   return ((X_0*S/Ks+S)*(k1*exp(-E1/RT)-k2*exp(-E2/RT)));
400 }
401
402 double F3(double t, double S, double X, double P)
403 {
404   return ((-Yp*X_0*S*10^3/Ks+S)*(k1*exp(-E1/RT)-k2*exp(-E2/RT)));
405 }
406
407
408
```

Appendix B

P. Sosnowski Model

```
1   #include <stdio.h>
2
3   #include <math.h>
4
5   double F1(double, double, double, double, double);
6   double F2(double, double, double, double, double);
7   double F3(double, double, double, double, double);
8   double F4(double, double, double, double, double);
9
10    int           n,i;
11    double        time[100000], S[100000], V[100000], CH4[100000], CO2[100000];
12    double        X_0;
13    double        h,xn,t16,t30,t0,y0_,x0=0.0,tinit,ch4,co2;
14    double        tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP;
15    double        ff10,ff11,ff12,ff13,ff14;
16    double        ff20,ff21,ff22,ff23,ff24;
17    double        ff30,ff31,ff32,ff33,ff34;
18    double        ff40,ff41,ff42,ff43,ff44;
19
20    double        dSdt,dVdt,dCH4dt,dCO2dt;
21    double        k,Ks,YV_S,YCH4_V,YCO2_S,YCO2_V,Vv;
22    double        xi,k1,k2,k3,k4,eta0,eta1;
23    double        k11,k12,k13,k14;
24    double        k21,k22,k23,k24;
25    double        k31,k32,k33,k34;
26    double        k41,k42,k43,k44;
27    double        eta10,eta11;
28    double        eta20,eta21;
29    double        eta30,eta31;
30    double        eta40,eta41;
31    double        X_0=4.0;
32    double        SINIT=4.8;
33    double        VINIT=0.48;
34    double        CH4INIT=0.0;
```

```
35    double        CO2INIT=0.0;
36
37  int main(void)
38  {
39    int j, j1, j2, j3, j4, j5, j6, i16,
40      jk,jych4v,jvv,jks,jyvs,jyco2s,jyco2v,numdat,
41      numk = 3, num_ych4v = 3, num_vv = 3, num_yvs = 3, num_yco2v = 3,
42      i3,i9,i15,i21,i27;
43
44    double      time;
45    double      tTEMP, STEMP, VTEMP, CH4TEMP, CO2TEMP;
46    double      tTEMP2, STEMP2, VTEMP2, CH4TEMP2, CO2TEMP2;
47    double      tTEMP3, STEMP3, VTEMP3, CH4TEMP3, CO2TEMP3;
48    double      tTEMP4, STEMP4, VTEMP4, CH4TEMP4, CO2TEMP4;
49    double      ch4_day3, ch4_day9, ch4_day15, ch4_day21, ch4_day27;
50    double      co2_day3, co2_day9, co2_day15, co2_day21, co2_day27;
51    double      error, errortemp;
52    double      SMIN, VMIN, CH4MIN, CO2MIN;
53
54    /*
55    double      kmin      = 0.01,    kmax  = 0.7, k_best;
56    double      ks_min    = 1.0,    ks_max = 20.0, ks_best;
57    double      yvs_min   = 1.0,   yvs_max = 50.0, yvs_best;
58    double      vv_min    = 0.1,    vv_max = 0.2, vv_best;
59    double      ych4v_min = 0.1, ych4v_max = 1.0, ych4v_best;
60    double      yco2s_min = 0.1, yco2s_max = 1.0, yco2s_best;
61    double      yco2v_min = 0.1, yco2v_max = 1.0, yco2v_best; */
62
63    double      kmin      = 0.00001, kmax  = 0.001, k_best;
64    double      yvs_min   = 1.0,   yvs_max = 50.0, yvs_best;
65    double      vv_min    = 0.0001, vv_max = 0.01, vv_best;
66    double      ych4v_min = 0.1, ych4v_max = 0.7, ych4v_best;
67    double      yco2v_min = 0.1, yco2v_max = 0.5, yco2v_best;
68
69
70    //double      kmin      = 0.01,    kmax  = 1.0, k_best;
71    //double      ks_min    = 0.8,    ks_max = 0.9, ks_best;
72    //double      vv_min    = 0.8,    vv_max = 0.9, vv_best;
73    //double      yco2s_min = 1.0, yco2s_max = 2.0, yco2s_best;
74
75    /* Old parameters range */
76    /*
77    double      kmin      = 0.001, kmax    = 1.0, k_best;
78    double      ks_min    = 0.1, ks_max    = 60.0, ks_best;
79    double      yvs_min   = 1.0, yvs_max   = 45.0, yvs_best;
80    double      vv_min    = 0.1, vv_max    = 1.0, vv_best;
81    double      ych4v_min = 1.0,ych4v_max  = 20.0, ych4v_best;
```

```
82      double      yco2s_min = 0.1, yco2s_max = 1.0, yco2s_best;
83      double      yco2v_min = 0.1, yco2v_max = 1.0, yco2v_best;
84      */
85
86      double      deltak, delta_ych4v, delta_vv, delta_ks, delta_yvs,
87                  delta_yco2s, delta_yco2v;
88
89      FILE *fopen(), *cdat1, *cdat2, *cdat3, *cdat4;
90      //k=0.17,Ks=11.25, YV_S=0.7, Vv=0.66, YCH4_V=0.66, YCO2_S=0.29, YCO2_V=0.19;
91      Ks=11.25, YCO2_S=0.29;
92
93      tinit = 0.0;
94      t0 = 0.0;
95      t16 = 16.0;
96      t30 = 30.0;
97      //n = 3000;
98      //n = 6000;
99      n = 12000;
100
101     h=(t30-t0)/n;  printf("\n h = %f\n",h);
102     i16 = (t16 - t0)/h;
103     i3 = 3.0/h;
104     i9 = 9.0/h;
105     i15 = 15.0/h;
106     i21 = 21.0/h;
107     i27 = 27.0/h;
108     printf("i16 = %d\n",i16);
109
110     ch4_day3  = 2.0;
111     ch4_day9  = 4.4;
112     ch4_day15 = 66.5;
113     ch4_day21 = 76.5;
114     ch4_day27 = 76.5;
115
116     co2_day3  = 12.83;
117     co2_day9  = 17.83;
118     co2_day15 = 45.9;
119     co2_day21 = 55.9;
120     co2_day27 = 55.9;
121
122     time = 0.0;
123
124     error = 0.0;
125
126
127     deltak = (kmax - kmin)/numk;
128     printf("deltak = %f\n",deltak);
```

```
129
130     delta_yvs = (yvs_max - yvs_min)/num_yvs;
131     printf("delta_yvs = %f\n",delta_yvs);
132
133     delta_vv = (vv_max - vv_min)/num_vv;
134     printf("delta_vv = %f\n", delta_vv);
135
136     delta_ych4v = (ych4v_max - ych4v_min)/num_ych4v;
137     printf("delta_ych4v = %f\n", delta_ych4v);
138
139     delta_yco2v  = (yco2v_max - yco2v_min)/num_yco2v;
140     printf("delta_yco2v = %f\n",delta_yco2v);
141
142     for (jyco2v=0; jyco2v<=num_yco2v; jyco2v++){
143       YCO2_V = yco2v_min + jyco2v*delta_yco2v;
144
145     for (jych4v=0; jych4v<=num_ych4v; jych4v++){
146       YCH4_V = ych4v_min + jych4v*delta_ych4v;
147
148     for (jvv=0; jvv<=num_vv; jvv++){
149       Vv = vv_min + jvv*delta_vv;
150
151     for (jyvs=0; jyvs<=num_yvs; jyvs++){
152       YV_S = yvs_min + jyvs*delta_yvs;
153
154     //start loop for k
155     for(jk=0;jk<=numk;jk++){
156       k = kmin + jk*deltak;
157
158       tTEMP = t0; STEMP=SINIT, VTEMP=VINIT, CH4TEMP=CH4INIT, CO2TEMP=CO2INIT;
159
160        ff10 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
161        ff20 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
162        ff30 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
163        ff40 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
164
165
166     /* Runge Kutta */
167     for(i=1; i<=3; i++)
168       {
169
170         errortemp = 0.0;
171       tTEMP = t0 + (i-1)*h;
172       k11 = h*F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
173       k21 = h*F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
174       k31 = h*F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
175       k41 = h*F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
```

```
176       tTEMP2 = tTEMP + h/2.0;
177       STEMP2 = STEMP + k11/2.0;
178       VTEMP2 = VTEMP + k21/2.0;
179       CH4TEMP2 = CH4TEMP + k31/2.0;
180       CO2TEMP2 = CO2TEMP + k41/2.0;
181
182       k12 = h*F1(tTEMP2,STEMP2,VTEMP2,CH4TEMP2,CO2TEMP2);
183       k22 = h*F2(tTEMP2,STEMP2,VTEMP2,CH4TEMP2,CO2TEMP2);
184       k32 = h*F3(tTEMP2,STEMP2,VTEMP2,CH4TEMP2,CO2TEMP2);
185       k42 = h*F4(tTEMP2,STEMP2,VTEMP2,CH4TEMP2,CO2TEMP2);
186       tTEMP3 = tTEMP + h/2.0;
187       STEMP3 = STEMP + k12/2.0;
188       VTEMP3 = VTEMP + k22/2.0;
189       CH4TEMP3 = CH4TEMP + k32/2.0;
190       CO2TEMP3 = CO2TEMP + k42/2.0;
191
192       k13 = h*F1(tTEMP3,STEMP3,VTEMP3,CH4TEMP3,CO2TEMP3);
193       k23 = h*F2(tTEMP3,STEMP3,VTEMP3,CH4TEMP3,CO2TEMP3);
194       k33 = h*F3(tTEMP3,STEMP3,VTEMP3,CH4TEMP3,CO2TEMP3);
195       k43 = h*F4(tTEMP3,STEMP3,VTEMP3,CH4TEMP3,CO2TEMP3);
196       tTEMP4 = tTEMP + h;
197       STEMP4 = STEMP + k13;
198       VTEMP4 = VTEMP + k23;
199       CH4TEMP4 = CH4TEMP + k33;
200       CO2TEMP4 = CO2TEMP + k43;
201
202       k14 = h*F1(tTEMP4,STEMP4,VTEMP4,CH4TEMP4,CO2TEMP4);
203       k24 = h*F2(tTEMP4,STEMP4,VTEMP4,CH4TEMP4,CO2TEMP4);
204       k34 = h*F3(tTEMP4,STEMP4,VTEMP4,CH4TEMP4,CO2TEMP4);
205       k44 = h*F4(tTEMP4,STEMP4,VTEMP4,CH4TEMP4,CO2TEMP4);
206       STEMP = STEMP     + (k11+2.0*k12+2.0*k13+k14)/6.0;
207       VTEMP = VTEMP     + (k21+2.0*k22+2.0*k23+k24)/6.0;
208       CH4TEMP = CH4TEMP + (k31+2.0*k32+2.0*k33+k34)/6.0;
209       CO2TEMP = CO2TEMP   + (k41+2.0*k42+2.0*k43+k44)/6.0;
210
211
212
213       if(i == 1){
214         ff11 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
215         ff21 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
216         ff31 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
217         ff41 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
218       }
219
220       else if(i == 2){
221         ff12 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
222         ff22 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
```

```
223         ff32 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
224         ff42 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
225       }
226
227       else if(i == 3){
228         ff13 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
229         ff23 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
230         ff33 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
231         ff43 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
232       }
233
234        }//end loop for Runge Kutta
235
236       for (i=4; i<=n; i++){
237        tTEMP = t0 + i*h;
238          /* Predictor */
239         STEMP   = STEMP   + (55.0*ff13-59.0*ff12+37.0*ff11-9.0*ff10)/24.0;
240         VTEMP   = VTEMP   + (55.0*ff23-59.0*ff22+37.0*ff21-9.0*ff20)/24.0;
241         CH4TEMP = CH4TEMP + (55.0*ff33-59.0*ff32+37.0*ff31-9.0*ff30)/24.0;
242         CO2TEMP = CO2TEMP + (55.0*ff43-59.0*ff42+37.0*ff41-9.0*ff40)/24.0;
243
244         /* Evaluate */
245         ff14 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
246         ff24 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
247         ff34 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
248         ff44 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
249
250         /* Corector */
251         STEMP   = STEMP   + (9.0*ff14+19.0*ff13-5.0*ff12+ff11)/24.0;
252         VTEMP   = VTEMP   + (9.0*ff24+19.0*ff23-5.0*ff22+ff21)/24.0;
253         CH4TEMP = CH4TEMP + (9.0*ff34+19.0*ff33-5.0*ff32+ff31)/24.0;
254         CO2TEMP = CO2TEMP + (9.0*ff44+19.0*ff43-5.0*ff42+ff41)/24.0;
255
256         /* Evaluate */
257         ff14 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
258         ff24 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
259         ff34 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
260         ff44 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
261
262         ff10 = ff11;  ff20 = ff21;  ff30 = ff31;  ff40 = ff41;
263         ff11 = ff12;  ff21 = ff22;  ff31 = ff32;  ff41 = ff42;
264         ff12 = ff13;  ff22 = ff23;  ff32 = ff33;  ff42 = ff43;
265         ff13 = ff14;  ff23 = ff24;  ff33 = ff34;  ff43 = ff44;
266
267       if(i == i3) errortemp = errortemp +(CH4TEMP - ch4_day3)
268                               *(CH4TEMP - ch4_day3)+(CO2TEMP - co2_day3)
269                               *(CO2TEMP - co2_day3);
```

```
270      if(i == i9) errortemp = errortemp +(CH4TEMP - ch4_day9)
271                              *(CH4TEMP - ch4_day9)+(CO2TEMP - co2_day9)
272                              *(CO2TEMP - co2_day9);
273      if(i == i15) errortemp = errortemp +(CH4TEMP - ch4_day15)
274                              *(CH4TEMP - ch4_day15)+(CO2TEMP - co2_day15)
275                              *(CO2TEMP - co2_day15);
276      if(i == i21) errortemp = errortemp +(CH4TEMP - ch4_day21)
277                              *(CH4TEMP - ch4_day21)+(CO2TEMP - co2_day21)
278                              *(CO2TEMP - co2_day21);
279      if(i == i27) errortemp = errortemp +(CH4TEMP - ch4_day27)
280                              *(CH4TEMP - ch4_day27)+(CO2TEMP - co2_day27)
281                              *(CO2TEMP - co2_day27);
282
283      }//end loop for predictor-corector
284
285
286         if(jk == 0 && jyvs == 0 && jvv == 0 && jych4v == 0 && jyco2v == 0){
287             error = errortemp;
288             k_best = k;
289             yvs_best = YV_S;
290             vv_best = Vv;
291             ych4v_best = YCH4_V;
292             yco2v_best = YCO2_V;
293                                  }
294                else if(errortemp < error){
295                     error = errortemp;
296                     k_best = k;
297                     yvs_best = YV_S;
298                     vv_best = Vv;
299                     ych4v_best = YCH4_V;
300                     yco2v_best = YCO2_V;
301                        }
302
303     }//end loop for k
304
305     }// end loop for ych4v
306
307     }
308
309     }
310
311     }
312
313     printf("k_best = %f, yvs_best = %f, vv_best = %f, ych4v_best = %f,
314
315     k    = k_best;
316     YV_S = yvs_best;
```

```
317     Vv   = vv_best;
318     YCH4_V = ych4v_best;
319     YCO2_V = yco2v_best;
320     printf("k = %f, YV_S = %f, Vv = %f, YCH4_V = %f, YCO2_V = %f\n",
321             k,YV_S,Vv,YCH4_V,YCO2_V);
322
323     /* n 6000 3it */
324     /* cdat1=fopen("./S.dat","w");
325     cdat2=fopen("./V.dat","w");
326     cdat3=fopen("./CH4.dat","w");
327     cdat4=fopen("./CO2.dat","w");*/
328
329     /* n 6000 10it  */
330     /*  cdat1=fopen("./S10.dat","w");
331     cdat2=fopen("./V10.dat","w");
332     cdat3=fopen("./CH410.dat","w");
333     cdat4=fopen("./CO210.dat","w");*/
334
335     /* n 12000 3it  */
336     /*  cdat1=fopen("./nS.dat","w");
337     cdat2=fopen("./nV.dat","w");
338     cdat3=fopen("./nCH4.dat","w");
339     cdat4=fopen("./nCO2.dat","w");*/
340
341     /* n 12000 10it  */
342     cdat1=fopen("./nS10.dat","w");
343     cdat2=fopen("./nV10.dat","w");
344     cdat3=fopen("./nCH410.dat","w");
345     cdat4=fopen("./nCO210.dat","w");
346
347     tTEMP = t0; STEMP=SINIT, VTEMP=VINIT, CH4TEMP=CH4INIT, CO2TEMP=CO2INIT;
348     printf("t = %f, S = %f, V = %f, CH4 = %f, CO2 = %f\n",
349             tTEMP, STEMP, VTEMP, CH4TEMP, CO2TEMP);
350
351     fprintf(cdat1,"%f %f\n",tTEMP, STEMP);
352     fprintf(cdat2,"%f %f\n",tTEMP, VTEMP);
353     fprintf(cdat3,"%f %f\n",tTEMP, CH4TEMP);
354     fprintf(cdat4,"%f %f\n",tTEMP, CO2TEMP);
355
356        ff10 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
357        ff20 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
358        ff30 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
359        ff40 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
360
361     for(i=1; i<=3; i++)
362       {
363       tTEMP = t0 + (i-1)*h;
```

```
364       k11 = h*F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
365       k21 = h*F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
366       k31 = h*F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
367       k41 = h*F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
368       tTEMP2 = tTEMP + h/2.0;
369       STEMP2 = STEMP + k11/2.0;
370       VTEMP2 = VTEMP + k21/2.0;
371       CH4TEMP2 = CH4TEMP + k31/2.0;
372       CO2TEMP2 = CO2TEMP + k41/2.0;
373       k12 = h*F1(tTEMP2,STEMP2,VTEMP2,CH4TEMP2,CO2TEMP2);
374       k22 = h*F2(tTEMP2,STEMP2,VTEMP2,CH4TEMP2,CO2TEMP2);
375       k32 = h*F3(tTEMP2,STEMP2,VTEMP2,CH4TEMP2,CO2TEMP2);
376       k42 = h*F4(tTEMP2,STEMP2,VTEMP2,CH4TEMP2,CO2TEMP2);
377       tTEMP3 = tTEMP + h/2.0;
378       STEMP3 = STEMP + k12/2.0;
379       VTEMP3 = VTEMP + k22/2.0;
380       CH4TEMP3 = CH4TEMP + k32/2.0;
381       CO2TEMP3 = CO2TEMP + k42/2.0;
382       k13 = h*F1(tTEMP3,STEMP3,VTEMP3,CH4TEMP3,CO2TEMP3);
383       k23 = h*F2(tTEMP3,STEMP3,VTEMP3,CH4TEMP3,CO2TEMP3);
384       k33 = h*F3(tTEMP3,STEMP3,VTEMP3,CH4TEMP3,CO2TEMP3);
385       k43 = h*F4(tTEMP3,STEMP3,VTEMP3,CH4TEMP3,CO2TEMP3);
386       tTEMP4 = tTEMP + h;
387       STEMP4 = STEMP + k13;
388       VTEMP4 = VTEMP + k23;
389       CH4TEMP4 = CH4TEMP + k33;
390       CO2TEMP4 = CO2TEMP + k43;
391       k14 = h*F1(tTEMP4,STEMP4,VTEMP4,CH4TEMP4,CO2TEMP4);
392       k24 = h*F2(tTEMP4,STEMP4,VTEMP4,CH4TEMP4,CO2TEMP4);
393       k34 = h*F3(tTEMP4,STEMP4,VTEMP4,CH4TEMP4,CO2TEMP4);
394       k44 = h*F4(tTEMP4,STEMP4,VTEMP4,CH4TEMP4,CO2TEMP4);
395       STEMP = STEMP     + (k11+2.0*k12+2.0*k13+k14)/6.0;
396       VTEMP = VTEMP     + (k21+2.0*k22+2.0*k23+k24)/6.0;
397       CH4TEMP = CH4TEMP + (k31+2.0*k32+2.0*k33+k34)/6.0;
398       CO2TEMP = CO2TEMP   + (k41+2.0*k42+2.0*k43+k44)/6.0;
399
400     fprintf(cdat1,"%f %f\n",tTEMP+h, STEMP);
401     fprintf(cdat2,"%f %f\n",tTEMP+h, VTEMP);
402     fprintf(cdat3,"%f %f\n",tTEMP+h, CH4TEMP);
403     fprintf(cdat4,"%f %f\n",tTEMP+h, CO2TEMP);
404     if(i == i16-1) printf("t = %f\n",tTEMP+h);
405
406     if(i == 1){
407       ff11 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
408       ff21 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
409       ff31 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
410       ff41 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
```

```
411       }
412
413       else if(i == 2){
414         ff12 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
415         ff22 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
416         ff32 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
417         ff42 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
418       }
419
420       else if(i == 3){
421         ff13 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
422         ff23 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
423         ff33 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
424         ff43 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
425       }
426
427        }
428
429       for (i=4; i<=n; i++){
430          /* Predictor */
431        tTEMP = t0 + i*h;
432         STEMP = STEMP + (55.0*ff13-59.0*ff12+37.0*ff11-9.0*ff10)/24.0;
433         VTEMP = VTEMP + (55.0*ff23-59.0*ff22+37.0*ff21-9.0*ff20)/24.0;
434         CH4TEMP = CH4TEMP + (55.0*ff33-59.0*ff32+37.0*ff31-9.0*ff30)/24.0;
435         CO2TEMP = CO2TEMP + (55.0*ff43-59.0*ff42+37.0*ff41-9.0*ff40)/24.0;
436
437         /* Evaluate */
438         ff14 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
439         ff24 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
440         ff34 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
441         ff44 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
442
443         /* Corector */
444         STEMP = STEMP + (9.0*ff14+19.0*ff13-5.0*ff12+ff11)/24.0;
445         VTEMP = VTEMP + (9.0*ff24+19.0*ff23-5.0*ff22+ff21)/24.0;
446         CH4TEMP = CH4TEMP + (9.0*ff34+19.0*ff33-5.0*ff32+ff31)/24.0;
447         CO2TEMP = CO2TEMP + (9.0*ff44+19.0*ff43-5.0*ff42+ff41)/24.0;
448
449         /* Evaluate */
450         ff14 = F1(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
451         ff24 = F2(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
452         ff34 = F3(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
453         ff44 = F4(tTEMP,STEMP,VTEMP,CH4TEMP,CO2TEMP);
454
455
456       fprintf(cdat1,"%f %f\n",tTEMP+h, STEMP);
457       fprintf(cdat2,"%f %f\n",tTEMP+h, VTEMP);
```

```
458      fprintf(cdat3,"%f %f\n",tTEMP+h, CH4TEMP);
459      fprintf(cdat4,"%f %f\n",tTEMP+h, CO2TEMP);
460
461
462        ff10 = ff11;  ff20 = ff21;  ff30 = ff31;  ff40 = ff41;
463        ff11 = ff12;  ff21 = ff22;  ff31 = ff32;  ff41 = ff42;
464        ff12 = ff13;  ff22 = ff23;  ff32 = ff33;  ff42 = ff43;
465        ff13 = ff14;  ff23 = ff24;  ff33 = ff34;  ff43 = ff44;
466
467      }
468
469       fclose(cdat1);
470       fclose(cdat2);
471       fclose(cdat3);
472       fclose(cdat4);
473
474    return (0);
475  }
476
477  double F1(double t, double S, double V, double CH4, double CO2)
478  {
479    return (-k*S);
480  }
481
482  double F2(double t, double S, double V, double CH4, double CO2)
483  {
484    return (YV_S*k*S-(Vv*V*X_0)/(Ks+V));
485  }
486
487  double F3(double t, double S, double V, double CH4, double CO2)
488  {
489    return (YCH4_V*Vv*V*X_0/(Ks+V));
490  }
491
492  double F4(double t, double S, double V, double CH4, double CO2)
493  {
494    return (YCO2_S*k*S+(YCO2_V*Vv*V*X_0)/(Ks+V));
495  }
496
497
```

Appendix C

Grau Model

```
1  %bio.m
2  function dx = bio(t, x)
3    global alfa  beta gamma delta;
4
5    %x(1)  = substrate;
6    %x(2)  = bacteria;
7    %x(3)  = methane;
8    %alfa  = Ys;
9    %gamma = Mu_max;
10   %beta  = Yp1;
11   %delta = Yp2;
12
13   dx = [0; 0; 0];
14   %dx = [0; 0];
15
16
17   %Modified Grau Model
18   dx(1) = -(1.0/alfa) * gamma * (x(1)/12.0) * x(2);
19   %dx(2) = gamma * (x(1)/12.0) * x(2);
20   dx(2) = gamma * (x(1)/12.0) * x(2) - delta * x(2);
21   dx(3) = beta * gamma * (x(1)/12.0) * x(2);
22   %dx(3) = (beta * gamma * (x(1)/12.0) * x(2)) + delta * x(2) ;
23
```

```
1  %fermentation
2
3  function y = fermentation(a, x)
4    global alfa beta gamma delta;
5
6    alfa  = a(1);
7    beta  = a(2);
8    gamma = a(3);
9    delta = a(4);
10
11    [t, y] = ode15s('bio', [0 x(end)], [12, 0.107, 0]);
```

```
12      %[t, y] = ode15s('bio', [0 x(end)], [12, 0.107]);
13      %[t, y] = ode15s('bio', [0 x(end)], [50000, 1, 0]);
14
15
16    y = interp1(t, y(:,3), x);
```

```
1   % Declare as global so the values can be passed into the SIR function
2   global alfa beta gamma delta;
3
4   % Set beta and delta to the values found by nlinfit
5   alfa  = values(1);
6   beta  = values(2);
7   gamma = values(3);
8   delta = values(4);
9
10  % Evaluate the ODE with the new values for alfa beta gamma and delta
11  [t, y] = ode15s('bio', [0 30], [12, 0.107, 0]);
12  %[t, y] = ode15s('bio', [0 30], [12, 0.107]);
13  %[t, y] = ode15s('bio', [0 14], [50000, 1, 0]);
14
15  % Display the results
16  hold on
17  grid on
18  %plot(t, y(:,2), '-b', day, sick*100, 'rs')
19  plot(t, y(:,3), '-b', day, methane, 'rs')
20  xlabel day
21  ylabel CH_4(g/L)
22  title('Methane-Experimental Vs Numerical')
23  legend('Numerical Results','Experiments')
24  hold off
```

```
1   %[t,x] = ode15s('biog', [0 30], [12 0.107 0], options);
2   %[t,x] = ode15s('bio', [0 30], [12 0.107], options);
3   [t,x] = ode15s('bio', [0 30], [12 0.107 0], options);
4   %[t,x] = ode15s('bio', [0 14], [50000 1 0], options);
5
6   plot(t,x);
7   hold on;
8   grid on;
9   legend('S', 'X', 'P');
10  %legend('S', 'X');
11  %legend('S', 'I', 'R');
```

```
1   function [beta,r,J,Sigma,mse,errorModelInfo,robustw] = nlinfit(X,y,model,beta,
2                                                                  varargin)
3   %NLINFIT Nonlinear least-squares regression.
4   %   BETA = NLINFIT(X,Y,MODELFUN,BETA0) estimates the coefficients of a
5   %   nonlinear regression function, using least squares estimation.  Y is a
```

```
6   %   vector of response (dependent variable) values.  Typically, X is a
7   %   design matrix of predictor (independent variable) values, with one row
8   %   for each value in Y and one column for each coefficient.  However, X
9   %   may be any array that MODELFUN is prepared to accept.  MODELFUN is a
10  %   function, specified using @, that accepts two arguments, a coefficient
11  %   vector and the array X, and returns a vector of fitted Y values.  BETA0
12  %   is a vector containing initial values for the coefficients.
13  %
14  %   [BETA,R,J,COVB,MSE] = NLINFIT(X,Y,MODELFUN,BETA0) returns the fitted
15  %   coefficients BETA, the residuals R, the Jacobian J of MODELFUN, the
16  %   estimated covariance matrix COVB for the fitted coefficients, and an
17  %   estimate MSE of the variance of the error term.  You can use these
18  %   outputs with NLPREDCI to produce confidence intervals for predictions,
19  %   and with NLPARCI to produce confidence intervals for the estimated
20  %   coefficients.  If you use a robust option (see below), you must use
21  %   COVB and may need MSE as input to NLPREDCI or NLPARCI to insure that
22  %   the confidence intervals take the robust fit properly into account.
23  %
24  %   [...] = NLINFIT(X,Y,MODELFUN,BETA0,OPTIONS) specifies control parameters
25  %   for the algorithm used in NLINFIT.  OPTIONS is a structure that can be
26  %   created by a call to STATSET.  Applicable STATSET parameters are:
27  %
28  %   'MaxIter'     - Maximum number of iterations allowed.  Defaults to 100.
29  %   'TolFun'      - Termination tolerance on the residual sum of squares.
30  %                   Defaults to 1e-8.
31  %   'TolX'        - Termination tolerance on the estimated coefficients
32  %                   BETA.  Defaults to 1e-8.
33  %   'Display'     - Level of display output during estimation.  Choices
34  %                       are 'off' (the default), 'iter', or 'final'.
35  %   'DerivStep'   - Relative difference used in finite difference gradient
36  %                   calculation.  May be a scalar, or the same size as
37  %                   the parameter vector BETA.  Defaults to EPS^(1/3).
38  %   'FunValCheck' - Check for invalid values, such as NaN or Inf, from
39  %                   the objective function.  'off' or 'on' (default).
40  %   'RobustWgtFun'- A weight function for robust fitting. Valid functions
41  %                   are 'bisquare', 'andrews', 'cauchy', 'fair', 'huber',
42  %                   'logistic', 'talwar', or 'welsch'. Default is '' (no
43  %                   robust fitting). Can also be a function handle that
44  %                   accepts a normalized residual as input and returns
45  %                   the robust weights as output.
46  %   'Robust'      - Robust will be removed in a future release. Use
47  %                   RobustWgtFun to invoke the robust fitting option.
48  %   'WgtFun'      - WgtFun will be removed in the future release. Use
49  %                   RobustWgtFun instead.
50  %   'Tune'        - The tuning constant used in robust fitting to normalize the
51  %                   residuals before applying the weight function.  A positive
52  %                   scalar.  The default value depends upon the weight function.
```

```
53  %                 This parameter is required if the weight function is
54  %                 specified as a function handle.
55  %
56  % [BETA,R,J,COVB,MSE] = NLINFIT(X,Y,MODELFUN,BETA0,OPTIONS,...,'Weights',W)
57  % nlinfit can accept an optional parameter name/value pair that specifies
58  % the observation weights:
59  %
60  % 'Weights'         A vector of real positive weights the same size as Y,
61  %                   each element of which specifies an observation weight.
62  %                   Reducing the weight of an observation reduces the
63  %                   influence of that observation on the fitted model.
64  %                   This can also be specified as a function handle that
65  %                   accepts a vector of predicted response values and
66  %                   returns a vector of real positive weights as output.
67  %                   Default is no weights.
68  %
69  % OPTIONS.RobustWgtFun must be [] when using observation weights. R and J
70  % are weighted residuals and a weighted model function Jacobian respectively.
71  % When J has full column rank, outputs MSE and R are related by:
72  %            MSE = (R'*R)/(n-p)
73  % where (n-p) = observations-parameters and COVB is related to J and MSE
74  % by:
75  %                    COVB = inv(J'*J) * MSE.
76  %
77  % [BETA,R,J,COVB,MSE,ERRORMODELINFO] = NLINFIT(X,Y,MODELFUN,BETA0,OPTIONS,...
78  %                                    'ErrorModel',val1,'ErrorParameters',val2)
79  % nlinfit can also accept parameter name/value pairs that specify the error
80  % model and an initial estimate of error parameters to be used by nlinfit.
81  % These parameter name/value pairs are described below.
82  %
83  % 'ErrorModel'      A string specifying the form of the error term.
84  %                   Default is 'constant'. Each model defines the error
85  %                   using a standard zero mean and unit variance variable e
86  %                   with independent components, model function value f,
87  %                   and one or two parameters a and b. The Choices for
88  %                   val1 are:
89  %
90  %                   'constant' (default)         y = f + a*e
91  %                   'proportional'               y = f + b*f*e
92  %                   'combined'                   y = f + (a+b*abs(f))*e
93  %
94  %                   If not specified, a 'constant' error model will be
95  %                   used. The only allowed 'ErrorModel' when using 'Weights'
96  %                   is 'constant'. OPTIONS.RobustWgtFun must be [] when
97  %                   using error models other than 'constant'.
98  %
99  % 'ErrorParameters'  A numeric array containing initial estimates of the
```

```
100  %                    error model parameters of the chosen 'ErrorModel'.
101  %                    Specify val2 as:
102  %
103  %                    a       (default 1)       for 'constant'
104  %                    b       (default 1)       for 'proportional'
105  %                    [a b]   (default [1,1])   for 'combined'
106  %
107  %                    For example, if 'ErrorModel' is 'combined', one could
108  %                    use 'ErrorParameters' as [1,2]. If not specified, the
109  %                    default values mentioned in brackets above will be used
110  %                    as initial estimates.
111  %
112  % The output ERRORMODELINFO is a structure with the following fields:
113  %
114  %                 ErrorModel   Chosen error model (default = 'constant')
115  %                 ErrorParameters   Estimated error parameters
116  %                 ErrorVariance   A function handle that accepts a n by p
117  %                                 matrix X and computes a n by 1 vector of
118  %                                 error variances using the specified error
119  %                                 model.
120  %                           MSE   Mean squared error.
121  %               ScheffeSimPred   Scheffe parameter for a simultaneous
122  %                                 prediction interval when using this error
123  %                                 model.
124  %               WeightFunction   True if a custom weight function was used
125  %                                 previously in nlinfit.
126  %                 FixedWeights   True if fixed weights were used previously
127  %                                 in nlinfit.
128  %         RobustWeightFunction   True if a robust fitting was used previously
129  %                                 in nlinfit.
130  %
131  %   When 'ErrorModel' is 'combined' or 'proportional', R and J can no
132  %   longer be interpreted as model fit residuals and model function
133  %   Jacobian respectively. When J has full column rank, outputs MSE and R
134  %   are related by:
135  %                MSE = (R'*R)/(n-p) where (n-p) = observations-parameters
136  %   and COVB is related to J and MSE by:
137  %                COVB = inv(J'*J) * MSE
138  %
139  %   NLINFIT treats NaNs in Y or MODELFUN(BETA0,X) as missing data, and
140  %   ignores the corresponding observations.
141  %
142  %   Examples:
143  %
144  %    MODELFUN can be an anonymous function:
145  %       load reaction;
146  %       fun = @(beta,x)(beta(1)*x(:,2) - x(:,3)/beta(5)) ./ ...
```

```
147 %                       (1+beta(2)*x(:,1)+beta(3)*x(:,2)+beta(4)*x(:,3));
148 %       beta = nlinfit(reactants,rate,fun,beta);
149 %
150 %      MODELFUN can be a function on the path specified using @:
151 %         load reaction;
152 %         beta = nlinfit(reactants,rate,@hougen,beta);
153 %
154 %      For an example of weighted fitting, see the example "Weighted
155 %      Nonlinear Regression".
156 %
157 %   See also NLPARCI, NLPREDCI, NLMEFIT, NLINTOOL, STATSET.
158
159 %   References:
160 %      [1] Seber, G.A.F, and Wild, C.J. (1989) Nonlinear Regression, Wiley.
161
162 %   NLINFIT can be used to make a weighted fit with known weights:
163 %
164 %   load reaction;
165 %   w = [8 2 1 6 12 9 12 10 10 12 2 10 8]'; % some example known weights
166 %   ratew = sqrt(w).*rate;
167 %   mymodelw = @(beta,X) sqrt(w).*mymodel(beta,X);
168 %
169 %   [betaw,residw,Jw] = nlinfit(reactants,ratew,mymodelw,beta);
170 %   betaciw = nlparci(betaw,residw,Jw);
171 %   [ratefitw, deltafitw] = nlpredci(@mymodel,reactants,betaw,residw,Jw);
172 %   rmse = norm(residw) / (length(w)-length(rate))
173 %
174 %   Predict at the observed x values.  However, the prediction band
175 %   assumes a weight (measurement precision) of 1 at these points.
176 %
177 %      [ratepredw, deltapredw] = ...
178 %      nlpredci(@mymodel,reactants,betaw,residw,Jw,[],[],'observation');
179
180 %   Copyright 1993-2014 The MathWorks, Inc.
181
182
183 if nargin < 4
184     error(message('stats:nlinfit:TooFewInputs'));
185 elseif ~isvector(y)
186     error(message('stats:nlinfit:NonVectorY'));
187 end
188
189 % Parse input arguments
190 [errormodel, weights, errorparam, options, iterative, maxweight] =
191
192 % Check sizes of the model function's outputs while initializing the fitted
193 % values, residuals, and SSE at the given starting coefficient values.
```

```
194 model = fcnchk(model);
195 try
196     yfit = model(beta,X);
197 catch ME
198  if isa(model, 'inline')
199    m = message('stats:nlinfit:ModelInlineError');
200     throw(addCause(MException(m.Identifier,'%s',getString(m)),ME));
201  elseif strcmp('MATLAB:UndefinedFunction', ME.identifier) ...
202           && ~isempty(strfind(ME.message, func2str(model)))
203    error(message('stats:nlinfit:ModelFunctionNotFound', func2str( model )));
204     else
205         m = message('stats:nlinfit:ModelFunctionError',func2str(model));
206         throw(addCause(MException(m.Identifier,'%s',getString(m)),ME));
207     end
208 end
209 if ~isequal(size(yfit), size(y))
210     expSizeStr = sprintf('%-d-by-',size(y));
211     actSizeStr = sprintf('%-d-by-',size(yfit));
212     error(message('stats:nlinfit:WrongSizeFunOutput',
213 end
214
215 % Check weights
216 nanweights = checkWeights(weights, yfit, y);
217
218 if strcmp(errormodel, 'exponential')
219     % Transform model.
220     [y, model] = applyLogTransformation(y, model);
221     % Transform yfit as well.
222     [yfit, ~] = applyLogTransformation(yfit, []);
223 end
224
225 % Find NaNs in either the responses or in the fitted values at the starting
226 % point.  Since X is allowed to be anything, we can't just check for rows
227 % with NaNs, so checking yhat is the appropriate thing.  Those positions in
228 % the fit will be ignored as missing values.  NaNs that show up anywhere
229 % else during iteration will be treated as bad values.
230 nans = (isnan(y(:)) | isnan(yfit(:)) | nanweights(:)); % a col vector
231 r = y(:) - yfit(:);
232 r(nans) = [];
233 n = numel(r);
234 p = numel(beta);
235 sse = r'*r;
236
237 % After removing NaNs in either the responses or in the fitted values at
238 % the starting point, if n = 0 then stop execution.
239 if ( n == 0 )
240     error(message('stats:nlinfit:NoUsableObservations'));
```

```
241 end
242
243 funValCheck = strcmp(options.FunValCheck, 'on');
244 if funValCheck && ~isfinite(sse), checkFunVals(r); end
245
246 % Set the level of display
247 switch options.Display
248     case 'off',    verbose = 0;
249     case 'notify', verbose = 1;
250     case 'final',  verbose = 2;
251     case 'iter',   verbose = 3;
252 end
253 maxiter = options.MaxIter;
254 robustw = [];
255
256 if isempty(options.RobustWgtFun)
257     % display format for non-robust fit
258     if verbose > 2 % iter
259         disp(' ');
260         disp('                                        Norm of         Norm of');
261         disp('   Iteration             SSE        Gradient           Step ');
262         disp('  ------------------------------------------------------------');
263         disp(sprintf('      %6d    %12g',0,sse)); %#ok<DSPS>
264     end
265
266     if iterative
267         % Iteratively re-weighted fitting
268 [beta,J,cause,errorparam,fullr] = nlweightedfit(X,y,beta,model,options,          v
269     else
270         % Known weights
271         % Cap the weights at max weight
272         weights(weights > maxweight) = maxweight;
273
274         w = sqrt(weights);
275         yw = y .* w;
276         modelw = @(b,x) w.*model(b,x);
277         [beta,J,~,cause,fullr] = LMfit(X,yw, modelw,beta,options,
278     end
279
280 else
281     % Allow for empty or scalar weights
282     if isempty(weights)
283         weights = ones(size(y));
284     elseif isscalar(weights)
285         weights = repmat(weights,size(y));
286     end
287
```

```
288       % Do a preliminary fit just to get residuals and leverage from the
289       % least squares coefficient.  We won't count this against the iteration
290       % limit.
291       [beta_ls,J] = LMfit(X,y, model,beta,options,0,maxiter,weights);
292       res = y - model(beta_ls,X);
293       res(isnan(res)) = [];
294       ols_s = norm(res) / sqrt(max(1,length(res)-numel(beta)));
295
296       % display format for robust fit
297       % Please note there are two loops for robust fit. It would be very
298       % confusing if we display all iteration results. Instead, only the last
299       % step of each inner loop (LM fit) will be output.
300       if verbose > 2 % iter
301           disp(' ');
302           disp(getString(message('stats:nlinfit:
303           disp(' ');
304           disp('   Iteration             SSE ');
305           disp('  ------------------------------');
306           disp(sprintf('      %6d    %12g',0,sse)); %#ok<DSPS>
307       end
308       [beta,J,sig,cause,fullr,robustw] = nlrobustfit(X,y,beta,model,J,ols_s,
309   end;
310
311   switch(cause)
312       case 'maxiter'
313           warning(message('stats:nlinfit:IterationLimitExceeded'));
314       case 'tolx'
315           if verbose > 1 % 'final' or 'iter'
316               disp(getString(message('stats:nlinfit:TerminatedRelativeNorm')));
317           end
318       case 'tolfun'
319           if verbose > 1 % 'final' or 'iter'
320               disp(getString(message('stats:nlinfit:
321           end
322       case 'stall'
323           warning(message('stats:nlinfit:UnableToDecreaseSSE'));
324   end
325
326   % If the Jacobian is ill-conditioned, then it's likely that two parameters
327   % are aliased and the estimates will be highly correlated.  Prediction at
328   % new x values not in the same column space is dubious.  NLPARCI will have
329   % trouble computing CIs because the inverse of J'*J is difficult to get
330   % accurately. NLPREDCI will have the same difficulty, and in addition,
331   % will in effect end up taking the difference of two very large, but nearly
332   % equal, variance and covariance terms, lose precision, and so the
333   % prediction bands will be erratic. It may also be that the Jacobian has
334   % one or more columns of zeros, meaning model is constant with respect to
```

```
% one or more parameters.  This may be because those parameters are not
% even in the expression in the model function, or they are multiplied by
% another param that is estimated at exactly zero (or something similar),
% or because some part of the model function is underflowing, making it a
% constant zero.

% Final J might have Inf/NaN values or may not be of the right size due to
% NaN removal in intermediate steps if FunValCheck was 'off'. If so, this
% is an error since no meaningful statistics can be computed from such a J.
% If FunValCheck is 'on', we know that the Jacobian must be good.
if ( funValCheck == false )
    nonfiniteJ = any(~isfinite(J(:)));
    wrongsizeJ = ( size(J,1) ~= n || size(J,2) ~= p );
    if ( nonfiniteJ || wrongsizeJ )
        warning(message('stats:nlinfit:BadFinalJacobian'));
        % [beta,r,J,Sigma,mse,errorModelInfo,robustw] = nlinfit(...)
        % beta will *not* have the converged value but some intermediate
        % value. Get "full" r. Fill in NaN's for J, Sigma and mse.
        % Use sch = p + 1 to get errorModelInfo. Make robustw all NaN's.
        r = fullr;
        J = NaN(numel(fullr),p);
        Sigma = NaN(p,p);
        mse = NaN;
        sch = p + 1; % conservative setting.
        errorModelInfo = fillErrorModelInfo(sch, errormodel, errorparam,
        robustw = NaN(size(robustw));
        return;
    end
end

% Get QR factorization of J and mark if J is ill-conditioned.
[Q,R] = qr(J,0);
illConditionedJ = false;
if n <= p
    illConditionedJ = true;
    warning(message('stats:nlinfit:Overparameterized'));
elseif condest(R) > 1/sqrt(eps(class(beta)))
    illConditionedJ = true;
    if any(all(abs(J)<sqrt(eps(norm(J,1))),1),2)
        warning(message('stats:nlinfit:ModelConstantWRTParam'));
    else
        % no columns of zeros
        warning(message('stats:nlinfit:IllConditionedJacobian'));
    end
end

% We have beta, J, fullr and we need [beta,r,J,Sigma,mse,errorModelInfo].
```

```
382  % Inspect nargout and compute required quantities.
383  if nargout > 1
384      % Return residuals and Jacobian that have missing values where needed.
385      r = fullr;
386      JJ(~nans,:) = J;
387      JJ(nans,:) = NaN;
388      J = JJ;
389  end
390
391  if nargout > 3
392      % Get rankJ, pinvJTJ and VQ using either:
393      % (a) SVD - if J is ill-conditioned or
394      % (b)  QR - if J is well-conditioned. In this case, we can set VQ equal
395      %           to Q from QR factorization of J to test the Scheffe parameter
396      %           condition.
397      if illConditionedJ
398          % Call isEstimable.
399          TolSVD = eps(class(beta));
400          [~,rankJ,pinvJTJ,~,~,VQ] = internal.
401      else
402          % Use existing QR factorization.
403          rankJ = size(J,2);
404          Rinv = R \ eye(size(R));
405          pinvJTJ = Rinv*Rinv';
406          VQ = Q;
407      end
408
409      % Get MSE using either residuals or from Robust fit.
410      if isempty(options.RobustWgtFun)
411          % Can use residuals to get MSE.
412          % The denominator degrees of freedom is (n-rankJ).
413          mse = sum(abs(r(~nans)).^2)/(n-rankJ);
414      else
415          % Using Robust fitting.
416          mse = sig.^2;
417      end
418
419      % Get Sigma, the co-variance matrix of beta.
420      % Sigma may be singular when J is ill-conditioned.
421      Sigma = mse*pinvJTJ;
422  end
423
424  if nargout > 5
425      % Estimate the Scheffe parameter for simultaneous prediction intervals.
426      % We will pass in VQ from above to avoid an SVD computation in
427      if ~isa(weights,'function_handle') && isnumeric(weights)
428          % Fixed weight vector. Conservative setting is:
```

```
429           % can do better than this.
430           EVFit = 1./weights(~nans); EVPred = ones(size(EVFit));
431           sch = internal.stats.getscheffeparam('WeightedJacobian',
432       else
433          % Either weights is a function_handle or errormodel =
434           sch = internal.stats.getscheffeparam('WeightedJacobian',
435       end
436
437       % Create the structure ErrorModelInfo.
438       errorModelInfo = fillErrorModelInfo(sch, errormodel,
439   end
440
441   end % function nlinfit
442
443   %==== subfunction fillErrorModelInfo ====
444   function S = fillErrorModelInfo(sch, errormodel, errorparam,
445       %  Initialize output structure with the following fields:
446       %  ErrorModel           - Name of the error model.
447       %  ErrorParameters      - Parameters of the error model.
448       %  ErrorVariance        - Variance function for the error model.
449       %  MSE                  - Mean squared error.
450       %  ScheffeSimPred       - Scheffe parameter for simultaneous
451       %  WeightFunction       - True if using 'Weights' with a weight
452       %  FixedWeights         - True if using 'Weights' with a fixed
453       %  RobustWeightFunction - True if using Robust fitting
454       S = struct('ErrorModel',[],'ErrorParameters',[],
455
456    % Set the fields of S. We will burn in the values of beta, model, mse,
457    % errorparam and weights when creating function handle ErrorVariance.
458      switch lower(errormodel)
459        case 'combined'
460        % Note that errorparam is already set during model fitting.
461          S.ErrorModel = 'combined';
462          S.ErrorParameters = errorparam;
463          S.MSE = mse;
464          S.ErrorVariance = @(x) mse * ( errorparam(1) +
465
466        case 'proportional'
467            % We must set errorparam explicitly.
468            errorparam = sqrt(mse);
469            S.ErrorModel = 'proportional';
470            S.ErrorParameters = errorparam;
471            S.MSE = mse;
472            S.ErrorVariance = @(x) mse * ( abs(model(beta,x)) ).^2;
473
474     case 'exponential'
475      % This is the 'constant' variance model but *after* log transformation.
```

```
476         errorparam = sqrt(mse);
477         S.ErrorModel = 'exponential';
478         S.ErrorParameters = errorparam;
479         S.MSE = mse;
480         S.ErrorVariance = @(x) mse * ones(size(x,1),1);
481
482         case 'constant'
483           if isa(weights, 'function_handle')
484           % If weights is a function handle, we need to compute
485           % sigma^2 * diag(W^{-1}) as the ErrorVariance function.
486           errorparam = sqrt(mse);
487           S.ErrorModel = 'constant';
488           S.ErrorParameters = errorparam;
489           S.MSE = mse;
490           S.ErrorVariance = @(x) mse * (1./weights(model(beta,x)));
491           S.WeightFunction = true;
492           elseif isnumeric(weights) && ~isscalar(weights) && ~all(weights==1)
493           % Known weight vector.
494           errorparam = sqrt(mse);
495           S.ErrorModel = 'constant';
496           S.ErrorParameters = errorparam;
497           S.MSE = mse;
498           S.ErrorVariance = @(x) mse * ones(size(x,1),1);
499           S.FixedWeights = true;
500             else
501           % The 'constant' variance model, maybe using robust regression.
502             errorparam = sqrt(mse);
503             S.ErrorModel = 'constant';
504             S.ErrorParameters = errorparam;
505             S.MSE = mse;
506             S.ErrorVariance = @(x) mse * ones(size(x,1),1);
507              end
508            if ~isempty(options.RobustWgtFun)
509               % Robust regression.
510            S.RobustWeightFunction = true;
511              end
512          otherwise
513          % This error model is not known - we should never get here.
514           error(message('stats:nlinfit:InvalidErrorModel'));
515       end
516
517   end % End of fillErrorModelInfo.
518
519   %-------------------------------------------------------------------
520   function  [beta,J,iter,cause,fullr] = LMfit(X,y, model,beta,options,
521   % Levenberg-Marquardt algorithm for nonlinear regression
522
```

```
523  % Set up convergence tolerances from options.
524  betatol = options.TolX;
525  rtol = options.TolFun;
526  fdiffstep = options.DerivStep;
527  if isscalar(fdiffstep)
528      fdiffstep = repmat(fdiffstep, size(beta));
529  else
530      % statset ensures fdiffstep is not a matrix.
531      % Here, we ensure fdiffstep has the same shape as beta.
532      fdiffstep = reshape(fdiffstep, size(beta));
533  end
534  funValCheck = strcmp(options.FunValCheck, 'on');
535
536  % Set initial weight for LM algorithm.
537  lambda = .01;
538
539  % Set the iteration step
540  sqrteps = sqrt(eps(class(beta)));
541
542  p = numel(beta);
543
544  if nargin<8 || isempty(weights)
545      sweights = ones(size(y));
546  else
547      sweights = sqrt(weights);
548  end
549
550  % treatment for nans
551  yfit = model(beta,X);
552  fullr = sweights(:) .* (y(:) - yfit(:));
553  nans = isnan(fullr); % a col vector
554  r = fullr(~nans);
555  sse = r'*r;
556
557  zerosp = zeros(p,1,class(r));
558  iter = 0;
559  breakOut = false;
560  cause = '';
561
562  while iter < maxiter
563      iter = iter + 1;
564      betaold = beta;
565      sseold = sse;
566
567      % Compute a finite difference approximation to the Jacobian
568      J = getjacobian(beta,fdiffstep,model,X,yfit,nans,sweights);
569
```

```
570       % Levenberg-Marquardt step: inv(J'*J+lambda*D)*J'*r
571       diagJtJ = sum(abs(J).^2, 1);
572       if funValCheck && ~all(isfinite(diagJtJ)), checkFunVals(J(:)); end
573       Jplus = [J; diag(sqrt(lambda*diagJtJ))];
574       rplus = [r; zerosp];
575       step = Jplus \ rplus;
576       beta(:) = beta(:) + step;
577
578       % Evaluate the fitted values at the new coefficients and
579       % compute the residuals and the SSE.
580       yfit = model(beta,X);
581       fullr = sweights(:) .* (y(:) - yfit(:));
582       r = fullr(~nans);
583       sse = r'*r;
584       if funValCheck && ~isfinite(sse), checkFunVals(r); end
585       % If the LM step decreased the SSE, decrease lambda to downweight the
586       % steepest descent direction.  Prevent underflowing to zero after many
587       % successful steps; smaller than eps is effectively zero anyway.
588       if sse < sseold
589           lambda = max(0.1*lambda,eps);
590
591       % If the LM step increased the SSE, repeatedly increase lambda to
592       % upweight the steepest descent direction and decrease the step size
593       % until we get a step that does decrease SSE.
594       else
595           while sse > sseold
596               lambda = 10*lambda;
597               if lambda > 1e16
598                   breakOut = true;
599                   break
600               end
601               Jplus = [J; diag(sqrt(lambda*sum(J.^2,1)))];
602               step = Jplus \ rplus;
603               beta(:) = betaold(:) + step;
604               yfit = model(beta,X);
605               fullr = sweights(:) .* (y(:) - yfit(:));
606               r = fullr(~nans);
607               sse = r'*r;
608               if funValCheck && ~isfinite(sse), checkFunVals(r); end
609           end
610       end
611       if verbose > 2 % iter
612           disp(sprintf('      %6d    %12g    %12g    %12g', ...
613               iter,sse,norm(2*r'*J),norm(step))); %#ok<DSPS>
614       end
615
616       % Check step size and change in SSE for convergence.
```

```
617       if norm(step) < betatol*(sqrteps+norm(beta))
618           cause = 'tolx';
619           break
620       elseif abs(sse-sseold) <= rtol*sse
621           cause = 'tolfun';
622           break
623       elseif breakOut
624           cause = 'stall';
625           break
626       end
627   end
628   if (iter >= maxiter)
629       cause = 'maxiter';
630   end
631   end % function LMfit
632
633   %-------------------------------------------------------------------------
634   function checkFunVals(v)
635   % check if the function has finite output
636   if any(~isfinite(v))
637       error(message('stats:nlinfit:NonFiniteFunOutput'));
638   end
639   end % function checkFunVals
640
641   %-------------------------------------------------------------------------
642   function [beta,Jw,cause,errorModelParam,fullr]=nlweightedfit(x,y,beta,          mo
643   % iteratively re-weighted fit
644
645   betatol = options.TolX;
646   sqrteps = sqrt(eps(class(beta)));
647
648   if strcmpi(errorModel, 'combined')
649    % Do a prelimary unweighted fit before calculating [a b]
650       beta = LMfit(x,y,model,beta,options,0,maxiter);
651   end
652
653   yfit = model(beta,x);
654   fullr = y(:) - yfit(:);
655   ok = ~isnan(fullr);
656
657   % Main loop of repeated nonlinear fits, adjust weights each time
658   totiter = 0;
659   w = NaN(size(y));
660   while maxiter>0
661       beta0=beta;
662
663    % Update weights
```

```
664     yfit = model(beta,x);
665     switch errorModel
666         case 'proportional'
667             w = 1./abs(yfit);
668         case 'combined'
669             ab = abs(fminsearch(@(ab) error_ab(ab,y,yfit),errorModelParam));
670             w = 1./abs(ab(1)+ab(2)*abs(yfit));
671             errorModelParam = ab;
672         case 'constant'
673             % You dont get here until weights is a function handle
674             w = realsqrt(wgtfun(yfit));
675       end
676
677       % Cap the weights at max weight
678       w(w > sqrt(maxweight)) = sqrt(maxweight);
679
680       % this is weighted nlinfit
681       yw = y .* w;
682       modelw = @(b,x) w.*model(b,x);
683       [beta,Jw,lsiter,cause,fullr] = LMfit(x,yw,modelw,beta0,options,
684       totiter = totiter + lsiter;
685       maxiter = maxiter - lsiter;
686       r = fullr(ok);
687
688       % if there is no change in any coefficient, the iterations stop.
689       if norm(beta-beta0) < betatol*(sqrteps+norm(beta))
690           cause = 'tolx';
691           break;
692       end
693
694       if verbose > 2 % iter
695           disp(sprintf('      %6d    %12g', ...
696               totiter, r'*r)); %#ok<DSPS>
697       end
698   end
699
700   % this is a warning about the non-convergence
701   if maxiter<=0
702       cause = 'maxiter';
703   end
704
705   end % function nlweighted fit
706
707   %-------------------------------------------------------------------------
708   function [beta,J,sig,cause,fullr,w]=nlrobustfit(x,y,beta,model,J,
709   % nonlinear robust fit
710
```

```
711   tune = options.Tune;
712   WgtFun = options.RobustWgtFun;
713   [WgtFun,tune] = statrobustwfun(WgtFun,tune);
714
715   yfit = model(beta,x);
716   fullr = y - yfit;
717   ok = ~isnan(fullr(:));
718   r = fullr(:);
719   pweights = pweights(:);
720   r = sqrt(pweights(ok)).*r(ok);
721   Delta = sqrt(eps(class(beta)));
722
723   % Adjust residuals using leverage, as advised by DuMouchel & O'Brien
724   % Compute leverage based on X, the Jacobian
725   [Q,~]=qr(J,0);
726   h = min(.9999, sum(Q.*Q,2));
727
728   % Compute adjustment factor
729   adjfactor = 1 ./ sqrt(1-h);
730
731   radj = r .* adjfactor;
732
733   % If we get a perfect or near perfect fit, the whole idea of finding
734   % outliers by comparing them to the residual standard deviation becomes
735   % difficult.  We'll deal with that by never allowing our estimate of the
736   % standard deviation of the error term to get below a value that is a small
737   % fraction of the standard deviation of the raw response values.
738   tiny_s = 1e-6 * std(y);
739   if tiny_s==0
740       tiny_s = 1;
741   end
742
743   % Main loop of repeated nonlinear fits, adjust weights each time
744   totiter = 0;
745   w = NaN(size(y));
746   while maxiter>0
747       beta0=beta;
748       s = madsigma(radj, numel(beta)); % robust estimate of sigma for residual
749
750       % Compute robust weights based on current residuals
751       w(ok) = feval(WgtFun, radj/(max(s,tiny_s)*tune));
752
753       % this is the weighted nlinfit
754       allweights = w(:).*pweights;
755       allweights = reshape(allweights,size(y));
756       [beta,~,lsiter,cause] = LMfit(x,y,model,beta0,options,0,
757       totiter = totiter + lsiter;
```

```
758       maxiter = maxiter - lsiter;
759       yfit = model(beta,x);
760       fullr = y - yfit;
761       r = fullr(:);
762       r = sqrt(pweights(ok)).*r(ok);
763       radj = r .* adjfactor;
764
765       % if there is no change in any coefficient, the iterations stop.
766       if  all(abs(beta-beta0) < Delta*max(abs(beta),abs(beta0)))
767           break;
768       end
769
770       if verbose > 2 % iter
771           disp(sprintf('      %6d    %12g', ...
772               totiter, r'*r)); %#ok<DSPS>
773       end
774   end
775
776   % this is a warning about the non-convergence
777   if maxiter<=0
778       cause = 'maxiter';
779   end
780
781   % We need the Jacobian at the final coefficient estimates, but not the J1
782   % version returned by LMfit because it has robust weights included
783   fdiffstep = options.DerivStep;
784   if isscalar(fdiffstep)
785       fdiffstep = repmat(fdiffstep,size(beta));
786   end
787   J = getjacobian(beta,fdiffstep,model,x,yfit,~ok,reshape(sqrt(pweights),
788
789   % Compute MAD of adjusted residuals after dropping p-1 closest to 0
790   p = numel(beta);
791   n = length(radj);
792   mad_s = madsigma(radj, p);
793
794   % Compute a robust scale estimate for the covariance matrix
795   sig = statrobustsigma(WgtFun,radj,p,mad_s,tune,h);
796
797   % Be conservative by not allowing this to be much bigger than the ols value
798   % if the sample size is not large compared to p^2
799   sig = max(sig, ...
800       sqrt((ols_s^2 * p^2 + sig^2 * n) / (p^2 + n)));
801   end % function nlrobustfit
802
803   %---------------------- Robust estimate of sigma
804   function s = madsigma(r,p)
```

```
805 %MADSIGMA    Compute sigma estimate using MAD of residuals from 0
806 n = length(r);
807 rs = sort(abs(r));
808 s = median(rs(max(1,min(n,p)):end)) / 0.6745;
809 end % function madsigma
810
811 % ---------------------- Jacobian
812 function J = getjacobian(beta,fdiffstep,model,X,yfit,nans,sweights)
813     function yplus = call_model_nested(betaNew)
814         yplus = model(betaNew, X);
815         yplus(nans) = [];
816     end
817 J = statjacobian(@call_model_nested, beta, fdiffstep, yfit(~nans));
818 if ~isempty(sweights)
819     sweights = sweights(~nans);
820     J = bsxfun(@times,sweights(:),J);
821 end
822 end % function getjacobian
823
824 %----------------------- Parse input arguments
825 function [errModel, weights, errModelParameter, options,                    i
826 % Parse input arguments
827
828 argsToParse = varargin{:};
829 options = statset('nlinfit');
830 iterative = false;
831
832 % Process PVP - ErrorModel, Weights and ErrorModelParameter inputs should
833 % be passed in as PV pairs. statset can be passed in either as PV pair or
834 % directly as a struct.
835 numargs = numel(argsToParse);
836 if numargs > 0 && rem(numargs,2)
837     % statset supplied directly as a struct.
838     if isstruct(argsToParse{1}) || isempty(argsToParse{1})
839         options = argsToParse{1};
840         argsToParse = argsToParse(2:end);
841     end
842 end
843
844 % Parse PV pairs
845 pnames = {'errormodel','weights', 'options', 'errorparameters', 'maxweight'};
846 defval = {'constant', 1, options, [], realmax};
847 [errModel, weights, options, errModelParameter, maxweight] = ...
848     internal.stats.parseArgs(pnames, defval, argsToParse{:});
849
850 % Validate property values
851 if ~ischar(errModel)
```

```
852         error(message('stats:nlinfit:InvalidErrorModel'));
853     end
854     ok = {'constant', 'proportional', 'exponential', 'combined'};
855     okv = find(strncmpi(errModel, ok, numel(errModel)));
856     if numel(okv) ~= 1
857         error(message('stats:nlinfit:InvalidErrorModel'));
858     end
859     errModel = ok{okv};
860
861     if ~isnumeric(weights) && ~isa(weights, 'function_handle')
862         error(message('stats:nlinfit:InvalidWeights'));
863     end
864
865     options = statset(statset('nlinfit'),options);
866
867     if numel(errModelParameter)>2 || ~isnumeric(errModelParameter)
868         error(message('stats:nlinfit:BadErrorParam'))
869     end
870     switch errModel
871         case 'combined'
872             if isempty(errModelParameter)
873                 errModelParameter = [1 1];
874             elseif numel(errModelParameter)~= 2
875      % For combined error model, ErrorModelParameter should be a vector [a b]
876                 error(message('stats:nlinfit:BadCombinedParam', errModel));
877             end
878         case 'proportional'
879             % Only a should be specified.
880             if isempty(errModelParameter)
881                 errModelParameter = 1;
882             elseif numel(errModelParameter)~=1
883                 error(message('stats:nlinfit:BadErrorParam1', errModel))
884             end
885         case {'constant', 'exponential'}
886             % Only b should be specified.
887             if isempty(errModelParameter)
888                 errModelParameter = 1;
889             elseif numel(errModelParameter)~=1
890                 error(message('stats:nlinfit:BadErrorParam1', errModel))
891             end
892     end
893
894     if ~isscalar(maxweight) || ~isreal(maxweight) || maxweight<=0
895         error(message('stats:nlinfit:InvalidMaxWeight'));
896     end
897
898     % Check for conflicting error model and weights
```

```
899   if ~strcmpi(errModel, 'constant')
900       if isa(weights, 'function_handle') || ~isscalar(weights) || weights~=1
901           error(message('stats:nlinfit:ErrorModelWeightConflict'));
902       end
903   end
904
905   % Robust fitting and weights
906   if ~isempty(options.RobustWgtFun) && (~strcmpi(errModel, 'constant') ||
907       error(message('stats:nlinfit:ErrorModelRobustConflict'));
908   end
909
910   if any(strcmpi(errModel, {'proportional', 'combined'})) || isa(weights,
911       % Iteratively reweighted fitting required for proportional and
912       % combined error model and weights that are a function of
913       % predicted values
914       iterative = true;
915   end
916
917   end % function parseInVarargin
918
919   %----------------------- Check weights
920   function nanweights = checkWeights(weights, yfit, y)
921   nanweights = zeros(size(y));
922
923   if (isnumeric(weights) && (~isscalar(weights) || weights~=1)) ||
924       % If weights are set
925
926     if isa(weights, 'function_handle')
927         % function handle
928         try
929           wVec = weights(yfit);
930         catch ME
931         if isa(weights, 'inline')
932           m = message('stats:nlinfit:InlineWeightFunctionError');
933             throw(addCause(MException(m.Identifier,'%s',getString(m)),ME));
934               elseif strcmp('MATLAB:UndefinedFunction', ME.identifier) ...
935                 && ~isempty(strfind(ME.message, func2str(weights)))
936                  error(message('stats:nlinfit:WeightFunctionNotFound',
937               else
938                   m = message('stats:nlinfit:WeightFunctionError',
939                   throw(addCause(MException(m.Identifier,'%s',
940               end
941           end
942       else
943           wVec = weights; % fixed weights
944       end
945
```

```
946       nanweights = isnan(wVec);
947       % w should be real positive vector of the same size as y
948     if any(~isreal(wVec)) || any(wVec(~nanweights)<=0) ||
949        error(message('stats:nlinfit:InvalidWeights'));
950       end
951
952   end
953
954   end % function validateWeights
955
956   function e = error_ab(ab,y,f)
957   g = abs(ab(1)) + abs(ab(2))*abs(f);
958   e = sum(0.5*((y-f)./g).^2 + log(g));
959   end % function error_ab
960
961   function [y, model] = applyLogTransformation(y, model)
962   % Exponential, y = f*exp(a*e), or log(y) = log(f) + a*e
963
964   if ~isempty(y)
965       if ~all(y>0)
966           error(message('stats:nlinfit:PositiveYRequired'));
967       else
968           y = log(max(y,realmin));
969       end
970   end
971
972   if ~isempty(model)
973       % Exponential error model. Linearize the model as
974       %   y = f*exp(a*e), or log(y) = log(f) + a*e
975       model = @(phi,X) log(max(model(phi,X),realmin));
976   end
977
978   end % function applyModelTransformations
979
```

Reference

[1] Buyukgungor, H., Gurel, L. 2009. The Role of Biotechnology on the Treatment of Wastes. African Journal of Biotechnology, Vol. 8(25), pp. 7253-7262. ISSN 1684-5315.

[2] Alam, P., Ahmade, K. 2013. Impact of Solid Waste on Health and the Environment. Special Issue of International Journal of Sustainable Development and Green Economics (IJSDGE), Vol. 2(1), pp. 165-168. ISSN 2315-4721.

[3] Dahiya, R. 2015. Projections for the Population Growth and Its Impact on Solid Waste Generation of a Medium Sized North Indian City. International Journal of Technical Research and Applications, Vol. 3(6), pp. 57-61. e-ISSN 2320-8163.

[4] Gebril Ali, A.O. 2013. Solid Waste Pollution and the Importance of Environmental Planning in Managing and Preserving the Public Environment in Benghazi City and Its Surrounding Areas. International Journal of Environmental, Chemical, Ecological, Geological and Geophysical Engineering, Vol. 7(12), pp. 924-929.

[5] Amani, T., Nosrati, M., Sreekrishnan, T.R. 2010. Anaerobic Digestion from the Viewpoint of Microbiological, Chemical, and Operational Aspects - a review. Environ. Rev. Vol. 18, pp. 255-278. doi. 10.1139/A10-011.

[6] Khanal, S.K., 2008. Anaerobic Biotechnology for Bioenergy Production, Principles and Applications. John Wiley and Sons, Inc. ISBN: 978-0-813-82346-1.

[7] Simeonov, Iv., Momchev, V., Grancharov, D. 1996. Dynamic Modeling of Mesophilic Anaerobic Digestion of Animal Waste. Wat. Res. Vol. 30(5), pp. 1087-1094.

[8] Bouallagui, H., Ben Cheikh, R., Marouani, L., Hamdi, M., 2003. Mesophilic Biogas Production from Fruit and Vegetable Waste in a Tubular Digester. Bioresource Technology, Vol. 86, 85-89.

[9] Bouallagui, H., Torrijos, M., Godon, J.J., Moletta, R., Ben Cheikh, R., Touhami, Y., Delgenes, J.P., Hamdi, M. 2004. Short Communication. Two-Phases Anaerobic Digestion of Fruit and Vegetable Wastes: Bioreactor Performance. Biochemical Engineering Journal, Vol. 21, pp. 193-197.

[10] Bouallagui, H., Haouari, O., Touhami, Y., Ben Cheikh, R., Marouani, L., Hamdi, M. 2004. Effect of Temperature of an Anaerobic Tubular Reactor Treating Fruit and Vegetable Waste. Process Biochemistry, Vol. 39, pp. 2143-2148.

[11] Bouallagui, H., Touhami, Y., Ben Cheikh, R., Hamdi, M., 2005. Bioreactor Performance in Anaerobic Digestion of Fruit and Vegetable Wastes. Process Biochemistry, Vol. 40, pp. 989-995.

[12] Bouallagui, H., Lahdheb, H., Ben Romdan, E., Rachdi, B., Hamdi, M., 2009. Improvement of Fruit and Vegetable Waste Anaerobic Digestion Performance and Stability with Co-Substrates Addition. Journal of Environmental Management, Vol. 90, pp. 1844-1849.

[13] Luwers, J., Appels, L., Thompson, I.P., Degreve, J., Van Impe, J.F., Dewil, R. 2013. Mathematical Modelling of Anaerobic Digestion of Biomass and Waste: Power and Limitations. Progress in Energy and Combustion Science, Vol. 39, pp. 383-402.

[14] Monod, J. 1949. The Growth of Bacterial Cultures. Annual Reviews of Microbiology, Vol. 3, pp. 371-394.

[15] Contois, D. E. 1959. Kinetics of Bacterial Growth: Relationship between Population Density and Specific Growth Rate of Continuous Cultures. Journal of General Microbiology, Vol. 21, pp. 40-50.

[16] Chen, Y. R., Hashimoto, A. G. 1978. Kinetics of Methane Fermentation. Biotechnology and Bioengineering Symposium, No. 8, pp. 269-282.

[17] Hill, D. T., Barth, C. L. 1977. A Dynamic Model for Simulation of Animal Waste Digestion. Journal of Water Pollution Control Federation, Vol. 10, pp. 2129-2143.

[18] Hill, D. T. 1982. A Comprehensive Dynamic Model for Animal Waste Methanogenesis. Transactions of the ASAE, pp. 1374-1380.

[19] Angelidaki, I., Ellegaard, L., Ahring, B.K. 1993. A Mathematical Model for Dynamic Simulation of Anaerobic Digestion of Complex Substrates: Focusing on Ammonia Inhibition. Biotechnology and Bioengineering, Vol. 42, pp. 159-166.

[20] Angelidaki, I., Ellegaard, L., Ahring, B. K. 1999. A Comprehensive Model of Anaerobic Bioconversion of Complex Substrates to Biogas. Biotechnology and Bioengineering, Vol. 63(3), pp. 363-372.

[21] Siegrist, H., Renggli, D., Gujer, W. 1993. Mathematical-Modeling of Anaerobic Mesophilic Sewage-Sludge Treatment. Water Science and Technology, Vol. 27(2), pp. 25-36.

[22] Siegrist, H., Vogt, D., Garcia-Heras, J. L., Gujer, W. 2002. Mathematical Model for Meso- and Thermophilic Anaerobic Sewage Sludge Digestion. Environmental Science and Technology, Vol. 36, pp. 1113 - 1123.

[23] Vavilin, V. A., Vasiliev, V. B., Ponomarev, A. V., Rytow, S. V. 1994. Simulation Model 'Methane' as a Tool for Effective Biogas Production during Anaerobic Conversion of Complex Organic Matter. Bioresource Technology, Vol. 48, pp. 1-8.

[24] Vavilin, V. A., Rytow, S. V., Lokshina, L. Y. 1995. Modelling Hydrogen Partial Pressure Change as a Result of Competition between the Butyric and Propionic Groups of Acidogenic Bacteria. Bioresource Technology, Vol. 54(2), pp. 171-177.

[25] Batstone, D., Keller, J., Newell, R., Newland, M. 2000. Modelling Anaerobic Degradation of Complex Wastewater. I: Model Development. Bioresource Technology, Vol. 75, pp. 67-74.

[26] Batstone, D., Keller, J. 2003. Industrial Applications of the IWA Anaerobic Digestion Model No.1 (ADM1). Water Science and Technology, Vol. 47(12), pp. 199-206.

[27] Batstone, D., Keller, J., Angelidaki, I., Kalyuzhnyi, S. V., Pavlostathis, S. G., Rozzi, A., Sanders, W.T., Siegrist, H., Vavilin, V. A. 2002. Anaerobic Digestion Model No.1. (ADM1). Iwa task Group, IWA Task Group for Mathematical Modelling of Anaerobic Digestion Processes, IWA Publishing, London.

[28] Mata-Alvarez, J., Mtz.-Viturtia, A., Llabres-Luengo, P., Cecchi, F., 1993. Kinetic and performance study of fruit and vegetable wastes. Biomass and Bioenergy. Vol. 5(6), pp. 481-488.

[29] P. Sosnowski, Klepacz-Smolka, A., Kaczorek, K., Ledakowicz, S. Kinetic Investigations of Methane Co-Fermentation of Sewage Sludge and Organic Fraction of Municipal Solid Waste. Bioresource Technology, vol. 99, pp. 5731-5737, 2008.

[30] Siles, J.A. Martin, M.A., Chica, A., Borja, R. 2008. Kinetic Modelling of the Anaerobic Digestion of Wastewater Derived from the Pressing of Orange Rind Produced in Orange Juice Manufacturing. Chemical Engineering Journal, Vol. 140, pp. 145-156.

[31] Vavilin, V.A. 2010. Anaerobic Degradation of Organic Waste: An Experience in Mathematical Modeling. Microbiology, Vol. 79(3), pp. 334-341.

[32] Li, C., Champagne, P., Anderson, B.C. 2011. Evaluating and Modeling Biogas production from Municipal Fat, Oil, and Grease and Synthetic Kitchen Waste in Anaerobic Co-digestions. Bioresource Technology, Vol. 102, pp. 9471-9480.

[33] Kafle, G.K., Bhattarai, S., Kim, S.H., Chen, L. 2014. Effect of Feed to Microbe Ratios on Anaerobic Digestion of Chinese Cabbage Waste under Mesophilic and Thermophilic Conditions: Biogas Potential and Kinetic Study. Journal of Environmental Management, Vol. 133, pp. 293-301.

[34] Sitorus, B., Sukandar, Panjaitan, S.D. 2013. Biogas Recovery from Anaerobic Digestion Process of Mixed Fruit-Vegetable Wastes. International Conference on Sustainable Energy and Engineering and Application (ICSEEA 2012). Energy Procedia, Vol. 32, pp. 176-182.

[35] Gerardi, M. H., 2003. The microbiology of anaerobic digester. A John Wiley and Sons, Inc., Publication. ISBN 978-0-471-20693-4.

[36] Gerber, M., Span, R. 2008. An analysis of available mathematical models for anaerobic digestion of organic substances for production of biogas. In International Gas Union Research Conference 2008, Paris, October 8-10, 2008. Association Francaise du Gaz. Curran Associates, Inc. Vol. 2, 1294-1323.

[37] Deublein, D., Steinhauser, A., 2008. Biogas from waste and renewable resources. A John Wiley and Sons, Inc., Publication. ISBN 978-3-527-31841-4.

[38] Fernandez, A., Sanchez, A., Font, X. 2005. Anaerobic Co-Digestion of a Simulated Organic Fraction of Municipal Solid Waste and Fats of Animal and Vegetable Origin. Biochemical Engineering Journal, Vol. 26, pp. 22-28.

[39] Lin, J., Zuo, J., Gan, L., Li, P., Liu, F., Wang, K., Chen, L., Gan, H. 2011. Effects of Mixture Ratio on Anaerobic Co-Digestion with Fruit and Vegetable Waste and Food Waste of China. Journal of Environmental Science, Vol. 23(11), pp. 1403-1408.

[40] Khalid, A., Arshad, M., Anjum, M., Mahmood, T., Dawson, L. 2011. Review. The Anaerobic Digestion of Solid Organic Waste. Waste Management, Vol. 31, pp. 1737-1744.

[41] Chen, Y., Cheng, J.J., Creamer, K.S. 2008. Review. Inhibition of Anaerobic Digestion Process: A Review. Bioresource Technology, Vol. 99, pp. 4044-4064.

[42] Angelidaki, I., Ahring, B.K. 1993. Thermophilic Digestion of Livestock Waste: The Effect of Ammonia. Applied Microbiology and Biotechnology, VOl. 38, pp. 560-564.

[43] Donoso-Bravo, A., Mailier, J., Martin, C., Rodriguez, J., Aceves-Lara, C.A., and Wouwer, A.V. 2011. Model Selection, Identification and Validation in Anaerobic Digestion: A Review. Water Research, vol. 45, pp. 5347-5364.

[44] Nielsen, J. 2001. Microbial Process Kinetics. Basic Biotechnology, 2nd Edition. Edited by Colin Ratledge and Bjorn Kristiansen. Cambridge University Press, p. 127-149, ISBN 0521 779170.

[45] Gavala, H. N., Angelidaki, I., Ahring, B. K., 2003. Kinetics and modeling of anaerobic digestion process. Advances in Biochemical Engineering/Biotechnology. 81, 57-93.

[46] Wardhani, P. K., Watanabe, M. 2014. Numerical Study and Simulation of Anaerobic Digestion of Fruit and Vegetable Waste. Conference Proceedings of Research Conference on Engineering and Technology, Bali, Indonesia, June 28-29, pp. 666-673.

[47] Wardhani, P.K., Watanabe, M. 2016. Numerical Study on Anaerobic Digestion of Fruit and Vegetable Waste: Biogas Generation. AIP Conference Proceeding, 1707, 050017, doi: 10.1063/1.4940849.

[48] Wardhani, P.K., Watanabe, M. 2017. Numerical Study of Anaerobic Digestion Processes and Biogas Generation from Fruit and Vegetable Waste. Universal Journal of Agricultural Research, Vol. 5(3), pp. 197-201, doi: 10.13189/ujar.2017.050303.

[49] Wardhani, P.K., Watanabe, M. 2017. Parameter Estimation in Anaerobic Digestion Processes of Fruit and Vegetable Waste. Proceedings of International Conference on Multidisciplinary Research & Practice (ICMRP) 2017, Osaka, Japan, 9-10 August 2017.

[50] Grau, P., Dohanyos, M., Chudoba, J. 1975. Kinetics of Multicomponent Substrate Removal by Activated Sludge. Water Research, vol. 9, pp. 637-642.

[51] Yu, L., Christian Wensel, P., Ma, J., Chen, S., 2013. Mathematical modeling in anaerobic digestion-AD. Journal of Bioremediation and Biodegradation. ISSN: 2155-6199 JBRBD.

[52] Sosnowski, P., Klepacz-Smolka, A., Kaczorek, K., Ledakowicz, S. 2008. Kinetic Investigations of Methane Co-fermentation of Sewage Sludge and Organic Fraction of Municipal Solid Wastes. Bioresource Technology, Vol. 99, pp. 5731-5737.

[53] Lyberatos, G., Skiadas, I.V. 1999. Modelling of Anaerobic Digestion - A Review. Global Nest: the Int. J. Vol 1(2), pp. 63-76.

[54] Ma, J., Frear, C., Wang, Z., Yu, L., Zhao, Q., Li, X., Chen, S. 2013. A Simple Methodology for Rate-Limiting Step Determination for Anaerobic Digestion of Complex Substrates and Effect of Microbial Community Ratio. Bioresource Technology, Vol. 134, pp. 391-395.

[55] Appels, L., Baeyens, J., Degreve, J., Dewil, R. 2008. Principles and Potential of the Anaerobic Digestion of Waste-Activated Sludge. Progress in Energy and Combustion Science, Vol. 34, pp. 755-781.

[56] Schlegel, H.G. and Fuchs, G. (2007). General Microbiology (Allgemeine Mikrobiologie), Eigth Edition. Georg Thieme Verlag, Stuttgart, p. 625.

[57] Anderson, K., Sallis, P. and Uyanik, S. 2003. Anaerobic treatment processes. In: D. Mara and N. Horan, eds., Handbook of Water and Wastewater Microbiology, chapter 24. Academic Press, London, pp. 391-26.

[58] Stronach, S.M., Rudd, T. and Lester, J.N. 1986. Anaerobic Digestion Processes in Industrial Wastewater Treatment. Springer-Verlag, Berlin, Germany.

[59] Grady, C.P.L.J., Daigger, G.T., Love, N.G. and Filipe, C.D.M. 2011. Biological Wastewater Treatment, Third Edition. CRC Press & IWA Publishing, Boca Raton.

[60] Moletta, R., Verrier, D., Albagnac, G. 1986. Dynamic Modelling of Anaerobic Digestion. Water Research, Vol. 20(4), pp. 427-434.

[61] MATLAB R2015a.