The novel method for parameter estimation for large bio-systems

Paulius Rasiukas

A thesis submitted for a degree of Doctor of Philosophy

Submitted to the School of Engineering, Newcastle university

December 2020

-Page left blank intentionally-

Abstract

In this thesis we propose a new method for the parameter identification of large-scale models. The proposed state substitution method can be applied to parametric, non-parametric or hybrid models, but in this work, we will focus on the parametric models, to show methods capabilities of identifying all parameter values. The method aims to decouple the whole system into separate sub-systems, whose parameters can be identified separately, therefore decomposing the solution space. By decreasing the solution space in this manner, traditional parameter identification techniques can be used to identify the parameters of each sub-model. The solved sub-systems are subsequently combined for re-optimisation using a global solver (in this work global search), which ensures statistical optimality of the parameter values.

The proposed decoupling method uses state substitution approach, i.e.: measured values (which contain process noise) are used to create a spline, which replaces coupled components in each ODE sub-system. This makes it possible to integrate each of the sub-systems separately, because the sub-systems are only dependant on the unknown model parameters. In addition, dividing the problem into smaller sections, reduces computational time significantly compared to current simultaneous solution methods.

The proposed state substitution method is compared with two state-of-art approaches. The derivative method and the integral method. Both state-of-art methods and the proposed state substitution method are used to identify parameters for four different cases studies, where they performance is compared. Cases studies increase in complexity allowing comparison of how each method handles different levels of complexity. First three cases studies use simulated data sets, and fourth one uses real measured data. First case study is an artificial benchmark problem, whereas case studies two, three and four are bio-system models, with increasing complexity.

This thesis also proposes ways of evaluating complexity of the system, so systems complexity can be relatively compared to other systems. This allows to assess each systems' relative complexity, an ensure that correct parameter identification method is chosen for the parameter identification. Complexity evaluation is quantified with three different methods, Principal component analysis visualization, self-organizing map analysis and sorted minimization.

-Page left blank intentionally-

Acknowledgments

I would like to thank both of my supervisors Mark Willis and Chris O’Malley, for helping me through the whole project, especially during write-up period.

Also, would like to thank my wife for supporting me emotionally throughout this project.

-Page left blank intentionally-

Content

Figures

Tables

1. Introduction

To be able to understand behaviour of any system, we perform experiments and record measurements, which then need to be interpreted to create knowledge about the system. Across fields, research has produced and produces knowledge in different types and forms and at varying levels of detail and scale, leaving a segregated and distributed vast of knowledge sources. Condensing knowledge into underlining controlling mechanics allows us to construct model of the systems, which can predict behaviour of these systems. In addition, the evolution in analytic techniques has tremendously increased the number of quantities that can be measured, particularly in the life science. The rise of the “system” research fields, which seek to combine knowledge into an integrated and coherent whole, was a consequent development that could be observed over the last decades, e.g., process system engineering, systems biology, or systems medicine. In all those “systems” fields methods are developed and sought that can help to integrate the different knowledge sources to faithfully describe the system (Wellstead 2008; Wolkenhauer 2014; Zhao 2012). These models consist of states that we can measure and of the parameters that influence this system. Within each individual system only the states vary with time, and all parameters are constant. However, if same model would be applied to similar system, which has same underlining mechanics, but different set-up this will lead to different parameter values, which are unique to the system in question. Development of such parametric models is very time consuming, as it involves performing large number of experiments, and vast amount of data to be condensed into underlining equations of the model. Which is why it is not reasonably possible to create unique model for every system, and generic model are used to predict behaviour of the similar systems. To be able to use generic models, parameters of the system need to be identified, such that it represents the system accurately.

As an alternative to parametric models that are time consuming to make, non-parametric models, which derivate their predictions of the system directly from data. Although this approach is less time consuming and can provide model that are accurate for specific application, they do not create any underlining understanding about how system behaves. It is simply a data driven analysis connecting, input variables to output variables. Also, non - parametric model accuracy is highly dependent on accuracy of the data used to create them, and these models are not transferable between similar systems.

Another alternative is to use a hybrid model, which are combination of parametric model and non-parametric model. This type of models allows us to use parametric parts of the model which we have knowledge off and use non-parametric parts of the model where we do not have enough knowledge about underlining mechanics of the system. Although hybrid model cuts of some of the parameters that need to be identified, the parametric part can be still complex with large search space leading to long computational time to identify the parameters.

This large amount of measured data leads to construction of larger and more accurate models, that become more complex each day. This development cycle is hindered by parameter identification step, as such large and complex models take extremely long time to solve due to sheer size of the search space.

Any model that has a parametric part, will always require a parameter identification step before it can be used, for system prediction purposes. If this parameter identification step if very time consuming it effect the ability to improve the models and use them for practical applications.

To be able to practically use these complex models, we require a method that would, cut on time required to identify the parameter within the system. If a methodology would exist that allowed to decouple the identification problems whenever possible, this would decrease the search space for the identification algorithm. Then parameter identification in large-scale complex systems would become solvable with current parameter identification techniques.

1.1. Aims

Aim of this thesis is to introduce a novel parameter identification method, which would allow to solve complex systems faster than current state of art methods. As this new proposed state substitution method is only suited for solving large and complex systems, it will perform worse than state of art methods when solving simpler systems. Therefore, a secondary aim of this thesis is to construct a way of quantifying the complexity of the system, which would help to decide what kind of parameter identification strategy should be used to solve the system within reasonable time and high accuracy.

1.2. Objectives

- Select state of art methods to be compared with the proposed state substitution method (Chapters 2 and 3)
- Construct complexity analysis (Chapter 4)
- Construct novel parameter identification method (Chapter 5)
- Compare the proposed state substitution method with state of art methods, with multiple cases studies (Chapter 6)
- Define boundaries where the proposed state substitution method performs better and where it does not (Chapter 7)

2. Literature review

This literature review discusses the importance and need for accurate and fast parameter identification techniques. Potential methods are separated into three categories a) gradient-based algorithms b) stochastic algorithms c) other algorithms. All categories are compared, and gradient based algorithms are selected as best option due ease of use, which is are highly important criteria for industry. Selected category of gradient based algorithms is explored in-depth. Gradient based algorithms can be broken down into several steps: a) objective function construction b) optimality conditions c) parameter changes. Each of these steps are discussed focusing the on the most commonly used techniques within each step. Moreover, gradient based algorithms can only optimise towards closest local minimum making them, not useful on their own, for optimisation where cost function have multiple local minima. As cost function defines a value of error between model and the measured data, by following the gradient it is only possible to reach closest solution to starting location, which may not be global solution. To reach the global minimum, amongst the multiple local minima, a global solver should be used instead of a local optimiser. To see which of the global solvers can be used as a state of art method, with local gradient-based solver - five global solvers are explored. The five chosen solvers are all present in the MATLAB optimization toolbox. For each global solver, the working principals are explained, and then they are evaluated using four criteria: a) speed – based on computational time b) complexity – based on how simple it is to use c) accuracy – based of how accurate the final model is d) stability – based on how replicable the results are. This is used to explain why global search optimization will be used for the derivative method and multi-start optimization used for the iterative method in later chapters. Furthermore, the literature review serves as basis, to explain why the derivative and iterative approaches, were chosen as two state of art parameter identification methods to be compared with the proposed new method of parameter estimation.

2.1. Introduction

Parameter identification is a problem where a user tries to find parameters values for provided generic model, to produce a model with accurate prediction capabilities for a specific application. Model is considered accurate, when it matches already measured data, and can make predictions of how system will develop in the future. Process of solving parameter identification problem normally starts with having a generic model consisting of first order ordinary differential equations. This generic model has condensed knowledge of similar system behaviour, that are only differentiated by the parameter values. Each of these equations may have multiple parameters, where specific numerical values would produce best fit to current measurements, enabling a user to use model for future predictions. Only over determined problems can be solved analytically as they provide more equations than unknowns. Unfortunately, normally these problems cannot be solved analytically because number of unknowns is larger than number of equations. Such problems are under determined, making it impossible to find a singular unique parameter values that satisfies the equation. For this reason, parameter identification algorithms have been developed to tackle this kind of problem. It tackles this problem iteratively, making each iteration more accurate than the previous one.

The ability to identify the parameters of models of complex large-scale (bio) chemical systems is critical in order to develop an understanding of a system as well as to use the model as a basis for process control or process optimization. Applications of such knowledge can be found in “Quality by design” approach, which is widely used by FDA (Food and drugs administration). Such design philosophy creates better products but requires firm knowledge about the system. (Riley and Li 2011; Yu and Woodcock 2015) Established methods of parameter identification normally simultaneously solve the entire set of nonlinear ordinary differential equations (ODEs) that describe the system to determine the model parameters. In order to solve them, ODEs are integrated, and the optimisation problem is set based on the model prediction and measured data difference, which then can be minimised. Figure 2.1, shows general structure of how measured data is used to calculate parameter values for specific application, using generic model.

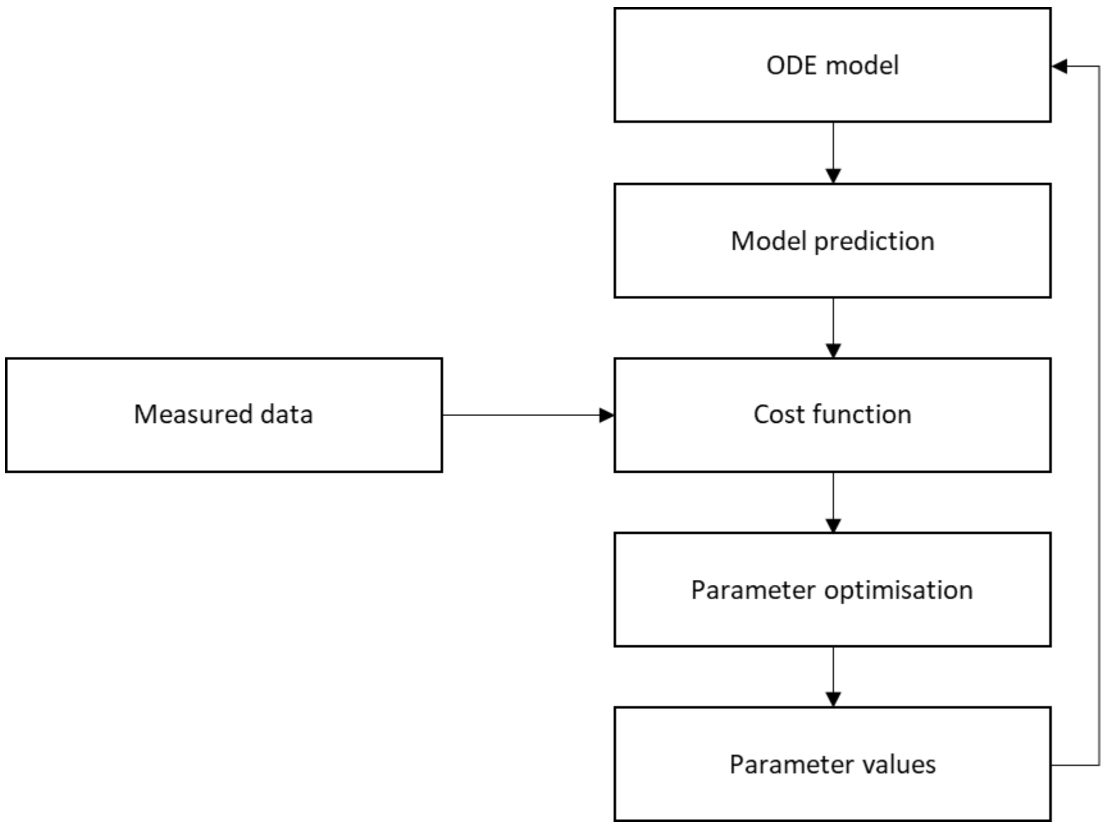


Figure 2.1 General structure of the parameter identification problem structure

Generic optimization methods can be separated into three categories: a) gradient based algorithms b) stochastic search algorithms c) other algorithms (Chou and Voit 2009).

Gradient based algorithms seem like a natural choice as optimisation problem are constructed in terms of cost functions that need to be minimised. This is achieved using gradient based regression and it is included in all major software's. Stochastic search algorithms consist of methods that are used for global optimization. Such algorithms as genetic algorithms, simulated annealing, and clustering methods. These methods are fit for purpose of finding global optimum in highly non-linear systems but require additional computational time to converge to a solution. Other algorithms consist of approaches that try to reduce the parameter search space or reduce risk of method getting stuck in local minima. These methods are hard to implement and are system specific. Such algorithms consist of Alternating regression and geometric programming.

Companies prefer to use easy to apply and fast algorithms for their kinetic analysis of systems (Steijns 1997). This makes gradient based algorithms the preferred options, as stochastic search algorithms, such as genetic algorithms (Y. Maki and Tominaga

2001), are slow and hard to implement (Kikuchi *et al*. 2003), and other methods that don't fit within the first two groups like geometric programming are normally hard to implement (Marin-Sanguino *et al*. 2007). For these reasons gradient based optimizers are included in all major software packages (Chou and Voit 2009).

Most gradient based parameter identification techniques can be separated into integral and derivative approaches. The derivative approach normally proceeds by estimating derivatives to approximate the change per unit time for a set of ODEs (Froment and Bischoff 1990; Holland and Rayford 1989). When approximating derivatives, you develop a set of algebraic equations for each state, which normally can be solved relatively fast because a numerical integrator is not used. However this method often results in sub-optimal parameter values (Willis and Stosch 2016; Yeow *et al*. 2003). This is caused by errors in measurements, which are amplified when estimating derivatives which can lead to inaccurate parameter values. This means that problem must be normalised, making it only suitable for rich data environments.

On the other hand, the integral approach can be used if a system can be written as a scalar differential equation of higher order. When combined with a measured data set an error model can be produced by creating a least squares cost function to estimate the parameters simultaneously (Voss and Timmers 2004). This has been done for several different approaches of the least square's method (Aguirre and Billings 1995; Breeden and Hubler 1990; Cremers and Hubler 1987; Crutchfield and McNamara 1987; Gouesbet 1991; Hegger 1998; Kadtke *et al*. 1993). These attempts achieved varying degree of success, but they all suffer from same problems: a) poor noise robustness – it has been shown that these approaches work well for small noise levels, but lead to inaccurate parameter values when noise levels are increased (Timmer J 2000) b) different size errors in variables – if not taken into account lead to biased and inaccurate parameter values (Voss and Timmers 2004) if non-sensitive variables are part of ODE equation that has larger error impact than the other ODEs within the system, optimizers would try to adjust those parameters in exchange for accuracy of sensitive variables within ODEs with lower noise c) these methods are all parametric and require full and accurate structure of the system.

Although, these techniques can work well, (taking into account points a, b and c mentioned above) they can be relatively slow, because for larger systems the solution space rapidly grows (Bardow and Marquardt 2004).

Both the integral and derivative methods can be improved by using multithreading, adding assessment of noise, using scaling variances. (Raue *et al*. 2013). Multithreading can be beneficial to any iterative algorithm as it can run multiple integration loops in parallel, reducing total computational time. However, this require that software and hardware would be compatible for such use. Assessment of noise can help to improve algorithms accuracy, as noise can be minimized in post-processing of the data. However, this requires having a prior knowledge of noise level, cause, and behaviour. Scaling variances enables algorithm to consider smaller variance of states which do not have high measured values. All these techniques increase the performance of the method but do not alter the method itself. This work will not focus on supportive algorithms that can improve performance of a method, because these supportive algorithms can be applied to most of optimization techniques regardless of a chosen method.

The method proposed in this work will be compared to one existing integral method and one existing derivative method. It is generally agreed that the current best integral method for identifying parameters is multi-start search with Latin hypercube sampling providing the initial conditions (Degasperi *et al*. 2017; Raue *et al*. 2013). This provides a best compromise between accuracy and computational time. As for the derivative method, derivative estimation at each point will be applied and compared with calculated derivatives, creating a network of simple algebraic equations that can be optimised, for the parameter values (D.I. Kamenski and Dimitrov 1993). This method is chosen for it's simple application and fast computational time, as these two criteria are desired by industry (Steijns 1997).

The following literature review will describe different gradient-based optimization algorithms that can be used with the integral and derivative approaches. The literature review will also cover the advantages and disadvantages of the proposed state substitution method in comparison with existing integral and derivative approaches.

2.2. Gradient based optimisation

Optimization algorithms are central to both the integral and derivative approaches, as they are used to calculate the unknown model parameters. Although the derivative and integral approaches use gradient based optimisation to find parameter numeric values, they reach different results. This makes them easily comparable side by side, as all differences in performance must come for their different approach strategies. Gradient based optimizers can have many different forms including but not limited to: a) non-linear regression (Nigel 1995) b) Levenberg-Marquardt algorithm (Simeone 2006) c) Newton flow optimization (Kutalik *et al*. 2007). Each of these variations have the same general structure (See the flowchart figure 2.2).

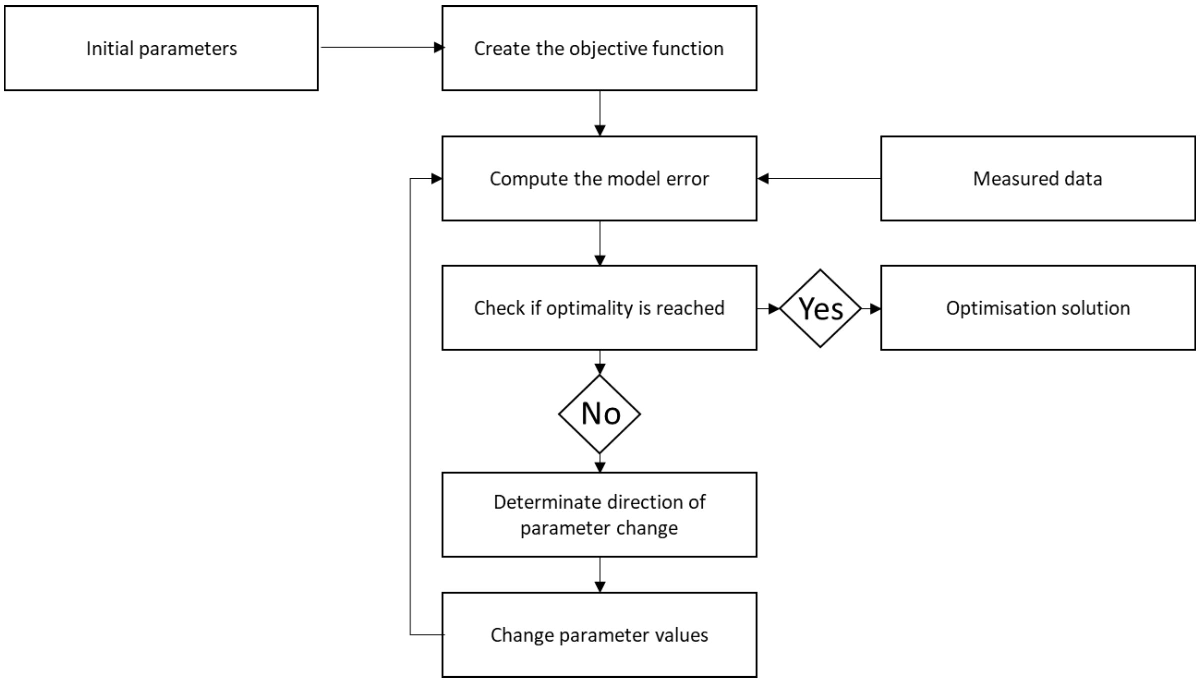


Figure 2.2 Flowchart of gradient based optimisation

This iterative procedure is repeated until the optimality conditions are satisfied and a satisfactory solution is obtained. Which is defined by some convergence criteria or tolerance being specified

2.2.1. Objective function

When constructing the objective function, to represent the error between measured data and model prediction. The most important thing is to be make sure that error provided, represent the model's validity the best. Most commonly this is represented by the sum of Euclidean distances between, measured data and model predicted data. Moreover, usually two different data sets are used for satisfactory identification, because a single data set, can be fitted with arbitrary parameters to produce good fit, without making the model useful for other predictions. This results in two objective functions of each data set, to form the final error value. Individual objective functions can be modified to best represent the system depending of the needs. Three most common techniques are:

a) Weighting: If the system consists of several species or states that are observed such as biomass, substrates, products, etc. That can be described as individual ODE's, the weighting of each ODEs error can improve the objective function representation of model performance. A higher weighting factor should be allocated to states that are less sensitive to parameter changes, this ensures that small changes created by important parameters are more easily observed in overall objective function. Unfortunately, there is no formal procedure to decide what weighting values should be allocated to each state to obtain best results. These values are usually derived from previous knowledge about the system and experimentation with different setups. Equation 2.1 shows example of cost function with weighting incorporated. J is for cost function value, w_i for weighting values for i state, X_{ij} measured value of i state at j time point, $\hat{X}_{ij}$ model prediction value of i state at j time point.

$$J = w_i \sum_{i=1}^{n} \left|(X_{ij} - \hat{X}_{ij})\right| \tag{2.1}$$

b) Normalization: If different state measurements are an order of magnitude different, it is good idea to normalise, all measurement and model predicted data, to avoid disproportional error representation. For example, if biomass concentration measurements are in the range of 0 to 1, and substrate concentration measurements are in the range 0 to 100. Ten percent error in both of these measurements would be maximum of 0.1 and 10. If the objective function is not normalised then the same relative size mismatch in substrate will perceived by algorithm as hundred times worse, then in biomass. To avoid this normalisation is performed by dividing all measurements of each state with the maximum value, of each measurement set. This "rescales" all measurements and predicted data to the scale 0 to 1. Equation 2.2 shows example of cost function with normalization incorporated. J is for cost function value, X_{ij} measured value of i state at j time point, $\hat{X}_{ij}$ model prediction value of i state at j time point.

$$J = \sum_{i=1}^{n} \left| \left(\frac{X_{ij} - \hat{X}_{ij}}{\max(X_{ij})} \right) \right| \qquad 2.2$$

c) Squaring: As a model of a system is normally not perfect, one hundred percent accurate predictions are not possible, nevertheless for practical applications one hundred percent accuracy is not needed and small deviations are acceptable, if general trend is accurate. To promote algorithm to avoid large error and ignore small ones, squaring of error may be performed. This has an effect where a large error between the measurement and model predictions, becomes even bigger and very small errors gets even smaller. Equation 2.3 shows example of cost function with squaring incorporated. J is for cost function value, X_{ij} measured value of i state at j time point, $\hat{X}_{ij}$ model prediction value of i state at j time point.

$$J = \sum_{i=1}^{n} (X_{ij} - \hat{X}_{ij})^2 \qquad 2.3$$

2.2.2. Optimality of numerical optimisation algorithms

Optimality refers to conditions chosen, that determinate if the current solution is satisfactory or if optimization should continue. Optimality can consist of single or multiple conditions. With multiple conditions, it usually enough to satisfy one of them. Optimality conditions should be chosen to be strict enough to produce accurate parameter values, but flexible enough to make it possible to reach the solution. The best optimality conditions are very system specific and could vary a lot from application to application. Four most common optimality conditions are:

a) Tolerance: Is a value, that is compared with objective function after each iteration. If objective function is lower than, the tolerance value optimality is reached. Increasing tolerance, makes optimization faster and easier in exchange for lower accuracy of the model prediction. Equation 2.4 example of tolerance optimality condition, where J is cost function value and ε is value of tolerance.

$$|J| \leq \varepsilon \qquad 2.4$$

b) Step tolerance: Is a value, that is compared with a difference between the current iteration objective function value and the previous one after each iteration. If the difference between the current and previous iterations objective function values is lower than the step tolerance value optimality is reached. Step tolerance allows the optimization algorithm to reach its local minimum without knowing the value of objective function at the minimum. This is most commonly used optimality condition of numerical optimization algorithms. Equation 2.5 example of step tolerance optimality condition, where J is cost function value, n current iteration number and ε is value of step tolerance.

$$|J_{n-1} - J_n| \leq \varepsilon \qquad 2.5$$

c) The number of iterations: Is an optimality condition that stops, the optimization algorithm after certain number of iterations. This optimality condition is usually set to high number and is a safeguard, that prevents algorithm from getting stuck in infinite loop. Equation 2.6 example of the number of iteration optimality condition, where n is current number of iterations, and ε is value of the number of iterations.

$$n = \varepsilon \qquad 2.6$$

d) The time of iteration: Is an optimality condition that stops, optimization after certain time has passed. Similarity to the number of iterations optimality condition, it is used most of the time as safeguard against infinite loops. It is also useful for optimization with very wide search spaces, to ensure algorithm does not spend too much time optimizing each set. Equation 2.7 example of the time of iteration optimality condition, where nt is current length of the iteration, and ε is value of the time of iteration.

$$nt = \varepsilon \qquad 2.7$$

2.2.3. Parameter change

The core of gradient based optimization algorithms is how the algorithm within each iteration decides to adjust the parameter values. This step determines how fast and accurate the overall optimization algorithm will be. Parameter values are changed based on the gradient of cost function with respect to each parameter. Within the system as it represents if parameter value gets closer or further from a local minimum. This can be easily visualized with a simple parabolic curve with a single parameter (fig 2.3). If we follow parameter value from -100 to 0, we can see that function value decreases, and the gradient value increases. Negative gradient means we are approaching local optimum and positive gradient means we are moving away from the local optimum; zero gradient is at a local optimum.

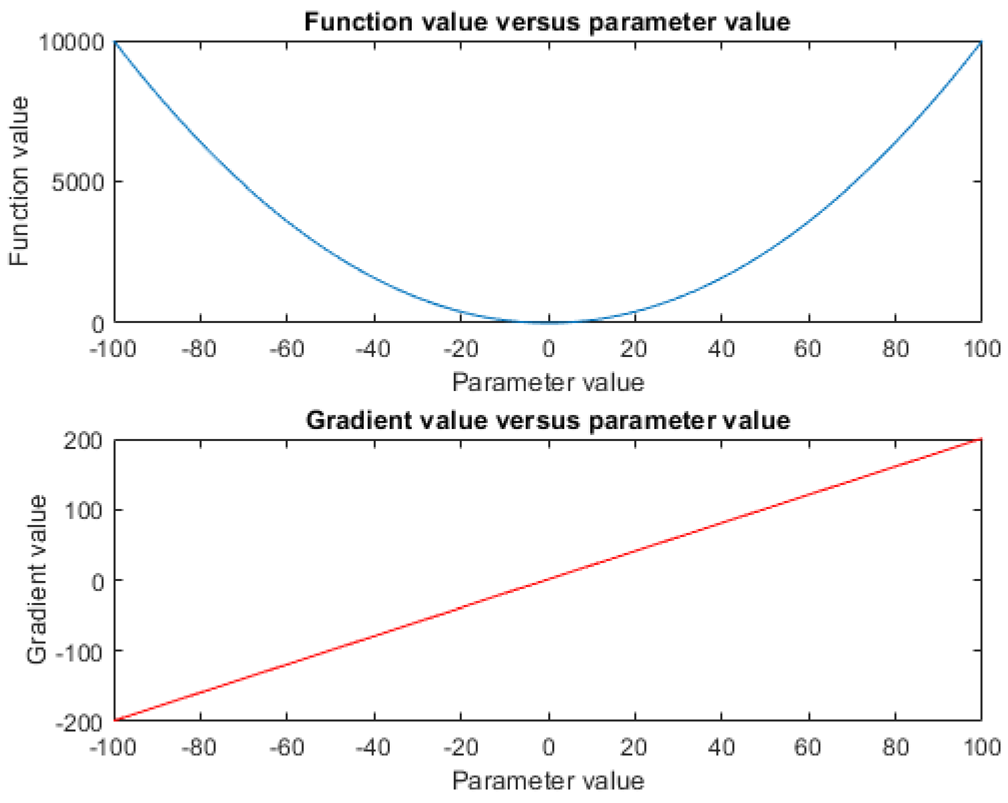


Figure 2.3 Objective function with single parameter value and gradient value versus parameter

This simple visualization is improved upon in the actual implementation of gradient based optimization algorithms in order to deal with multiple parameters and multiple functions. The two most common algorithms are the Gauss-Newton algorithm and the Levenberg-Marquardt algorithm. Both methods can be used to solve non-linear optimization problems which is most common type of optimization problems.

The Gauss-Newton algorithm (GNA) is derivation of Newton–Raphson method, but has the advantage of not needing second order derivatives of the system, which can be hard to obtain (Mittelhammer 2000). The Gauss-Newton method can minimize the summed squared error of multiple variables and functions at the same time. Starting with the provided initial conditions x(0), the algorithm creates next iteration parameter set by applying (Equation 2.8), Jr within the equation represents the first order partial derivatives and is also known as the Jacobian matrix. If y (function) and x (parameters) are column vectors the Jacobian matrix can be written as (Equation 2.9). In case number of evaluated functions and number of estimated parameters are equal Gauss-Newton method can be simplified (Equation 2.10). Although method works well with one unique solution it can become unstable, while trying to optimize system with multiple solutions, or multiple local minimums (Mascarenhas 2013). For this reason, Gauss-Newton algorithm will not be used in this work.

$$x(t+1) = x(t) - (Jr^T Jr)^{-1} Jr^T y(x(t)) \tag{2.8}$$

$$(Jr)_{ij} = \frac{\partial y_i(x(t))}{\partial x_j} \tag{2.9}$$

$$x(t+1) = x(t) - Jr^{-1} y(x(t)) \tag{2.10}$$

The Levenber-Marquardt algorithm(LMA) was first published in 1944 (Levenberg 1944). The LMA is an improvement to the GNA, by increasing robustness, but slightly increasing computational time. The LMA modifies (Equation 2.8), by introducing a damping factor λ, and rearranging the equation to calculate a change in parameter values instead of the new parameter values (Equation 2.11). The damping factor is adjusted each iteration, based on the change in objective function value. If the change in the objective function is sufficient the damping factor is reduced, which brings it closer to the GNA, but if the change in objective function is not sufficient the damping factor is increased bringing it closer to pure gradient descent.

$$x(t+1) - x(t) = -(Jr^T Jr + \lambda I)^{-1} Jr^T y(x(t)) \qquad 2.11$$

The strategy of adjusting the damping factor can be different, which will provide different speeds of optimization and accuracy. A common way of adjusting the damping factor within the LMA optimization is to set an initial damping factor value (λ_0) and velocity value (v). The velocity value must be higher than 1 (v > 1). After each iteration multiple new damping factors are calculated (table 2.1). The objective function is computed with each damping factor and the damping factor that produces the lowest function value, is set as the new λ. It should be noted that velocity value (v) stays the same with each iteration. The absolute values of v and λ are based on the scale of the objective function.

Possible damping factors		
$\lambda = \lambda_0$	$\lambda = \lambda_0/v$	$\lambda = \lambda_0 v^j$, j = 1,2,3...

Table 2-1 Damping factors to be evaluated after each iteration

As mentioned before, this type of LMA will lead to more robust solutions in exchange for a slight increase to computational time, when compared with GNA. To improve LMA computational time Geodesic acceleration can be used. Instead of only adjusting parameters, based on the first order derivatives, second order derivatives can be incorporated to adjust the parameter change at each iteration. It only requires the single directional second derivative, which does not add a large amount of computational time, but improves convergence significantly (Mark K 2012). The parameter change is then defined by velocity (first order) and acceleration (second order) (Equation 2.12). The first order change is estimated as explained above while the second order directional derivative (Equation 2.13) can be estimated using finite difference approximation. By using finite difference approximation only one additional function evaluation f(x+hδ) needs to be done, as the Jr matrix and f(x) is already computed in the previous step (Equation 2.14). This requires selecting an arbitrary step (h) value. It was reported that a value of 0.1 works well for most cases (Mark K 2012). Lastly, to accept this modified parameter change it must satisfy condition (Equation 2.15), if not it is rejected and the unmodified change of velocity (Δx_1) is selected. The value of α can be selected to be anything below 1, but it is suggested to use 0.75 for most cases and 0.1 for difficult problems (Mark K 2012).

$$\Delta x = \Delta x_1 + \Delta x_2 = v\delta t + \frac{1}{2}a\delta t^2 \qquad 2.12$$

$$\Delta x_2 = -\frac{1}{2}(Jr^T Jr + \lambda I)^{-1} Jr^T y\big(x(t)\big)^n \qquad 2.13$$

$$y\big(x(t)\big)^n \approx \frac{2}{h}\left(\frac{\mathrm{f(x+h\delta)-f(x)}}{h} - J\Delta x_1\right) \qquad 2.14$$

$$\frac{2|\Delta x_2|}{|\Delta x_1|} \leq \alpha \qquad 2.15$$

Combined with improvements, LMA outperforms GNA in every aspect, which is why LMA was chosen as the algorithm for gradient based optimization in this work.

2.3. Global solver

Gradient based optimization on its own can only find local minimum. If a problem has multiple local minimums, only the closest local minimum to the initial conditions will be found. To work around this issue a global solver strategy needs to be used. There are many global solvers, but we will focus only on the ones MATLAB provides, as it is the software used to develop the proposed new method and compare it to the state-of-the-art methods. MATLAB offers five different global solvers (MATLAB documentation) a) Pattern search (Charles Audet; J. E. Dennis 2002) b) Particle swarm (James Kenedy 1995) c) Genetic algorithm (Goldberg 1989) d) Surrogate optimization (Gutmann 2001) e) Global search (Ugray 2007).

a) The pattern search algorithm works, by creating several separate groups of potential solution around the initial point and evaluating the objective function at each of these points. These groups are called pooling. For example, if the objective function has two parameters, the pattern search algorithm would create four new parameter sets and evaluate the change of the objective function with parameters in all four cardinal directions. The size of change for these parameters is called a mesh, which is doubled on successful pooling and halved on unsuccessful pooling by default. Pooling is considered to be successful when at least one of the newly defined parameter sets produces a lower objective function value, than the previous iteration. On successful pooling the parameter set with lowest objective function value is considered to be the new initial point for the next iteration, in case of unsuccessful pooling the initial point is not changed, only the mesh size is adjusted. Pattern search is robust to discontinuities in the objective function, as such solutions would be simply ignored. However, this approach does not guarantee that a global optimum will be found, but may avoid some local minimums, in which simple gradient-based approach would get stuck to. This method is usually fast, but not extremely accurate. In one of the examples when it is compared to genetic algorithm (Michael Wetter 2003), it is showed that two out of three cases accuracy of the method was lower than GA. It is stated that although pattern search is global optimiser it can be attracted to a local minimum, which make it difficult to bypass discontinuities or local minima, that are between starting position and global optimum.

b) The particle swarm algorithm works by searching a bounded parameter space without needing to have an initial starting point. The algorithm initializes an array of random parameters values known as 'particles' within a bounded region. For each iteration the particle swarm algorithm, generates a random velocity vector for each particle, then moves a particle to a new location based on its own unique velocity vector and estimates the objective function value at each of these locations. The position with the lowest objective function is considered to be the new position of that particle. After a new position is found a particles velocity vector is adjusted based on certain criteria such as: previous velocity, distance between previous and new location and distance between other particles. This process is repeated for each particle. Particle swarm has a high success rate of finding global minimum, as the number of evaluated functions is significantly higher than e.g., pattern search. This comes at a cost of increased computational time. Particle swarm is a stochastic algorithm, which means it will not yield same results for every optimisation run. Particle swarm can be applied as parameter identification algorithm, but it is not popular option. Only 2.8% of published paper until 2007 about particle swarm, were used for modelling applications. (Poli *et al*. 2007). Although its popularity rise it does not seem as go-to robust method for parameter optimisation currently.

c) A genetic algorithm is another stochastic method that is based on real life evolution. The method is initialized by a provided random or pre-selected population of samples. Each of the samples are evaluated using the objective function and are ranked based on their score. Lower objective function values provide higher score. A certain percentage of population with the best scores are selected to create the next iteration population - to be 'parents'. From the parents next iteration population is most commonly created in three ways. First a very small number of 'elite' samples are selected to be part of next iteration population. ‘Elite’ samples are the ones with best score values. A common value for 'elite' is 1% of the selected 'parent' samples. Second the rest of the 'parent' population is randomly selected for one of the two method, which are mutation and crossover. During mutation, a single random parent is chosen, and its parameter values are randomly adjusted, this creates a 'child' for next iteration population. During crossover two random parents are selected and by combining their parameter values a 'child' for next iteration is made. Crossover and mutation are repeated till a new population reaches the size of the initial population. Once a new population is ready, the same process is repeated. A genetic algorithm has a near 100% probability to reach global minimum if given enough time, however the sheer number of evaluations and iterations needed to reach good solution increases computational time exponentially. Genetic algorithms can be successfully applied to variety of problems from energy usage optimisation (Leonori *et al*. 2020; Salata *et al*. 2020), to optimising PID controllers (Abadlia *et al*. 2020). Each of these cases does not go into detail of computational demand for these optimisations, as this is not the focus of the study, but it is safe to assume computational time is high, due to how genetic algorithm are executed. Furthermore, application for genetic algorithm typically are the ones that value high accuracy of optimisation and does not have time constraints.

d) Surrogate optimization operates with two main steps, construction of a surrogate model and search for the minimum. During construction of the surrogate model quasi-random parameter sets are created. If any initial parameters are defined, they are used together with quasi-random parameter sets. Each of the sets are evaluated using an objective function to provide the value to each parameter set. After evaluation the objective function values of each set are interpolated using (a / the) radial basis function (Powell 1992) to create surrogate model of the objective function. Then the incumbent point is found – which is the lowest objective function value from given parameter sets. Once the surrogate model is constructed and the incumbent point found the algorithm moves to search for a minimum. Search starts around the incumbent point, by using a set 'scale' value and the algorithm determines the range of search around the incumbent point. Within this range of search thousands of pseudo-random vectors are created and applied to the incumbent point to find an array of new sample points. Each sample points are evaluated with the merit function, which is a combination of surrogate values and distance values (Equation 2.16-18). Increasing weight (w) within the merit function increases the rewards for the algorithm to search close to incumbent point, where lowering weight (w) value encourages algorithm to search further from incumbent point revealing new regions. The parameter set with lowest merit function value is evaluated with the objective function and the surrogate model is updated by adding newly acquired value. If addition of a new value, changes the incumbent point, search is considered successful. After multiple consecutive successful searches, the scale of search is increased, similarly after multiple consecutive unsuccessful searches the scale of search is decreased. This ends a single iteration. By default, the stopping criteria for surrogate optimization is the number of iterations which must be defined, prior to start of optimization. This might lead to the algorithm not reaching the global optimum due to insufficient number of iterations, or running much longer than needed, after the global optimum is found.

$$f_{merit}(x) = wS(x) + (1-w)D(x) \tag{2.16}$$

$$D(x) = \frac{d_{max} - d(x)}{d_{max} - d_{min}} \tag{2.17}$$

$$S(x) = \frac{s(x) - s_{min}}{s_{max} - s_{min}} \tag{2.18}$$

d(x) is distance between sampled points and s(x) is surrogate function value of samples points. Surrogate optimisation while useful according to (Lu *et al*. 2020), can struggle to optimise parameters when possible parameter boundaries are too large.

e) Global Search or multi start algorithms work similarly but have some differences. Key differences are that global search rejects start points that are unlikely to improve, current best local minimum, whereas the multi start optimizes all start points. Multi start allows multiple local solvers, whereas global search does not. Furthermore, multi start can run in parallel. Global search starts with taking the provided initial conditions and a running local optimization solver and if the initial start point cannot converge to a solution the algorithm cannot continue. Once convergence is reached the algorithm records, the start position, end position and radius of estimated basin between those two positions. Furthermore, the final objective function value is recorded to be used in a score function later on. The score function consists of the objective function and a multiple of the sum of the constraint violations, this way viable points score function equal to their objective function value. After initial local optimization the algorithm generates a trial point by using non symmetrical scatter (Glover 1998). These trial points are potential starting points for the next step. To obtain a stage 1 start points, a fraction of trial points are selected to be evaluated for their score function. The trial point with best score function is selected to be locally optimized, and similar to initial start values, its start position, end position and radius of estimated basin between those two positions is recorded. After stage 1 start point evaluation, the algorithm moves on to stage 2 evaluation, where the rest of trial points, selected previously, are locally optimised if they fit the following criteria: a) the selected position is not in any existing basin b) the position score value is below a local solver threshold. If the trial point satisfies these conditions, it is locally optimised, and the newly created basin is added to the list together with adjusted threshold value. This process is repeated till no trial points are left. All solutions create a global solution vector, where all samples are ranked based on their objective function value and lowest value is chosen as final solution.

In comparison multi start approach is simpler in comparison. At the start of the algorithm, multi start generates randomly equally distributed parameter sets within specified boundaries. The number of generated start points has to be manually selected, and the algorithm use the provided start points first and will generate any additional sets if needed. After having required number of starting points each point is locally optimized and solutions are stored into a global solution vector, which is ranked in descending order based on the objective function. The best solution is selected which managed to achieve lowest objective function. Both of these methods are commonly used in global optimization as they are computationally efficient to reach desirable solution.

Each of these solvers can be used for global optimization, but all of them have their own advantages and disadvantages. There are four main criteria we can look at when selecting a global solver: a) speed b) complexity c) accuracy d) stability. Speed of the solver is important as fast solvers can be adjusted multiple times, allowing the user to experiment with parameter conditions, without losing valuable time. From the previously discussed five global solvers the fastest one is pattern search, and slowest is genetic algorithm (documentation;). Complexity of a solver can be important criteria if the algorithm needs to be applied to a large number of different problems and set up time and effort becomes significant. Setup complexity of all five global solvers is generally similar, with the exception of particle swarm as it does not need initial conditions and can be started with only the objective function. Accuracy is a criterion that is important, when results of optimization need to be trustworthy. If an optimization problem is only required to be solved for general knowledge and an exact result is not important some of the accuracy can be exchanged to improve the other criteria. Based on MATLAB benchmarks most accurate MATLAB global solver is a genetic algorithm followed by global search and surrogate optimization (documentation;). The stability of a solver is important for each optimization to be consistent and being robust to stiff problems and problems with discontinuities. Purely stochastic algorithms like particle swarm and genetic algorithm very rarely yield similar results between repeats, and an algorithm like pattern search deal with discontinuities easily but will navigate away from solution that yield incomplete solution as an answer.

As mentioned before industrial companies prefer fast algorithms that are easy to implement (Steijns 1997), but we cannot neglect accuracy. Between methods that are able to delivery robust and high accuracy results, we choose the one that have lowest computational time. Computational time is important as it creates time saving, that can be used performing experiments or improving model structure itself. This still leaves us with at least two options that seem to fit the description, which are global search and surrogate optimization. Both of these methods could be used, as a state of art method for global optimization and would make good benchmark comparison to the proposed state substitution method. However global search was selected to be the global solver, because of the reports in literature of various multi start being one of the best currently optimization techniques (Degasperi *et al*. 2017; Raue *et al*. 2013). Multi start and global search are very similar, but have their differences as discussed before. For this reason, both of these approaches will be used for comparison, global search with derivative approach and multi-start with integral approach.

2.4. Summary

The literature review chapter looks at why parameter identification problem is important and how it can be solved focusing on gradient based optimisation algorithms. Literature reviews identify the need for faster parameter identification techniques for large systems. This is primary gap that the proposed state substitution method for parameter identification is trying to fill. Flowchart (fig 2.2) of gradient based optimisation algorithm is defined, and key features are investigated in detail, such as the objective function construction, the optimality conditions and the parameter change algorithms. It is noted that local optimisation is not sufficient to obtain accurate parameter values in complex systems. Therefore, five global solvers are investigated and compared, based on four criteria speed, complexity, accuracy, and stability. From information about local and global solver, two variations of gradient based optimisation are chosen to serve as state of art methods, for comparison with the new proposed state substitution method for parameter identification.

3. Kinetic model calibration: State of the art methods

This chapter will discuss, two state of the art methods that were chosen as a basis for comparison with, the proposed state substitution method. The literature review helped to establish that the integral and the derivative approaches are best choices. This chapter will discuss the details of these two approaches. To make sure that both approaches are best suited for the case studies that were selected, modifications are made. Modifications, such as different sampling algorithms, screening or re-optimization are required to cut down computational time, that does not contribute towards more accurate identification of parameter values. This way the state of art methods should perform at their best when compared to the proposed state substitution method.

3.1. Integral approach

The integral method is a direct method that requires an algorithm to integrate the whole model at each integration time step and to evaluate the error between model and measured data. This leads to a slow working method which, spends up to 95% of its time integrating model, and not optimizing the parameters. Furthermore if the model equations are stiff, this increases computational time almost by 100% (Voit and Almeida 2004). Nevertheless, the integral approach predicted model output is very accurate as numerically integrating the model allows the optimisation algorithm to capture all interactions between parameters and states of the model. In addition, the integral method is robust to noisy data, and is easy to implement. The main drawback of integral method is the long computational time, with most of that time being spent integrating the model ODEs, it is important to focus on decreasing the amount of time needed to integrate the model. To have a good state of art integral approach, for accurate comparison, modifications were made to improve the integral approach computational time without sacrificing the accuracy of the method. Multi-start was chosen as the best global optimizer for this approach, as it only locally optimizes given starting locations, and does not search for new start points. This is important as new start points would increase the number of integrations required, thus increasing computational time.

3.2. Derivative approach

The derivative approach tries to deal with the main drawback of integral approach the long computational time associated with model integration. This is done by removing the integration step completely and replacing it by set of algebraic equations that can be optimized simultaneously or separately. In order to do this, slope estimation is required, that would replace the derivative term in each ODE making it a single point algebraic equation. Slope estimation can be done in various ways including: a) linear interpolation b) splines (P.J. Green and Silverman 1994) c) Three point method (R.L. Burden and Faires 1993) d) Hand fitting (Voit 1982) e) Artificial neural networks (ANNs) (Almeida 2002; J.S. Almeida 2003) f) Filters (Eilers 2003; M. Vilela 2007; Whittaker 1923).

Slope estimation methods a, c, and d can work for low or noise free data, as noise tends to get amplified when estimating slopes. As noise is inevitable in any real system this leaves b, e, and f methods, which can deal with noise to certain levels by performing smoothing of the data. ANNs are accurate at finding general trends and fitting data with 'universal functions', but they require a lot of data sets and computational time to be trained. This adds additional complexity to the setup of the method and computational time, which we try to reduce. Between splines and various filters, splines where chosen as a better candidate for the derivative method due to simpler use. Splining is done by fitting data with a polynomial (equation 3.1), or other types of splines, to minimize the error between the spline and measured data. With a spline state values can be approximated at any time point. Which allows to estimate slopes (equation 3.2) at any time point to be replaced in the ODE to change it into group of algebraic equations (equation 3.3). With slopes estimated the algebraic equation can be optimised locally for each data point with non-linear least squares method, creating vector of solutions x_t, as noise can affect these solutions an average of all solutions is taken to represent starting point for global optimisation.

$$C(x) = \alpha_1 + \alpha_2 x + \alpha_3 x^2 + \cdots + \alpha_n x^{n-1} \qquad 3.1$$

$$\frac{C(x_t) - C(x_{t+1})}{t_n - t_{n+1}} = \frac{\widehat{dC}}{dt} \approx \frac{dC}{dt}, at\ t_n \qquad 3.2$$

$$\frac{dC}{dt} = A * \mathrm{x} \approx \frac{\widehat{dC}}{dt} = A * x_t, where\ x_t\ are\ parameter\ values\ at\ \mathrm{t}\ time\ point \qquad 3.3$$

Where C(x) is polynomial of variables x, a are weights, and t is time.

3.3. Integral method modifications

Two modifications were done to general integral multi-start approach to improve its accuracy and computational time. Sampling of the search space was changed to Latin-hyper cube sampling as it provides benefit of each sample on average being closer to the global solution. Screening was introduced to reduce the number of starting samples that need to be optimised. This is important for complex systems especially as they introduce discontinuities which cannot be optimised and there is no need to waste computational time optimising them.

3.3.1. Sampling of the search space

In an attempt to ensure that global minimum is found using multi-start approach, sampling of the search space for starting points, has to be done methodically, while trying to minimize the number of starting points, but covering the whole search space. The simplest solution for sampling the search space would be a factorial design, which is commonly used for experimental setups, to capture the effects of each parameter (Abdel Moamen *et al*. 2015; Biró *et al*. 2009). Unfortunately, this expands the number of samples, exponentially with increasing number of parameters. Furthermore, unlike experiments, model parameters can be very sensitive even to minor changes requiring a lot of steps for each parameter. A better sampling technique for this application is Latin-hyper cube sampling (McKay *et al*. 1979). The search space is split up into equal size pockets, and each sample point is randomly put into one of these pockets, while having straight line of sight with each coordinate end, without interfering with other samples points line of sight (fig 3.1). This decreases the number of samples required, to capture dynamics of the model, as each new sample provides unique setup, without overlapping with other samples. The Latin-hypercube sampling requires predetermined number of samples, which has to be manually selected. The number of initial samples should increase with larger and more complex models and decrease with simpler and smaller models.

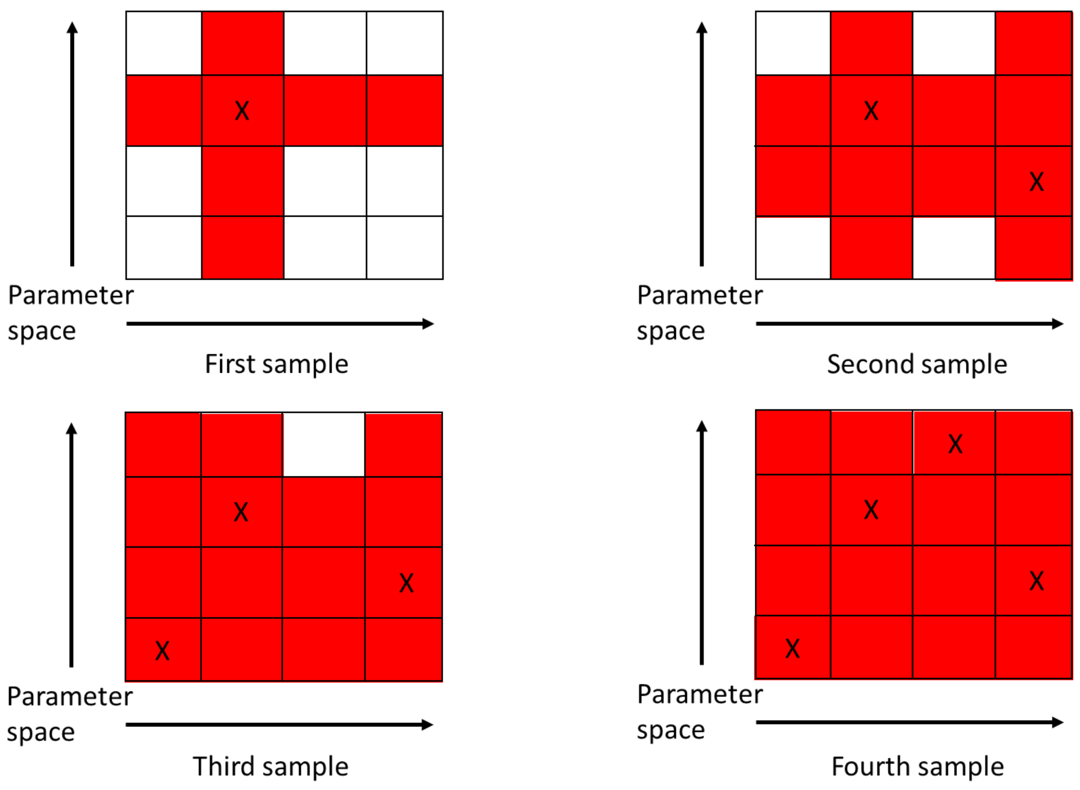


Figure 3.1 Example of Latin hyper cube sampling in 2d space, with X denoting each sample and red square places that are unavailable to following samples.

3.3.2. Screening

The second modification to improve integral method, is screening of initial start points. This is required because, we do not know if we have selected too many samples and some of them can be redundant and for complex systems a lot of starting points extremely far away from global optimum or cannot be integrated. Screening is performed for all samples evaluating their objective function, then 10% of best starting points are given to multi-start algorithm to optimize further. This screening process cuts off a large amount of unnecessary time, without compromising accuracy of the overall method. This type of screening works, because as parameter values get closer to global optimum, error between prediction and the model decreases. This makes it safe to assume that values that have initial low error values before optimization, will have even lower final error value.

3.3.3. Modified integral method structure

When these modifications are implemented the incremental method approach flow chart changes (fig 3.2). First the search space is sampled using Latin-hyper cube sampling method for specific number of samples, second all samples are screened for their objective function value and the best 10% passes to next stage, lastly multi-start global optimiser locally optimises all provided starting samples and global solution is reached. This modified version of integral approach should be most competitive versus the proposed state substitution method.

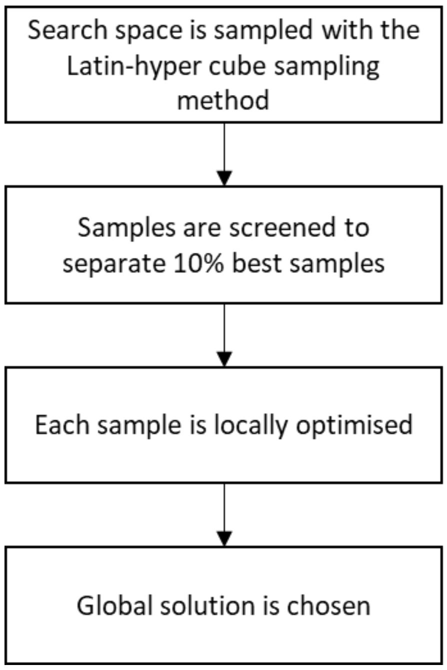

Search space is sampled with the Latin-hyper cube sampling method
Samples are screened to separate 10% best samples
Each sample is locally optimised
Global solution is chosen

Figure 3.2 The integral method flowchart

3.4. Derivative estimation method modifications

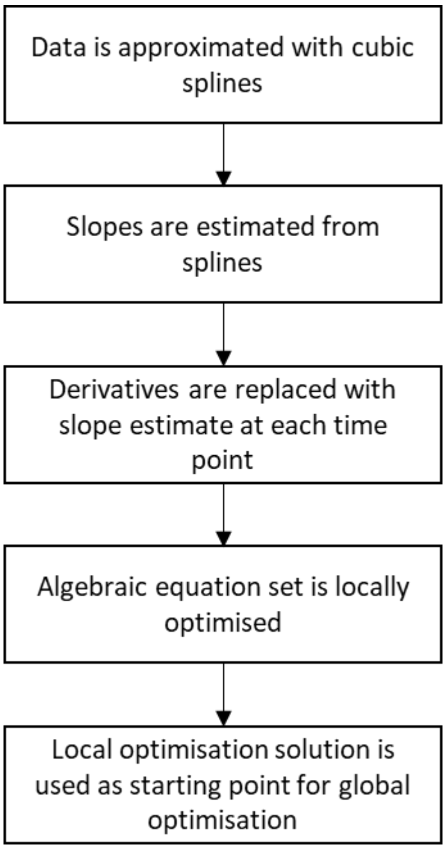
Only one modification was added to the derivative estimation method. Re-optimization step is required to convert derivative estimation from local only into global optimisation technique.

3.4.1. Re-optimization

Global optimisation is performed with global search, as it is similar to multi-start but only requires single starting point input. This step is called re-optimization to ensure that solution is globally optimum, as algebraic equation were only locally optimised. Global search method is described in section 2.3.

3.4.2. Modified derivative estimation method structure

When re-optimization is implemented into the derivative method it structure becomes as follows (fig 3.3). First measured data is splined and approximated with cubic spline. Then slopes are estimated using splines and used to replace derivates in ODEs of the model, to construct set of algebraic equations that are locally optimised at every data point. Solution vector is averaged and supplied as starting point for global search algorithm for re-optimization step. The final solution is produced by global search algorithm.



Data is approximated with cubic splines
Slopes are estimated from splines
Derivatives are replaced with slope estimate at each time point
Algebraic equation set is locally optimised
Local optimisation solution is used as starting point for global optimisation

Figure 3.3 The derivative method flowchart

3.5. Benchmark problem

To study the effects of modifications, within each method, a benchmark problem was optimised, with and without these modifications and compared to assess the impact of each modification. The chosen benchmark problem is a simplified version of Monod kinetics (equation 3.4). This system consists of two states biomass (x, g/L) and substrate (S, g/L). This model also has three parameters U_{max}, K_s and q, which are the maximum specific growth rate (h^{-1}), half-velocity constant (g/L) and the substrate consumption rate constant (g/h) respectively. Models' outputs are rate of change in biomass $\frac{dx}{dt}$ and substrate $\frac{dS}{dt}$.

$$\frac{dx}{dt} = u * x$$
$$\frac{dS}{dt} = -q * u * x \qquad 3.4$$
$$u = U_{max} * \frac{S}{K_s + S}$$

3.5.1. Generating benchmark data

To simulate data sets using this model, parameters values need to set to desired constants and the model needs to be initialised with initial conditions for biomass (x) and substrate (S). Data sets were generated using numerical integrator ode45 in-build into MATLAB. Output of each ODE is added to existing value of the state producing time-series data of biomass (x) and substrate (S). Data was generated at 10h^{-1} sample rate, and white random noise with 10% magnitude was added after, data generation, but before it being used in optimisation algorithm. The parameter values that were used to generate data are U_{max} = 0.9, K_S = 0.3 and q = 4. These are considered true values of the parameters and the parameter identification algorithm accuracy can be evaluated based on how close identified parameter values are to real values. Typical data set have low starting biomass concentration and high substrate concentration. As time progress substrate is consumed exponentially and biomass growths exponentially (fig 3.4).

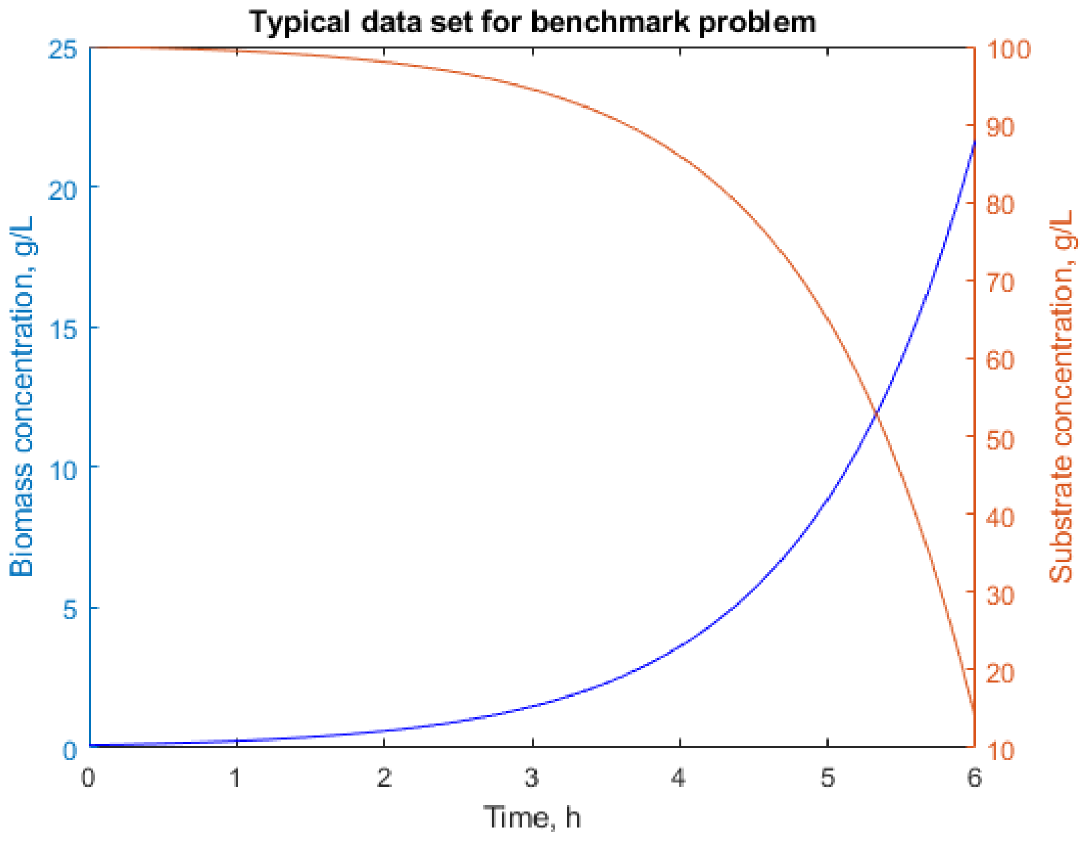


Figure 3.4 Typical data set of benchmark problem, without the noise

3.5.2. Objective function

Objective function for this benchmark model was constructed as a sum of two different data sets errors. Where error was defined as squared difference between data sets of generated noisy data, and model predictions (equation 3.5). X and Y and are generated data sets, $\hat{X}$and $\hat{Y}$ are model predictions.

$$J = \left(X_i - \hat{X}_i\right)^2 + \left(Y_i - \hat{Y}_i\right)^2 \qquad 3.5$$

3.6. Effects of the modifications

Each of the modifications for the state of art methods were done with intention to increase accuracy or decrease computational time. This sub-section describes with example how these modification effect state of art methods.

3.6.1. Sampling space

To test how much difference there are between the Latin hyper cube sampling technique and factorial design both sampling techniques were compared while using integral method. When exposed to the benchmarking problem, both methods were able to identify a solution, but Latin hyper cube method was faster and produced more accurate model (table 3.1). Benchmarking problem was convex and global solution can be achieved from many different starting points. However, Latin-hyper cube sampling provided with better overall parameter values (table 3.2). Also, the run time is significantly different, the Latin hyper cube sampling method outperforms factorial design by 15% in terms of computational time. As both methods had same number of samples, faster computational time is achieved by having initial parameter values (samples) on average closer to the global solution. This leads to optimiser having to do less iterations to reach a solution. Visual comparison of the model, with each identified sets of parameters is shown in figure 3.5.

Sampling method	**Squared error of the model**	**Run time, s**
Latin hyper cube	425.29	728.47
Factorial design	1439.79	862.33

Table 3-1 Performance comparison of different sampling methods with benchmarking problem

Parameter	**Real values**	**Latin-hyper cube sampling**	**Factorial design sampling**
U_{max}	0.9	0.898	0.875
K_s	0.3	0.083	0.248
q	4	3.987	3.992

Table 3-2 Identified parameter values using different sampling techniques.

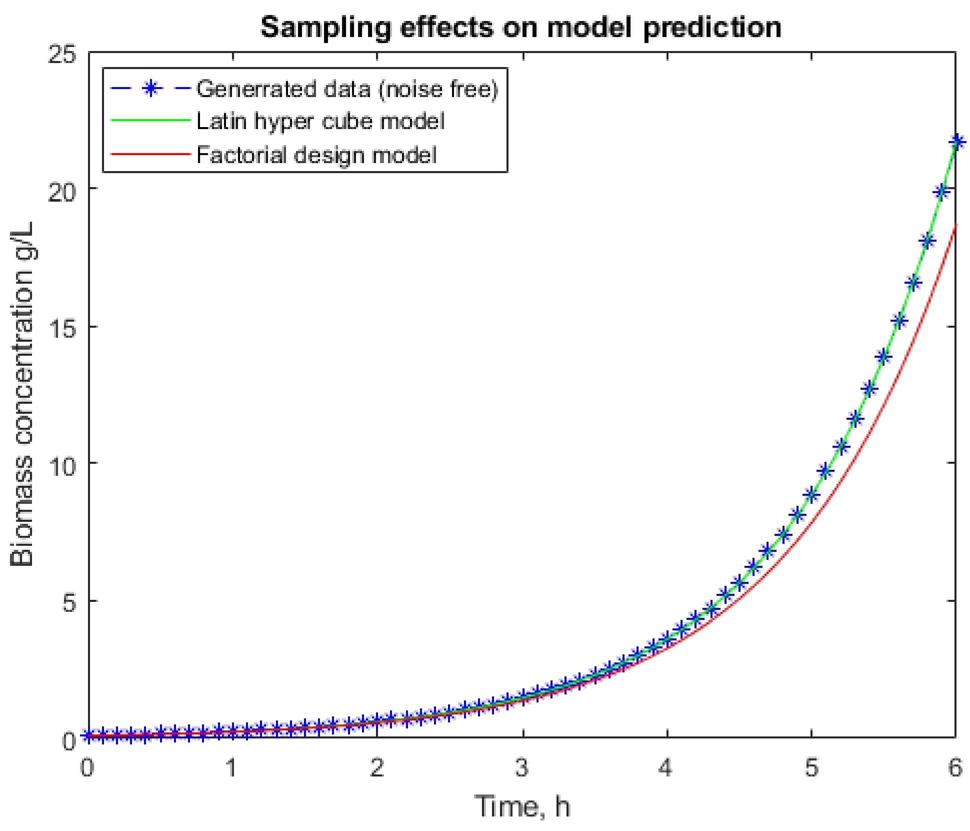


Figure 3.5 Comparison on biomass concentration model predictions, of different sampling methods with benchmarking problem

3.6.2. Screening

A benchmarking test was performed to check the effect of screening on accuracy and computational time (table 3.3). By only optimizing 10% of best initial parameter values, computational time can be reduced by 97%. At the same time accuracy does not change significantly. Model performance can be observed at figure 3.6 and identified parameter values at table 3.4.

Screening method	**Squared error of the model**	**Run time, s**
None	425.29	728.47
10% best	425.98	21.57

Table 3-3 Comparison of accuracy and computational time, with 10% screening and without

Parameter	**Real values**	**10% Screening**	**No screening**
U_{max}	0.9	0.904	0.898
K_S	0.3	0.248	0.083
q	4	3.951	3.987

Table 3-4 Identified parameter values using 10% screening and without it

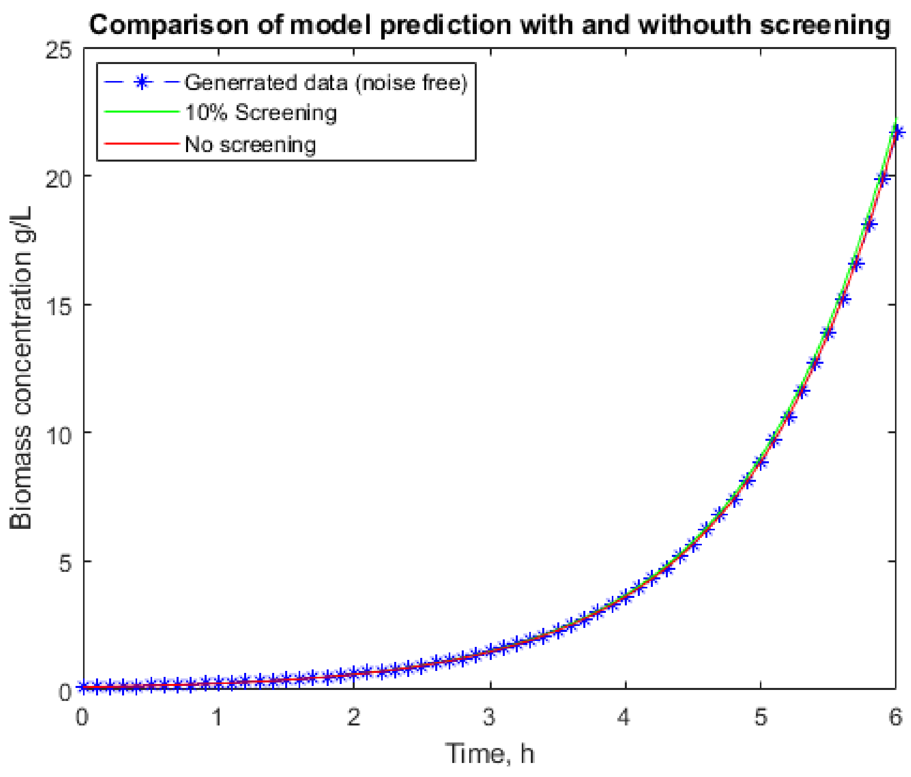


Figure 3.6 Comparison on biomass concentration model predictions with 10% screening and without

3.6.3. Re-optimization

Benchmark test was performed to compare accuracy pre- and post-re-optimization to assess the effect of re-optimization. It was found that re-optimization improved the benchmark problem solution accuracy by 195% (table 3.5). This is also clear from identified parameter values (table 3.6). Pre-optimization values are very susceptible to noise in generated data, as slope estimation is very sensitive to noise. This re-optimization step makes the method more robust to noise and makes model prediction very accurate (fig 3.7). Also using global re-optimization step makes the derivative method similar to integral method, so comparison between two is more direct.

	Squared error of the model
Pre re-optimization	36606.03
Post re-optimization	422.66

Table 3-5 Comparison of model accuracy pre- and post-re-optimization for the derivative method

Parameter	**Real values**	**Pre re-optimisation**	**Post re-optimisation**
U_{max}	0.9	1.064	0.899
K_s	0.3	0.565	0.242
q	4	4.543	3.989

Table 3-6 Comparison of the identified parameter values pre- and post-re-optimization for the derivative method

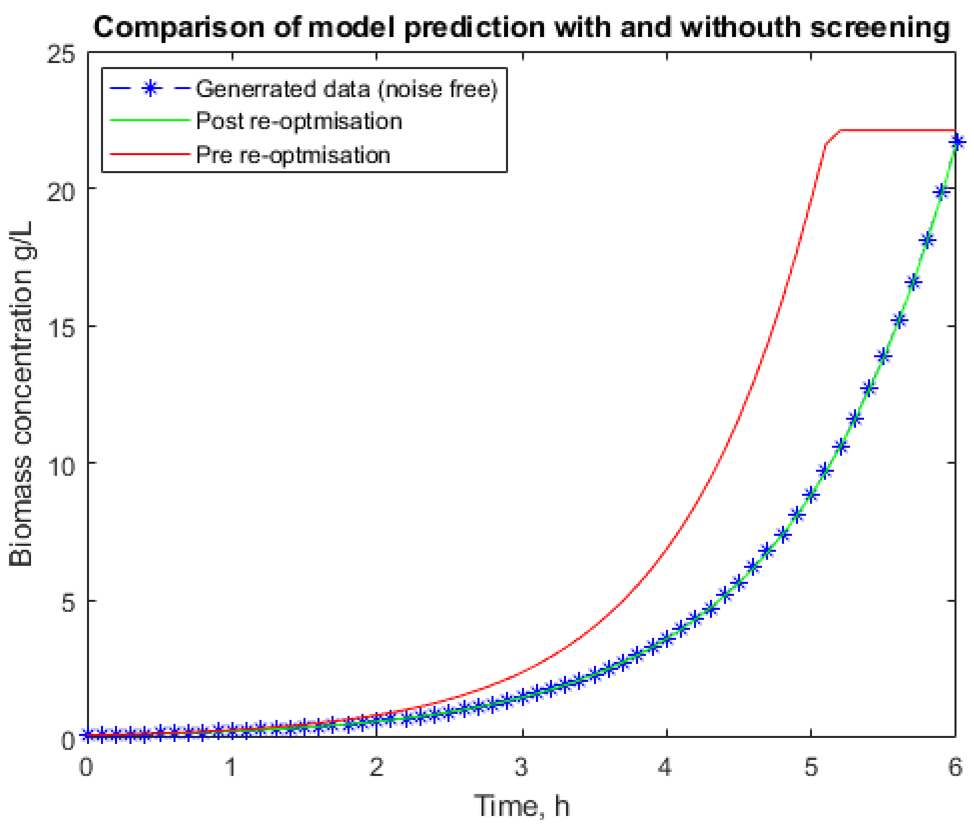


Figure 3.7 Comparison of model performance pre- and post-re-optimization

3.7. Summary

Chapter focuses on the derivative and integral methods as core state of art approaches. The integral method is improved with two modifications, Latin-hyper cube sampling and screening. Latin-hyper cube sampling is chosen as a sampling technique instead of factorial design or random distribution of samples, because it provides, on average, samples that are closer to global optimum given same number of samples. Screening is implemented to reduce number of samples needed to be optimized, as it is showed with benchmark problem, that accuracy of the solution is increased, and computational time is decreased significantly. For the derivative estimation method, it is chosen to estimate slopes, by using splines, as they are robust to noise, easy to use and does not require large amount of different data sets. Re-optimization step is introduced to increase solution accuracy. Final flowchart of both methods is shown, and these methods are used as comparison with the proposed state substitution method later in the work.

4. Problem visualization

In this chapter, we will discuss how the “curse of dimensionality” affects parameter identification and how it should be addressed. We will investigate a common problem of not being able to visualize the cost function error plane if we have a high dimensional problem, and what information we are losing because of it. Firstly, we will discuss what techniques can be used to visualize high dimensional problem and how to extract the same information, that we can normally extract from 3d visualization of cost function error plots. The methods investigated will involve, principal component analysis (PCA) visualization, self-organizing maps (SOM) and the used of sorted cost function performance graphs. For each of the techniques a simple example is used to help understanding how the technique works and what information it portrays. Secondly, we will address the concept of “complex” problems and how visualization techniques in conjunction with convexity criteria can allow the quantification and assessment of “complexity”. Lastly, we will discuss how these techniques can be used to determinate if state of art parameter identification methods, are capable of solving a given problem within reasonable time, or if a different approach should be taken (e.g., proposed state substitution method State-substitution).

4.1. Introduction

When solving parameter identification problems, it is very useful to be able to visualize your solution space, as this provides insight about the model you are working with. By observing solution space, you can verify if there are multiple optimum solutions, how much impact each parameter has on the model and how sensitive those parameters are. Simple way of visualizing the solution space of a model is to make an error plot. An error plot is a graph, where all parameters of a system a varied and each parameter combination is given a model performance value (error), when compared to the measured data. From such an error plot we can find how many optimum solutions there are by counting number of different valleys, that converge on a point called local minimum. We can assess parameter impact on the system, by measuring difference between maximum and minimum error values when only one parameter is varied. Similarly, we can assess parameter sensitivity, by measuring rate change of error when only one parameter is varied. To show example of how to use error plot consider equation 4.1. This simple system has single output of y, and two parameters P1 and P2. If we assume real parameter values are P1 = 1 and P2 = 2, then we can calculate error between y = sin(1) – 2 and ym = sin(P1) – P2, where p1 and P2 is varied.

$$y = \sin(P1) - P2 \qquad 4.1$$

Because P1 and P2 values are not know, we vary these values and plot them on x and y axes. Each combination of P1 and P2 has an error value on z axis. Error is calculated as square difference between global optimum (P1 = 1, P2 = 2), and any given parameter values. This construct error plot of equation 4.1 (fig. 4.1).

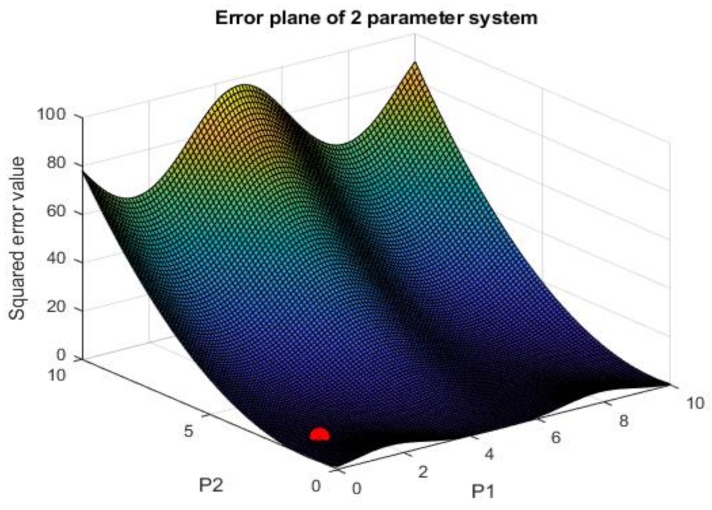


Figure 4.1 Error plot of the system (equation 4.1)

4.1.1. *Analysis of the error plane*

There are multiple ways to analyse the error plane of the system. Most useful observation can be broken down into a) number of local optima b) Impact of the parameters c) sensitivity of the parameters.

4.1.1.1. Number of local optima

Red circle in error plot marks global optimum i.e. P1 = 1, P2 = 2. By observing this error plot, we can tell that there two local optimum solution as we can see two separate valleys, which are defined by P1 parameter. First valley between values (0,4) and second valley between values of (4,10). This provides useful insight, telling us that if we use gradient based solver, we should start in correct valley to reach global solution.

4.1.1.2. Impact of the parameters

When trying to assess the impact each parameter makes to the system, we measure the maximum error value that can be reached by varying one parameter and keeping others at optimal values. We can tell that parameter P2 is more impactful as it can reach up to 70 on error axis while P2 is at optimal value. On other hand when varying P2 with P1 being fixed at optimal value, we observe only small up increase in error up to around 5. We can also observe that there a link of impact between parameters. As value of P2 increases, impact of the P1 increases too, but it is still lower than impact of P2.

4.1.1.3. Sensitivity of the parameters

When trying to assess the sensitivity of the parameter similar approach to impact measuring take place, where we vary one parameter keeping other constant. We can observe that P1 has more variation in error, than P2, making it more sensitive to changes. Where P1 has two peaks, throughout the sampling range, P2 only has a steady increase. With increasing values of P2 sensitivity of P1 is more pronounced, but general trend is the same.

These insights from the error plot provides valuable information about parameters itself, but also how to avoid local minimums and provides knowledge about the system. Unfortunately, this is only possible for two parameter systems, and a higher dimension problem, cannot be simply plotted, as there are not enough axes. Having a method to visualize higher order problems would allow the extracting of this information from high dimensional parameter identification problems.

4.2. Example systems

To facilitate explanation of the methods, which were used to visualize problems, and assess the system complexity, two system will be used. First model is simplified version of Monod kinetics (equation 4.2), this system consists of two states biomass (x, g/L) and substrate (S, g/L). This model also has three parameters U_{max}, K_s and q, which are maximum specific growth rate (h^{-1}), half-velocity constant (g/L) and substrate consumption rate constant (g/h) respectively. This model calculates rate of change in biomass $\frac{dx}{dt}$ and substrate $\frac{dS}{dt}$. Two simulate data sets using this model parameters values need to set to desired constants and model needs to be initialised with initial conditions for biomass (x) and substrate (S). Data sets were generated using numerical integrator ode45 in-build into MATLAB. Output of each ODE is added to existing value of the state producing time-series data of biomass (x) and substrate (S).

$$\begin{aligned} \frac{dx}{dt} &= u * x \\ \frac{dS}{dt} &= -q * u * x \\ u &= U_{max} * \frac{S}{K_s + S} \end{aligned} \tag{4.2}$$

Second model is a simple polynomial (equation 4.3). It is a simple function that outputs f(x) value based on input x values. It has twenty parameters (p1-p20). This model does not require any simulations as it is simple one value input-output system.

$$f(x) = x * p1 + x * p2 + x * p3 \dots x * p20 \tag{4.3}$$

These two systems are selected as they differ in two criteria. First is complexity, Monod kinetics are much more complex system then the polynomial model, because it involves ODE, rate of change, multiple equations, and time series data. Second criteria, which separates these systems is number of parameters, polynomial model has twenty parameters whereas Monod kinetics model has only three.

4.3. Utilizing PCA to visualize the model

The main problem regular error plot visualization faces with higher order problems is that there are not enough axes for each parameter. Even if we could plot higher dimensional planes, it would be hard or even impossible for us to interpret results as we are used to work in 3d space. Therefore, any potential solution should provide visualization in 3d. One of the possible solutions is to reduce number of parameters we need to plot converting model from n-parameter to 2-parameter model. We can achieve this by using PCA (Principal component analysis) developed by (Pearson 1901), modern version explanation can be found (Rasmus Bro 2014). PCA is a technique that is useful for the compression and classification of data. The purpose is to reduce the dimensionality of a data set (sample) by finding a new set of variables, smaller than the original set of variables, that nonetheless retains most of the sample's information. PCA achieves this by combines variables to produce new ones, that hold combined information about combines variables. For example, if we have two variables that seems to correlate (table 4.1), we can combine them by fractional addition (equation 4.4). Where F is new combined variable, w_i is fraction factor and x_j is variables.

$$F = w_1x_1 + w_2x_2 + \cdots + w_ix_j \qquad 4.4$$

	Ethanol g/L	Biomass g/L
Sample 1	90	60
Sample 2	45	120
Sample 3	0	180

Table 4-1 Example of two variables being heavily correlated

Only question is what fraction of each variable we should use to represent new combined variable. In this example there is strong and uniform correlation between variables, which would mean we should weight them evenly, but instead of weighting vector being w = [0.5 0.5], we should consider making it take into account that new combined variable represents two variables and not one, therefore it’s magnitude should be bigger than just and average of two variables. Similarity to vector addition, adding two vectors of same size at 90˚ angle, would make combined vector larger in magnitude (fig. 4.2). This can be applied when combining two variables as x and y axis are at 90˚. We can use unit vector and Pythagoras theorem to figure out what weighting we should use for same magnitude samples (fig. 4.2). This works out to be w = [0.7,0.7].

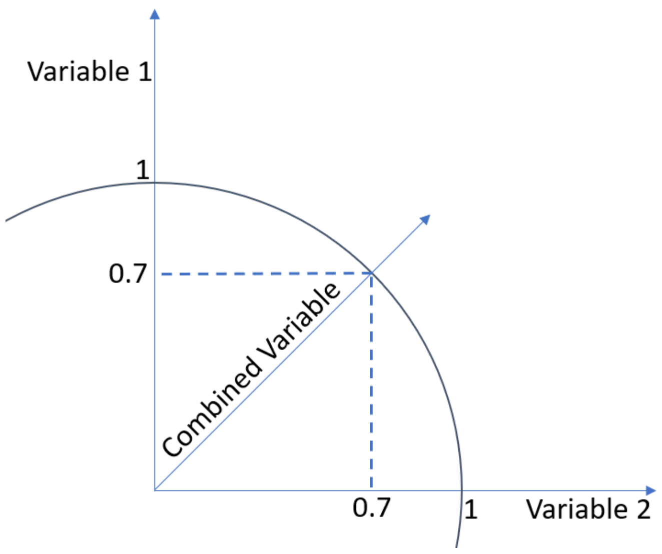


Figure 4.2 Example of unit vector addition

With this we can define PCA as a function of data X, weighting vector w and PC which is new variable representing whole system called scores (equation 4.5). This kind of combination of variables can be performed for more then two variables at the time, but it is quite rare that variables corelated so perfectly as in this example. Which is why we can only try to select such weighting vector w, that combined new variable represents most of the systems variability.

$$Xw = PC \tag{4.5}$$

So, problem becomes how to measure how much variability of the system variable PC represent in comparison to variables X. We can do that by performing regression of all X variables on PC, which will provide us with equation 4.6. Where r is regression coefficients and E is matrix of residuals.

$$X = PCr^T + E \tag{4.6}$$

Once we have our residual matrix, we can calculate how much variability is explained by PC variable (equation 4.7). Now we can optimise our selection of weighting vector w, by maximising variability expressed by PC.

$$\frac{\|X\|^2 - \|E\|^2}{\|X\|^2} * 100\% \tag{4.7}$$

In most cases one principal component will not be able to represent system in it entirety, this process can be repeated to generate second principal component, which will have explain less variability of the full system, than first principal component (PC).

Nevertheless, combining multiple PCs we can represent system well with lower number of variables, then system had initially. Using PCA we can create two principal parameters with highest explained variability. This will allow to plot similar plot as regular error plot. Downside to this method is that we will not be able to tell parameter sensitives anymore as they will be combined.

4.3.1. Example of applying PCA for visualization

Plot (fig 4.3) reveal similar information to normal error plot, it shows global/local optimums, and in this case, there is one clear “valley” around -1 of principal component (PC) 2, within it a global optimum around 0 PC1 and -1 PC2. There two potentially difficult areas while optimising this model. First the bottom of the main “valley”, for gradient based optimiser may look like an optimum if its optimisation path is along PC2 axis. The Second problem there is lot of flat surface, where gradient based optimizer can get stuck for the same reason. As these both problematic regions do not represent clear local minimums, it is subjective to judge how much of a difficulty they pose for the optimizer. It would depend on optimiser settings, initial guess, and other optimizer options.

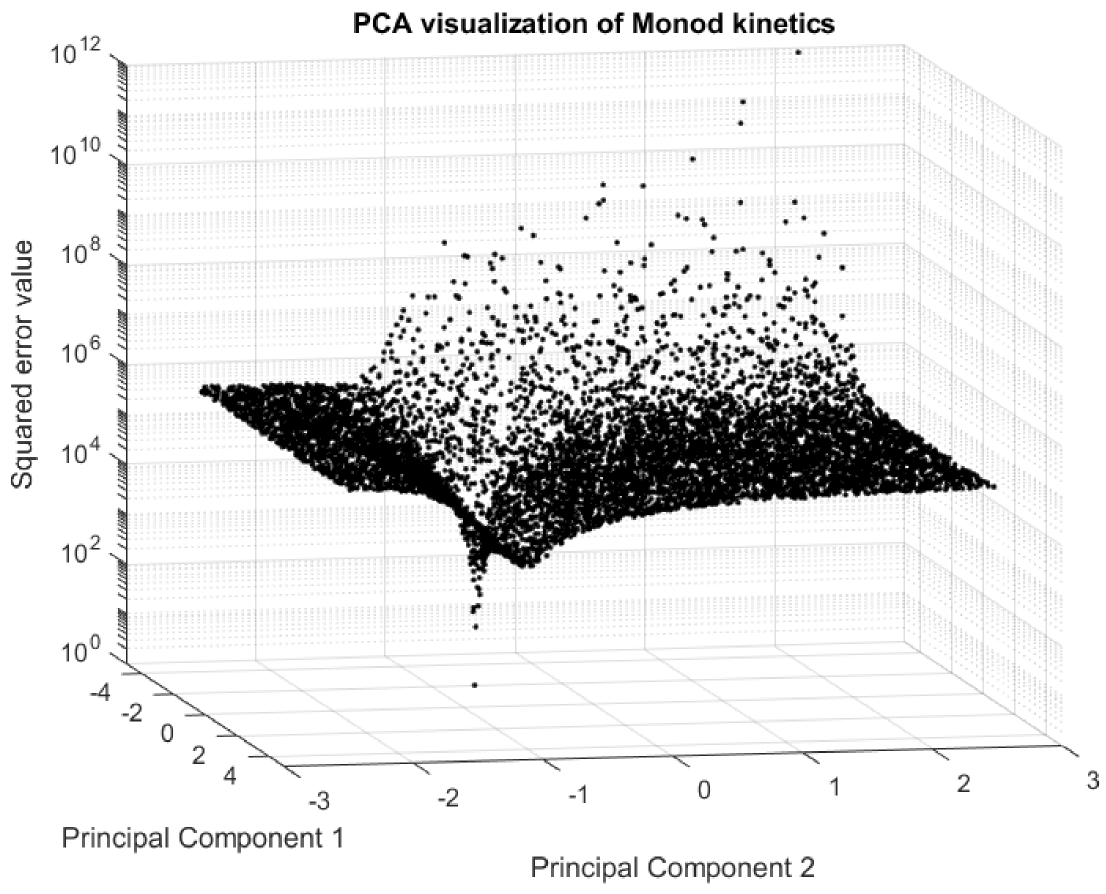


Figure 4.3 Error plot of Monod model first two principal components (PC's)

4.4. Convexity with PCA

When problematic regions in error plot/PCA error plot are not well-defined additional analysis is needed. In order to judge difficulty of optimization more objectively a convexity criterion is applied to all sample's points within the PCA error plot. Convexity is a criterion, of how likely sample is to reach an optimum. To meet convexity criteria equation below must be satisfied (Hass *et al*. 2018):

$$J(\alpha\theta(1) + (1 - \alpha)\theta(2)) \leq \alpha J(\theta(1)) + (1 - \alpha)J(\theta(2)) \tag{4.8}$$

Where J is cost function, θ(1) first set of samples/parameters, θ(2) is second random set of samples/parameters that satisfies relationship $||\theta(2) - \theta(1)|| = 1$ and α is a random location on connecting line between these two sample parameters sets.

This criterion can be calculated for every sample point (or combination of parameters), and then we can calculate percentage of samples, which are convex within sampling space. Providing a percentage convexity of a system.

To analyse model (equation 4.2), more in-depth we join this connectivity criteria with, PCA error plot to more information.

Figure 4.3 shows that flat areas are more problematic than "valley" for optimizers to solve, nevertheless this system has 27% convexity. This means if initial conditions for optimization algorithm are not within flat/red regions optimiser will solver problem with relative easily. Example of colour coded PCA visualization can be found in figure 6.13. Although, this method gives us good indication about complexity of system and possibility of running into local minimum it does not show which of the parameters are dominant.

4.5. Self-organizing map

Another way to visualize higher dimension problems are to use self-organizing map (SOM). First SOM was published by Teuvo Kohonen in 1982 (T.Kohonen 1982). A self-organizing map (SOM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples. Within SOM algorithm each data point competes to be represented among their neighbours. Winning node or sample is awarded, with higher change to be selected for next random sample comparison. This allow maps to form shapes, such as square, rectangular, hexagonal, toroid. Benefit for this technique that it will allow to have some insight of individual parameter importance, by comparing each parameter map with error map and to confirm PCA visualization results. SOM algorithm can be summarized in fallowing steps (T.Kohonen 1990):

1. Each node's weights are randomly initialized.
2. A vector is chosen at random from the set of training data.
3. Every node is examined to calculate which one's weights are most like the input vector. This is done using Euclidean distance formula. The winning node with lowest distance between input vector and its weight is commonly known as the Best Matching Unit (BMU).
4. The neighbour nodes of the BMU have their weights recalculated to pull them closer to the BMU node (equation 4.9). W is node weight, X is input vector, $\theta(u, v, s)$ is neighbourhood function, $\alpha(s)$ is learning rate function, s current iteration step.

$$W(s+1) = W(s) + \theta(i, j, s) * \alpha(s) * (X - W(s)) \qquad 4.9$$

Both learning rate and neighbourhood functions can be defined in various ways, but they should always be decreasing functions (Fausett 1994). Learning rate function controls the size of the weight vector. Most common ways of using learning rate are linear (equation 4.10), inverse of time (equation 4.11) and power series (equation 4.12) (J. Vesanto 2000). N is total number of iterations.

$$a(s) = a(0)\frac{1}{s} \qquad 4.10$$

$$a(s, N) = a(0)\left(1 - \frac{s}{N}\right) \qquad 4.11$$

$$a(s, N) = a(0)e^{\frac{s}{N}} \qquad 4.12$$

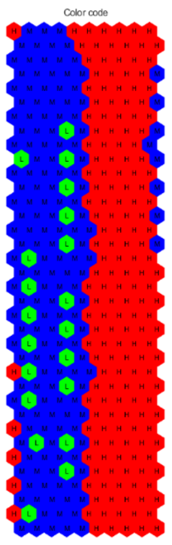
Neighbourhood function defines which nodes considered to be in the neighbourhood of the BMU. Most commonly used function is Bubble (equation 4.13) (W. Natita 2016). Nc is the index set of the neighbour nodes close to BMU node.

$$\theta(i, j, s) = \begin{cases} a(s), (i, j) \in N_c \\ 0, (i, j) \notin N_c \end{cases} \qquad 4.13$$

5. The winning weight is rewarded with becoming more like the sample vector. The neighbours also become more like the sample vector. The closer a node is to the BMU, the more its weights g*et al*tered and the farther away the neighbour is from the BMU, the less it learns.
6. Repeat step 2 for N iterations.

4.5.1. Example of SOM application

SOM can be difficult to analyse, but they are consistent with other visualization techniques. First it should be noted is that green nodes seem to be scattered as it would suggest there are multiple local minima, but by observing U-matrix we can see that line that separates those green areas is very tiny in terms of distance. This means it is a valley rather than multiple local minimums. This observation matches with observation done with PCA visualization. This reassures that visualization techniques are showing correct results. The biggest benefit that SOM provides, and that other visualization techniques lack is variability of each parameter within n-dimensional plane. For Monod model (equation 4.2) if we compare figure 4.5 colour coded figure with figure 4.6 of each parameter figure we can see that pattern of U-matrix matches U_{max} parameter pattern. Starting with high values on the right side with decreasing values to the left and increasing again once past lowest valley point. This observation shows connection between changes in U_{max} parameter and the model error, implying high impact from the parameter. How to use SOM maps to assess parameter importance will be discussed in further chapters.



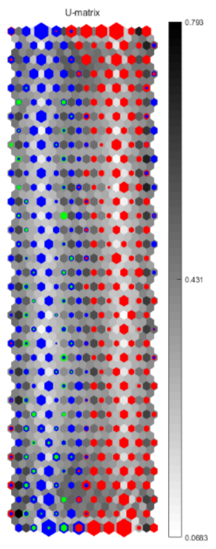


Figure 4.4 Left 2d SOM of Monod model (equation 4.2), colour coded based of model error, where green (L) is low error, blue (M) is medium error, and red (H) is high error. Right 2d SOM with same colour code as left, but also showing relative distance in n-dimensional plane represented as colour bar.

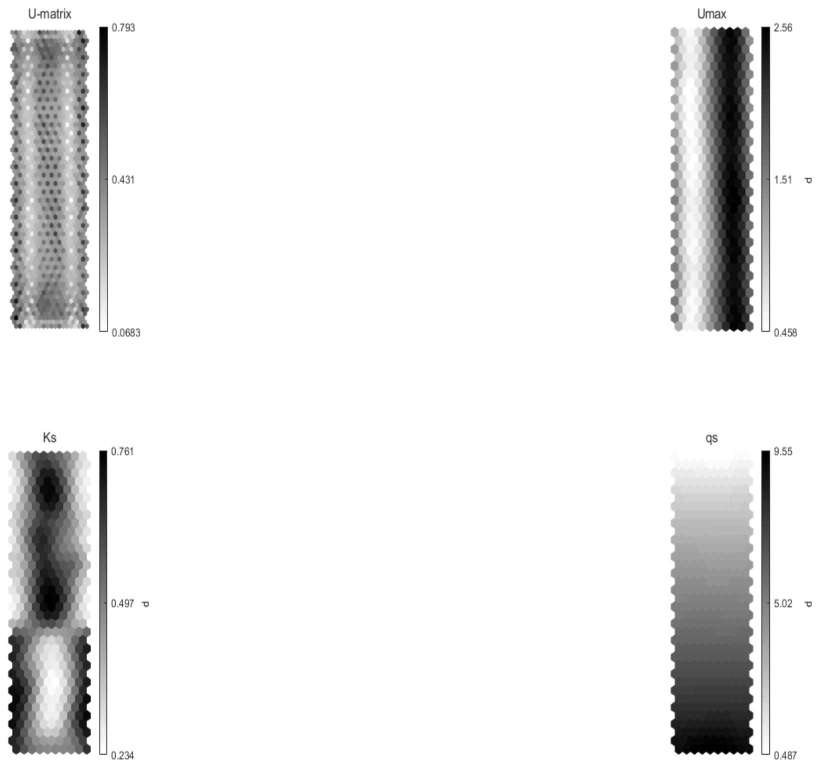


Figure 4.5 From top left: 1) n-dimensional plane represented in 2d SOM with relative distances as colour bar. 2) U_{max} parameter variation within n-dimensional plane, parameter values denoted in colour bar. 3) K_s parameter variation within n-dimensional plane, parameter values denoted in colour bar. 4) q parameter variation within n-dimensional plane, parameter values denoted in colour bar.

4.6. Performance optimum vs parameter optimum

To ensure, that visualization techniques produce correct results, optimization of the sample space was performed. Taking thousand samples from the sample space and optimizing each of them individually. This is computationally very demanding, but it helps to make sure that other techniques provide reliable results. Once all sample points were optimized, their cost function values are sorting in acceding order and plotted to show potential local minimums. Simple problems will tend to have large flat regions because most of sample will converge to same cost function value. Where complicated problems will have varied region indicating multiple local minimums. For comparison two systems are observed, Monod kinetics (equation 4.2), and polynomial (equation 4.3).

Figure 6.13 shows the results of the Monod model (equation 4.2). Model is performing as expected, for single valley problem. The system has one flat area to indicate the “valley” and drop down to show that is has one global minimum that is hard to reach.

With low tolerance even non-convex point can be optimized to reach optimum “valley”. Optimizing just convex points provides advantage over optimizing all sample points as only a fraction of samples needs to be optimized to achieve same trend.

On other hand polynomial model (equation 4.3) does not perform as you would expect from model with 100% convexity (figure 6.2). It shows lots of local minimum before reaching flat area of global minimum, but difference between worst and best optimization is very small (10^{-10} scale). This indicates that although there many different local minimums it does not affect overall model error.

These two models show two different behaviours, one where cost function optimization leads to correct parameter values, hence parameter optimum can be reached and other where cost function optimization led to good model performance, but wrong parameter values, hence performance optimum can be reached. Model with only performance optimum, have one feature in common - their parameters are only relatively sensitive to each other. This means that if parameter value difference is relative same, performance of model will not suffer even when those values are far away from correct values. For examples if take polynomial model, as it is additive model all parameter values (p1, p2…) can be changed in position and it would not affect overall model performance. Therefore, identifying these parameter values correctly becomes impossible.

4.7. Summary

A key reason to search for different approach to parameter identification is when state of art methods, struggle to produce accurate results or their computational demand is too high. Both are encountered when model is complex. Unfortunately, complexity of a model is not directly associated with model size or number of parameters. By performing analysis of a model by using techniques described in this chapter it is possible identify if model complexity is something that would be of concern and take appropriate approach when identifying parameters. Firstly, convexity criteria evaluation is a good start, as it is fast analysis and can immediately tell if model is not complex. Because, parameters upper and lower limits will affect accuracy of any identification algorithm, to obtain accurate convexity value, calculation should be performed within same boundaries as identification algorithm. Performing PCA visualization with marked convex points, would be next step. By observing PCA visualization scatter plot there are several things to look for, first how many local minimums can be observed, second are local minimums separated with convex or non-convex regions. Multiple local minima make model more complex, but it is much worse if they are separated with non-convex regions. This leads to problem, where optimization algorithm not only will need to search for global minimum between all local minimum, but also it will not converge at all when situated in non-convex region. When dealing with models that have multiple local minima with non-convex regions in between, models should be treated as medium complexity and parameter identification algorithm set appropriately. This might involve modifying state of art approach to better deal with specific model or use alternative methods. Lastly, if still not certain about complexity of a model an optimization full sample space can be done, to observe if model can be optimized only towards performance or also towards correct parameter values. It is suggested to sample parameter space with Latin-hyper cube, as it will make this computational demanding analysis more efficient. Once all optimized cost function values are sorted, obtained graph can be analysed. Within this sorted minimization graph each flat region represents local minimum, this should be same as number local minimum observed in PCA visualization. If graph has a “curve” such as figure 4.8, and difference between maximum and minimum cost function values is large, this tell that system is complex and should be approached with caution.

This kind of approach is enough to quantify complexity of a model. Although overall complexity may not be numerically quantified, this approach has enough quantitative measurements to compare model to each other to determinate which is more complex.

5. Methods

In this chapter the working principles of the proposed state substitution method are explained in detail, and any methods involved within the framework of the state substitution method. First the key steps of the State Substitution method are explained, which are approximation of data, decoupling of the ODE model, integration of sub-sets of the model and parameter identification and re-optimization of the whole model structure. Then any additional algorithms required for State Substitution to work are explained, in addition to any supportive methods, which include the use of a bipartite chart, sub-set solver, global solver, data generation and algorithm for establishing hierarchy of sub-sets.

5.1. State Substitution method

The proposed state substitution method for parameter identification of the unknown constants of models described by ODEs consists of four main steps a) approximation of data b) decoupling of the system of nonlinear ODEs c) integration of the subsets of nonlinear ODEs and parameter identification d) re-optimization of full ODE model. In order to assist with the explanation of the method a simple example is used as a demonstration throughout this chapter (equation 5.1). It is assumed that there is a set of measured data, i.e. measurements of X, S and P. at various time points. The data measurements do not have to be equally spaced and can have varying noise levels (fig. 5.1). The aim is to use the data to estimate the unknown model parameters (U_{max}, K_s, q and q_s) such that predictions of the model would match measured data. As discussed in chapter 2, most of the methods would numerically integrate the whole set of coupled ODEs and using iterative optimisation algorithm would get the optimal parameter values.

$$\begin{cases} \frac{dX}{dt} = uX - DX \\ \frac{dS}{dt} = -quX - D(S - Sf) \text{ , where } u = \frac{U_{max}S}{K_s + S} \\ \frac{dP}{dt} = q_pX - DP \end{cases} \qquad 5.1$$

Where t is time (h), X is biomass concentration (g/L), S is substrate concentration (g/L), P is product concentration (g/L), D is dilution factor, Sf is feed substrate concentration (g/L), U_{max} is maximum specific growth rate (h^{-1}) ,K_s is half-velocity constant (g/L), q substrate consumption rate constant (g/h) and q_p is product production rate constant (g/h).

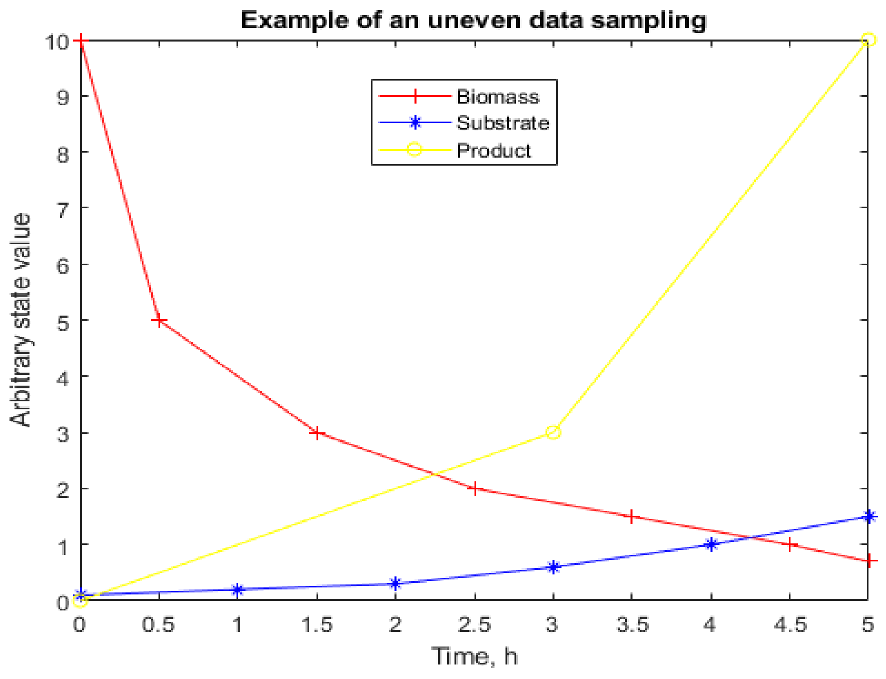


Figure 5.1 Example of an uneven sampling

a) Approximation of data

It is a required step to generate data sets from measured data that can be used to decouple the system. As measurements do not consists of sample points at all possible time points, we require to approximate it. Approximation can be done with various methods such as polynomials, splines etc. as discussed in chapter 3. In this work approximation of data is done with cubic spline, which is explained in detail in in section 5.5. For simpler explanation we will use polynomial as an approximation technique for this example. With polynomial any measured data can be expressed as (equation 5.2), where $\hat{Y}_t$ is approximated data value at time t, α_n values are optimised to provided good data fit of $\hat{Y}_t$ to measured data. Finalized approximated data set can produce a state value at any time point. This is important for later step withing the method as decoupled sub-systems get integrated state values need to be known sat every time point.

$$\hat{Y}_t = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \cdots + \alpha_n t^n \qquad 5.2$$

b) Decoupling of the system

During this step we replace any coupled states within subset, with polynomials produced in step a). This allows to solve each subset individually, thus reducing the solution space. If consider example system (equation 5.1), we need to create a different polynomial for each state within the system (equation 5.3). These polynomials then can replace coupled states within each ODE to effectively decouple the whole system (equation 5.4). This allows the system to be solved as three sub-systems, thus reducing the search space.

$$\begin{aligned}\hat{X}_t &= \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \dots + \alpha_n t^n \\ \hat{S}_t &= \beta_0 + \beta_1 t + \beta_2 t^2 + \dots + \beta_n t^n \\ \hat{P}_t &= \gamma_0 + \gamma_1 t + \gamma_2 t^2 + \dots + \gamma_n t^n\end{aligned} \tag{5.3}$$

$$\begin{gathered}\frac{dX}{dt} = \frac{U_{max}\hat{S}_t}{K_s + \hat{S}_t} * X_t - DX_t \\ \frac{dS}{dt} = -q\frac{U_{max}S}{K_s + S}\hat{X}_t - D(S - Sf) \\ \frac{dP}{dt} = q_p\hat{X}_t - DP\end{gathered} \tag{5.4}$$

To visualise how decoupling reduces search space, we can start by visualized full system with its four unknown parameters (U_{max}, K_s, q and q_s). To draw a four-dimensional search space in 3d we need to construct a tesseract. Tesseract is four-dimensional equivalent of a cube, which is three-dimensional search space (fig. 5.2). After the decoupling we have three separate search spaces that consist of a plane, cube, and a straight line (fig. 5.3).

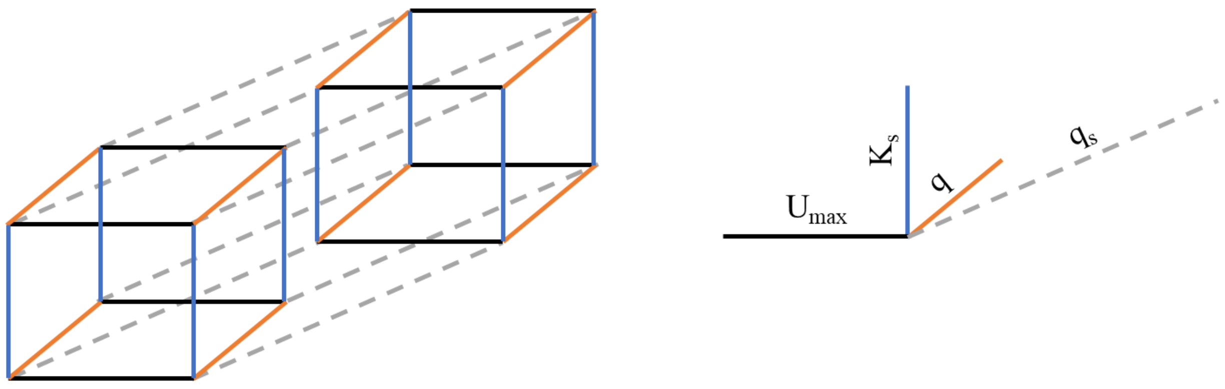


Figure 5.2 Search space visualization of the example system (equation 5.1) which has four parameters

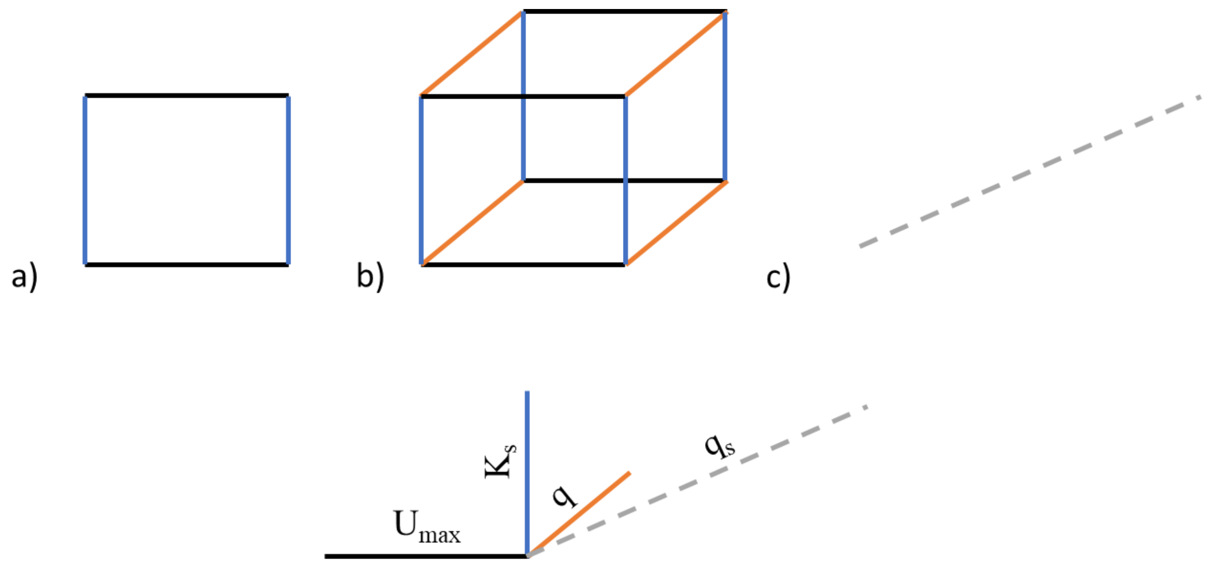


Figure 5.3 Decoupled search space of the exampled system (equation 5.1) into three separate sub-systems (equation 5.4)

c) Integration of subsets and parameter identification

If a polynomial regression model replaces coupled states, each ODE can be integrated separately, in this work Ode45 solver was used for that purpose. It should be noted that choice integrator will affect results if your system is stiff, so dependent on your system adequate solver should be used (Spyridon Dallas 2017). Whereas for parameter identification each subset was solved using lsqnonlin. This solver was chosen, because it is made for solving non-linear problems and performed squaring of the error. Solving order is important as decoupled subsets can be solved individually, but only a specific order gives best results. To come up with structure hierarchy, bipartite chart is used to separate most influential states and parameters. Considering influence of state and parameters, subsets are separated in different levels. Levels must be solved in sequential matter, where everything within same level can be solved simultaneously. Detail explanation of how model hierarchy is defined is explained in section 5.3.

d) Re-optimization

Solving each individual ODE will only provide sub-optimal parameter values, because we do not account for interactions between different ODEs. To ensure statically optimal parameters values re-optimization should be done using a global solver (in this work a genetic algorithm is used) and the sub-optimal parameter values as initial guesses. This step makes the method robust to noise, as readjustment can be done for parameters which are harder to identify from subsets only. To achieve the best results each state should be weighted, according to hierarchy in cost function. Meaning top level states have higher weightings and lowest level states have lowest.

5.2. Bipartite graph and structure hierarchy

To be able to decouple more complex systems, knowledge of the involved parameter/state relationships and their influence on the system is required. One way of observing these relationships is through the construction of a bipartite graph (Bomhoff *et al*. 2010). A bipartite chart is constructed by plotting the connection between the model state and other states and parameters within the model. Required information can be extracted directly from the ODEs. This type of chart is required to design the order in which ODE's need to be solved, because after decoupling it can be solved in number of ways. It can be adapted to any system. To understand how it works an example is provided using an example system (equation 5.1).

By inspection of the ODEs (equation 5.1) we can construct a bipartite graph by putting states of the model on the top of the graph and parameters/states of the model at the bottom. A connection can then be drawn from the bottom to the top showing the relationship between the state and parameter/state of the model. Dilution factor D and substrate concertation Sf are not on a graph, because these values are known constant values. The resulting graph is shown in figure (5.4).

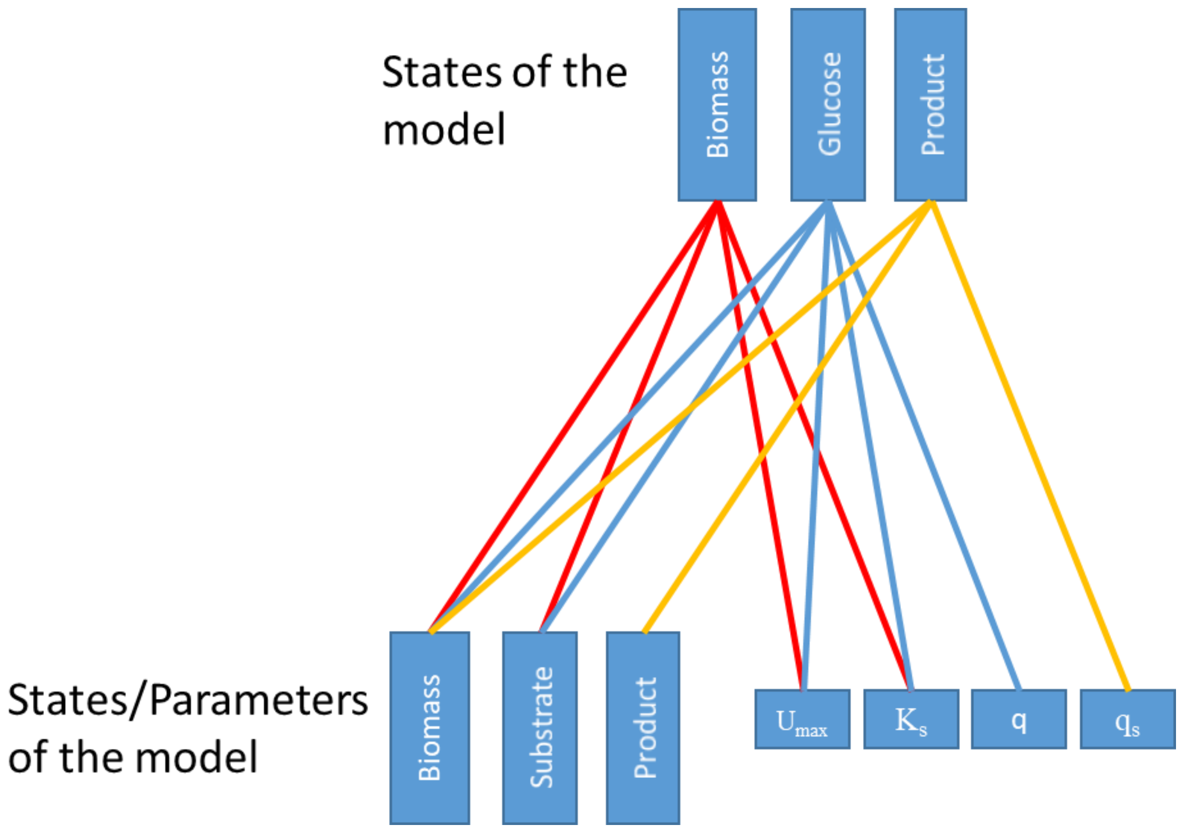


Figure 5.4 Bipartite chart of the example system (equation 5.1)

By the assessing the structure of bipartite chart we can construct a decoupling structure. Starting with the states with most connections to least. In this example we can observe that the biomass(X) state affects all three states which means it's of highest importance and should be solved first during decoupling. Furthermore, parameters U_{max} and K_S estimates from the biomass(X) ODE can be used in further calculation to ensure that they do not change, creating disturbance in X state model. In addition to this it can be observed that state S and state P have no common parameter connections, which means that they can be solved both at the same time after solving for state X state ODE. To summarise the decoupled state ODE hierarchy is constructed in two level, first one consisting of biomass state ODE and second level consisting of substrate and product ODE. Full hierarchy is provided in Figure 5.5:

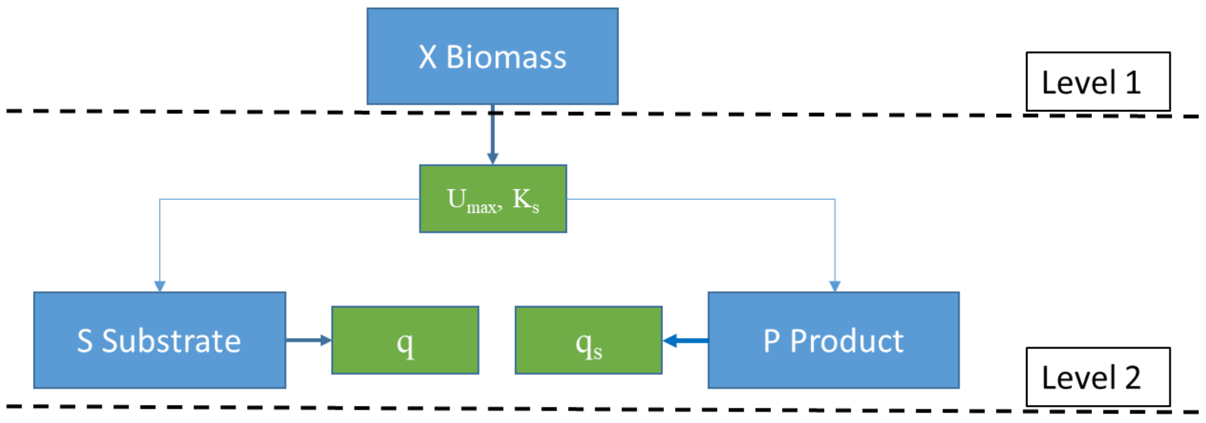


Figure 5.5 Hierarchy of decoupled ODE's for Monod model

5.2.1. Parameter impact

Although bi-partite chart provides information about connections between the model state and other model states and parameters, it does not provide how impactful those parameters are for overall system. This is important as to achieve best result we want to identify most impactful parameters from state of highest importance. In other words, we need to solve for most sensitive parameters of the system first and then for the rest in ascending order of sensitivity. To perform full sensitivity analysis of all the parameters can be very hard for high dimensional system, with high number of parameters. Nevertheless, a Self-organizing map can be used to perform parameter impact correlation on overall system to determinate most impactful variables, which effect overall system. SOM's produce component maps and unified distance maps (chapter 4), which we can colour, based on average error values of each cluster. We colour the lowest error cluster as white and highest error cluster as black and everything in between as a gradient. SOM produced component maps already are in this format. This allows direct cross-correlation of component maps versus system error map. Cross-correlation of maps is performed pixel by pixel to produce heat maps of parameter impact. Cross correlation matching factor of 1 denotes perfect match and 0 a complete mismatch. These correlations are performed with a normal component map and inverse component map, colour wise, as parameters can be positively or negatively corelated. Average correlation factor can be calculated from these heatmaps. Example system (equation 5.1) produced heatmaps can be seen in figures 5.6-5.7, and their correlation factors in table 1. Matching factor calculation are performed five times and standard deviation is presented together with results. These figures, together with table, show that most impactful parameter for this system is U_{max}, followed by q then K_S and q_S. This agrees with our bipartite chart observation that U_{max} value should be identified from biomass state and level 1.

Parameters	**Positive Matching Factor**	**Negative Matching Factor**
U_{max}	0.81 ± 0.03	0.50 ± 0.00
K_S	0.65 ± 0.02	0.63 ± 0.03
q	0.68 ± 0.03	0.60 ± 0.02
q_S	0.63 ± 0.01	0.61 ± 0.01

Table 5-1 Table of positive and negative mean matching factor of each parameter for system

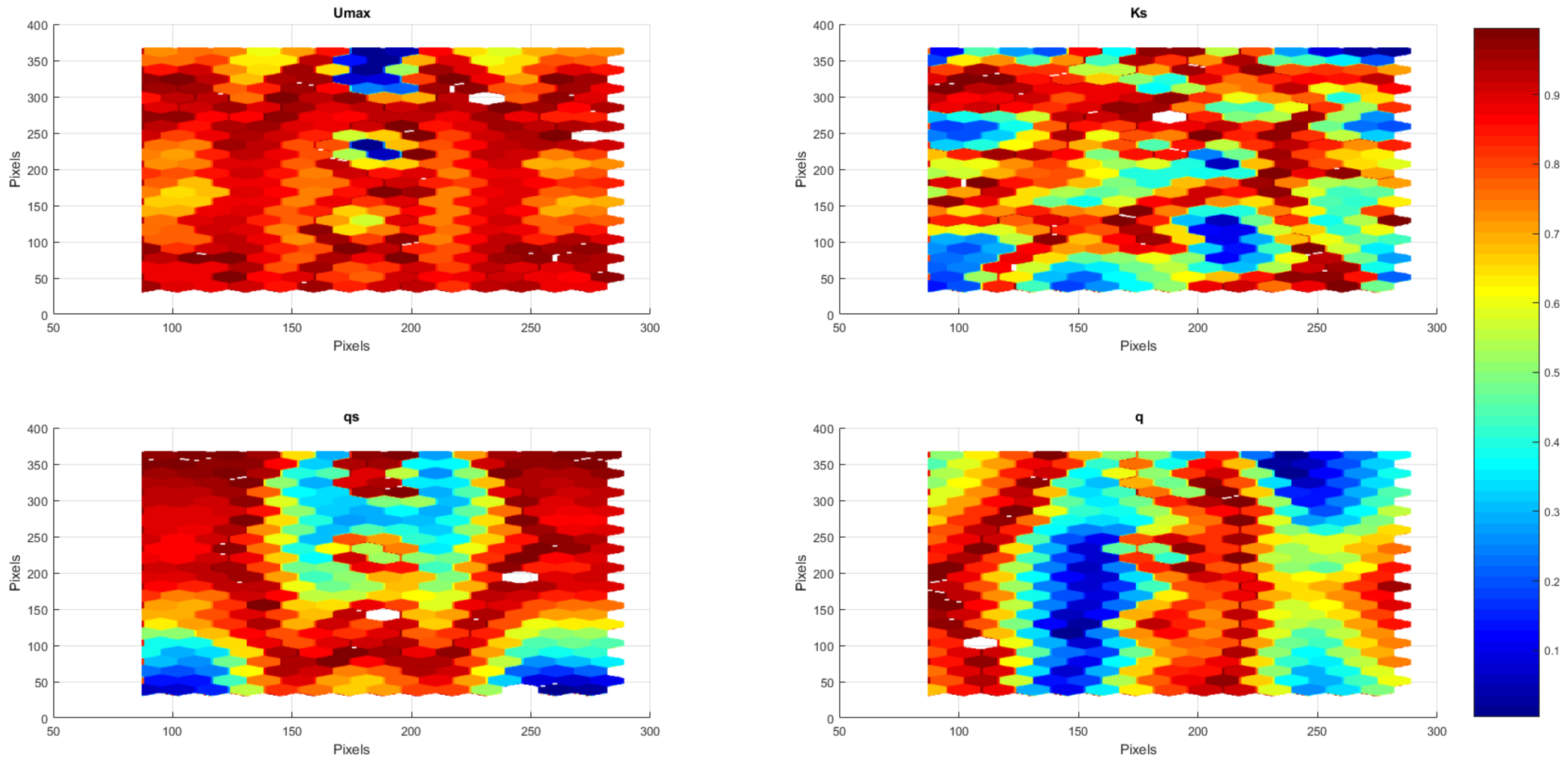


Figure 5.6 Positive cross-correlation between each parameter map and overall error map.

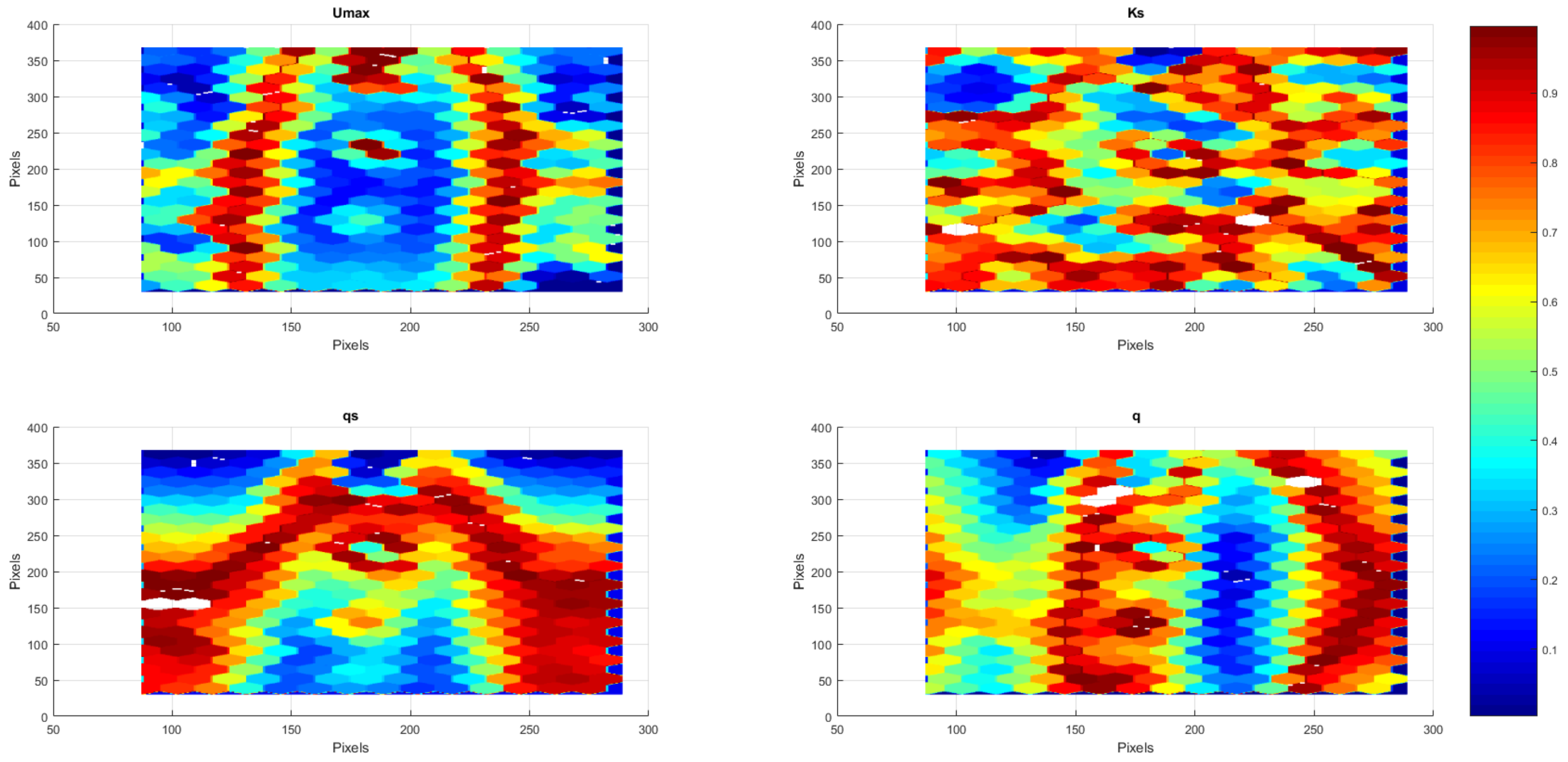


Figure 5.7 Negative cross-correlation between each parameter map and overall error map.

5.3. Data generation

For case studies which use simulated data sets, data is generated by solving ODE system given initial conditions. The Ode45 solver in MATLAB is used for this task as it is solver that is most fit for the task based on (Spyridon Dallas 2017). The options for solver are provided in Table 4.2, with values provided for example system (equation 5.1):

Option	Description	Value
Nonnegative	The scalar or vector selects which solution components must be nonnegative.	1:3 (Covers all states)
RelTol	Relative tolerance. This tolerance measures the error relative to the magnitude of each solution component.	1e-13
AbsTol	Absolute tolerance. This tolerance is a threshold below which the value of the solution becomes unimportant.	1e-13

Table 5-2 Options used for ode45 solver

The Sampling time was varied between 0.1h and 0.3h. The Lower bound was selected by calculating time constants of system responses using simple method described by (Niemann and Miklos 2014) and taking 1/10 of fastest response time constant , to ensure no process dynamics are lost. Whereas the upper bound was found by trial and error at the point where the method still worked, but performance is not satisfactory.

After initial data generation is finished, desired amount of random white noise is added to noise free values to create 'measured' values. In this work all data was exposed to 5% or 20% random white noise. General structure of noise addition is provided below:

$$\text{Measured values} = \text{Noise free data} *(1 + ((b-a)*\text{Random number} + a)) \qquad 5.5$$

Where a is upper percentage bound, b is lower percentage bound and random number stand for randomly generated number between 0 and 1. For example for 5% noise a = 0.05, and b = -0.05.

5.4. Approximation spline

In this work cubic spline was used to approximate measurement data sets. Splining data allows us to supply ode solver with required data points for integration. Furthermore, when done correctly, splining of data cancels out some of the system noise, without losing any crucial information. If approximation is done to high level information about system kinetics is lost due to smoothing it out, if it is done to low level noise is dominant instead of system dynamics. Therefore, it is important to optimise your approximation method to not lose process dynamics from measurement data. The chosen smoothing spline function in MATLAB was csaps. It's a cubic smoothing spline of given data x, y. It allows you to specify a smoothing parameter p, which controls the smoothness level. When p is 0 csaps fit least squares straight line to given data, on other hand when p is 1 fitted function is the `natural' or variational cubic spline interpolant. This smoothing spline minimizes the function provided:

$$p\sum_{j=1}^{n} w(j)|y(:,j) - f\big(x(j)\big)|^2 + (1-p)\int \lambda(t)\,|D^2 f(t)|^2 dt \qquad 5.6$$

Where w(j) is weighting vector, y(:,j) is provided data matrix, f(x(j)) is newly replaced value matrix, j is length of time vector(or length of x axis points), D^2f(t) is second derivative of function f.

Value of p was chosen to be 0.95 as during initial testing of error sensitivity to smoothing parameter revealed it to be optimum value in most cases.

5.5. Sub-system solver

The solver that was chosen to solve decoupled sub-systems was lsqnonlin. This solver is non-linear least square solver and incorporated error squaring into algorithm, for these two reasons it was best fit solver as a sub-system solver. The options for solver are provided in Table 4.3, with values provided for example system (equation 5.1):

Option	Description	Value
DiffMaxChange	The scalar value, of maximum change for finite-difference gradients	0.1
TolFun	Function tolerance, this is value is threshold which algorithm stop if objective function value reaches it.	1e-8
TolX	Step function tolerance, this is value is threshold which algorithm stop if change in objective function is lower than the threshold.	1e-8
MaxFunEval	Maximum number of function evaluation allowed before termination of the algorithm	10000
MaxIter	Maximum number of iterations allowed before termination of the algorithm	2000
Jacobian	Specifies if algorithm should use finite difference Jacobians or user defined ones	'on'

Table 5-3 Options table for sub-set solver lsqnonlin

Initial parameter points where supplied as 0.5 for each subsystem, as its reasonable initial conditions for most cases, being positive and close to 0. It should be noted initial parameter guess have no effect on solution, as long it is within solution bonds. The lower bounds where specified as 0 and upper bounds as double the real parameter value rounded up to closest integer. Each subset was supplied with enough different data sets so all parameters within sub-system would be observable. For the example system (equation 5.1) two different data sets where required across all three subsystems. To increase solver accuracy and speed Jacobian matrix was defined for each of the sub-systems. Structure of Jacobian matrix is showed in equation 5.7:

$$\frac{d}{dt}\frac{dx_i}{dp_j} = \frac{df}{dp_j} + \frac{df}{dx_i} * \frac{dx_i}{dp_j} \tag{5.7}$$

Where x_i is state which is integrated, f is function which is integrated, p_j is parameter, which is integrated, and t is time.

This Jacobian matrix provides solver trust-region function removing need of evaluating function of each parameter, consequently it allows solver to reach optimal solution faster. Cost function consisted of difference between measured values of the state and predicted values of the state. For each unique data set used by the solver one cost function was created. In a case of multiple cost function, total error was calculating by adding all cost functions with equal weightings. Jacobian matrix is collection of parameter sensitivities over time, this allows to estimate all parameter change with single function. To obtain the Jacobian matrix function for the example system (equation 5.1), we start with defining unknown parameter vector (equation 5.8) and ODE function vector (equation 5.9). Then we can replace corresponding terms within Jacobian matrix function (equation 5.7), to obtain general form (equation 5.10). Letting $S([X,S,P],\widehat{p}_j,t)=\frac{d[X,S,P](t)}{d\hat{p}_j}$ to be sensitivities of states (X,S,P), for the model parameter p$_j$ we get sensitivity ODEs, which then are used as first order derivatives in objective function during optimisation. It should be noted that sensitivity initial conditions are always $S([X,S,P],\widehat{p}_j,0)=0$. Example of final Jacobian matrix for all states and U$_{max}$ parameter is provided in (equation 5.11).

$$\hat{p}_j = [\mathrm{U}_{max}\ \mathrm{K}_s\ \mathrm{q}\ \mathrm{q}_s] \qquad 5.8$$

$$f(t)=[f_1(t),f_2(t),f_3(t)]^T=\begin{bmatrix} \frac{U_{max}S}{K_s+S}*X-DX \\ -q\frac{U_{max}S}{K_s+S}X-D(S-Sf) \\ q_pX-DP \end{bmatrix} \qquad 5.9$$

$$\frac{d}{dt}\begin{bmatrix} \frac{dX(t)}{d\hat{p}_j} \\ \frac{dS(t)}{d\hat{p}_j} \\ \frac{dP(t)}{d\hat{p}_j} \end{bmatrix} = \begin{bmatrix} \frac{df_1(t)}{dX(t)} & \frac{df_1(t)}{dS(t)} & \frac{df_1(t)}{dP(t)} \\ \frac{df_2(t)}{dX(t)} & \frac{df_2(t)}{dS(t)} & \frac{df_2(t)}{dP(t)} \\ \frac{df_3(t)}{dX(t)} & \frac{df_3(t)}{dS(t)} & \frac{df_3(t)}{dP(t)} \end{bmatrix} \begin{bmatrix} \frac{dX(t)}{d\hat{p}_j} \\ \frac{dS(t)}{d\hat{p}_j} \\ \frac{dP(t)}{d\hat{p}_j} \end{bmatrix} + \begin{bmatrix} \frac{df_1(t)}{d\hat{p}_j} \\ \frac{df_2(t)}{d\hat{p}_j} \\ \frac{df_3(t)}{d\hat{p}_j} \end{bmatrix} \qquad 5.10$$

$$\frac{d}{dt}\begin{bmatrix} S(X,U_{max},t) \\ S(S,U_{max},t) \\ S(P,U_{max},t) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{U_{max}S}{K_s+S} - D & \frac{U_{max}S}{K_s+S} - \frac{U_{max}SX}{(K_s+S)^2} & 0 \\ -q\frac{U_{max}S}{K_s+S} & q\frac{U_{max}SX}{(K_s+S)^2} - q\frac{U_{max}X}{K_s+S} - D & 0 \\ q_p & 0 & -D \end{bmatrix} \begin{bmatrix} S(X,U_{max},t) \\ S(S,U_{max},t) \\ S(P,U_{max},t) \end{bmatrix} \qquad 5.11$$

$$+ \begin{bmatrix} \frac{SX}{K + S} \\ -q\frac{SX}{K_s+S} \\ 0 \end{bmatrix}$$

5.6. Global solver

Global solver chosen for the proposed state substitution method is Global search as discussed in section 2.4. Global search algorithm requires only couple of the inputs a) Objective function b) Initial parameter values c) Parameter value boundaries d) Select local solver e) Any additional solver options (optional). For the proposed state substitution method global search was used during re-optimisation step for all four case studies.

a) Objective function was constructed for all cases studies as a sum of weighted least-squared problem. First, all state values were normalised, to avoid over-representation of high values states in total error of the objective function. Then each of the of the model state values were subtracted from the measured state values and squared. Weighting was assigned based on the level of in the hierarchy model, where first level is assigned highest weight value.
b) Initial parameter values for the global search were values that were optimised, by the decoupling approach.
c) Parameter value boundaries were defined as ± 20% of the initial parameter values. This is an arbitrary decision, but it aims to keep global solver search space close to initial start point. This encourages the global solver to make precise search around the starting point
d) Local solver was selected to be ‘fmincon’. This is default solver and the most robust one.

e) Only single additional solver option was added, which was maximum number of function evaluations, which was set to 3200. By default, this value is 100*Number of parameters. This option value was changed to high static value, to ensure detailed search with low parameter amount, and to stop algorithm from optimising increasingly small amounts for larger number of parameters.

It should be noted that if parameter value search space grows too large or is unknown global search algorithm cannot be used for re-optimisation. It makes algorithm not capable of optimising parameter values anymore as, search space is too large. If that is the case global search should be swapped for genetic algorithms (GA). Furthermore, changes to default (GA) algorithm should be made as described below.

The population size should be chosen based on work done by Stanley Gotshall (Gotshall and Rylander 2002). His experiments showed effect of population size for 3 distinctive problems keeping other variables same. The results show that optimal population size for 3 parameters would be 85. This would produce lowest amount of incorrect solution while keeping number of generations at minimum.

There are several crossover functions that serves different purposes. Intermediate crossover function should be chosen because, initial population provided is already close to global minimum, which means creating child within the hypercube defined by placing the parents at opposite vertices would provide solution close to initial guess.

Mutation function should be chosen based such that it generates direction of mutation based of successful and unsuccessful generation. In addition, mutation should be kept within upper and lower constrains. Both characteristics are desired for re-optimization.

Elite count should be set to 50% of population size. This specific number is chosen for two reasons. First it is same values used in (Gotshall and Rylander) when selecting population size and second as initial population is already close to global minimum it is desired to keep most of population to stop it from migrating too far.

Function tolerance should be lowered form default value of 1e-6 to 1e-3, because during testing it was observed that GA reach place where cost function is almost flat in all direction making optimization take longer that it should without any significant improvement.

Most of options were to limit GA search around initial points instead of allowing it full range search, thus increasing efficiency and speed of algorithm. Initial population should be generated by creating random vectors of parameters with the standard deviation of 1 and mean of values that were produced by the decoupling step.

5.7. Summary

Chapter describes all algorithms used in the proposed state substitution method for parameter identification required for methods set-up and its execution. The set-up requires the user to determinate most optimal model hierarchy after decoupling, which is achieved with help of two methods, Bi-partite chart and the parameter impact correlation using SOM. Bi-partite chart provides a straightforward answer, which of the systems states affect most of the other states. This helps while setting up the hierarchy, as states with most impact should be higher solved first. Similarity the parameters that affect most of the systems states should be solved first and passed on to lower levels of hierarchy. Parameter impact correlation allows to calculate, which of the parameters have most effect on overall prediction error. Prioritizing these parameters to be solved first, increases accuracy of the state substitution method. Execution of the proposed state substitution method for parameter identification can be broken down into following steps: data generation (if needed), data approximation, local decoupled optimisation, global re-optimization. For testing of the validity and accuracy of the proposed state substitution method data of the system was generated with artificial levels of random white noise. Data approximation was performed by cubic spline, with smoothing parameter of 0.95. Local optimisation was performed by nonlinear least squared algorithm and global optimization was performed with genetic algorithm.

6. Results and discussion

This chapter presents results collected from four case studies and analyses their complexity. Three out of four models are based on simulated data, and one is based on experimental data. Case studies was selected to represent different levels of complexity. Complexity was evaluated based on methods described in chapter 4. Each case study consists of four sub sections: a description of system, complexity analysis, method comparison and summary of the results. The description of the systems explains why the system was chosen and what are the key features of this system. The complexity analysis provides details of system complexity, so it can be compared relatively to other cases studies. The method comparison shows results of the system performance while varying noise levels and sampling time of the system. The proposed State-Substitution method is compared with the derivative estimation method and Latin hyper-cube sampled multi start method. The accuracy of each method and computational time is compared and discussed~~.~~ The method comparison sub section also provides a list of the difference between identified parameters and theoretical (real) values for comparison of parameter identification accuracy.

6.1. Case study 1 – Polynomial model

This system was created as a benchmark problem, to show that the complexity of a system is not directly linked to a large number of parameters, rather, it is related to the amount of coupling within system. For this reason, a low complexity polynomial is created, consisting of twenty parameters and only one differential equation, meaning there is no coupling effects. This also means that the proposed state substitution method will not work as intended, because there no way to utilise the decoupling technique on a single ODE system. The proposed state substitution method will be applied to identify parameters of this system for completion and comparison with other methods (Derivative estimation, Multi-Start). Equation 6.1 shows the polynomial system.

$$\frac{dx}{dt} = p_1 x + p_2 x + p_3 x + \cdots + p_{20} x \tag{6.1}$$

6.1.1. Complexity analysis

To compare complexities of different systems several criteria are considered. First, a PCA visualization plot of the first principal component scores versus model error will be used to provide an overall impression of the systems error plane and will show if a global minimum is clearly visible. By marking the PCA visualization plot with convex/non-convex points we see where problematic regions are and get the convexity number in terms of a percentage. Second, each of the PCA visualization points, have scores that relate to certain set of parameters, we can optimize each of these sets and sort them out from highest final error to lowest. This will show how many local minima we can expect and how well they are defined. This will also help us to determine if the system can be optimised towards performance only or for parameter values too. Separating systems into these two groups is crucial, so in the event where a model provides accurate predictions, we know how confident of the parameter values we can be. Third, we will look at SOM analysis, we will mark the unified distance matrix plot with colours based on the dominant error value within each cluster. This will allow to visualise multiple minimum (if they exist). In addition, SOM component plots can help identify parameters that influence the system most. If any individual parameter SOM patterns agree with SOM patterns of overall error, we can infer that the system is dominated by that parameter. Complexity analysis will be performed for each case study so direct comparison of relative complexity can be made.

6.1.1.1. PCA visualization

The PCA visualization graph is constructed by decomposing the array of parameter values. These values are obtained by using the Latin hyper cube sampling technique to provide ten thousand samples. For each parameter set, the first two principal component scores are obtained. Then these scores are plotted against squared error of the difference between models with theoretical values and sample values. Looking at the figure 6.1 it shows us that it is largely flat in PCA2 axis (scores of second principal component) and has a slight curve in PCA1 axis (scores of first principal component). This makes a downward slope towards global optimal solution. Looking at graphs we can also see that all plotted points are marked blue, meaning this system is 100% convex, or in other words no matter where the initial parameters values are (within analysed boundaries) a gradient based optimisation algorithm will always tend towards the global optimum. The downward slope also means all solution are more sensitive with respect to PCA1 than PCA2, this is expected as during PCA analysis PCA1 will always hold most variability followed by PCA2, then PCA3 and so on. For this system both PCA1 and PCA2 explain 5% variability of the system for a total of 10%. Simple surface structure of this error plot and 100% convexity suggests that this is low complexity problem to optimize.

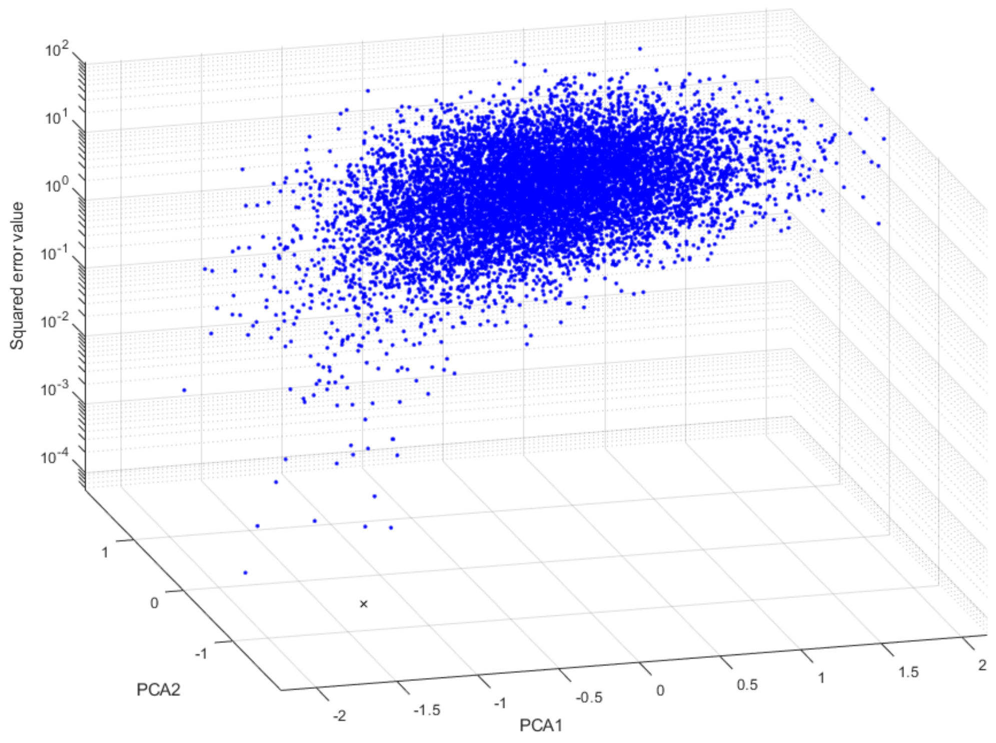


Figure 6.1 PCA error plot for Polynomial model (6.1), with colour coded points for convexity, where blue points are convex, red points are non-convex and green points are failed integrations. Black x marks global optimum solution.

6.1.1.2. Sorted minimization

Sorted optimization is performed same way as described in section 4.7. Parameter space is sampled for thousand initial parameter values. As mentioned before in chapter 4.7, for a system with 100% convexity, you would expect a horizontal straight line as all the points should converge to the global optimum, but that is not the case. This is because this polynomial system, is only solvable for performance optimum not parameter optimum. Looking at the difference between 'worst' and 'best' optimisation provides more clarity in what is happening. With 'worst' squared error being in a range of 10^{-12} and 'best' squared error being 10^{-18}. For all practical application both squared error values are same as zero. Meaning although there is a difference, and there are plenty of different local minimum within that gap, the difference between these local minimums is negligible. The reason why this system behaves like this is to do with its structure. Parameters values are interchangeable, meaning if any two parameter values would be swapped places no change in overall function (6.1) would occur. It becomes very clear if we simplify this system to two parameter system such as 10 = x + y. The combination of pairs of x + y values which can sum to ten is infinite. Each of those pair would satisfy system performance criteria i.e. the solution outcome is correct, but the actual values can vary. This shows us that this polynomial model can be easily solved to perform well, but it is true parameter values, cannot be identified.

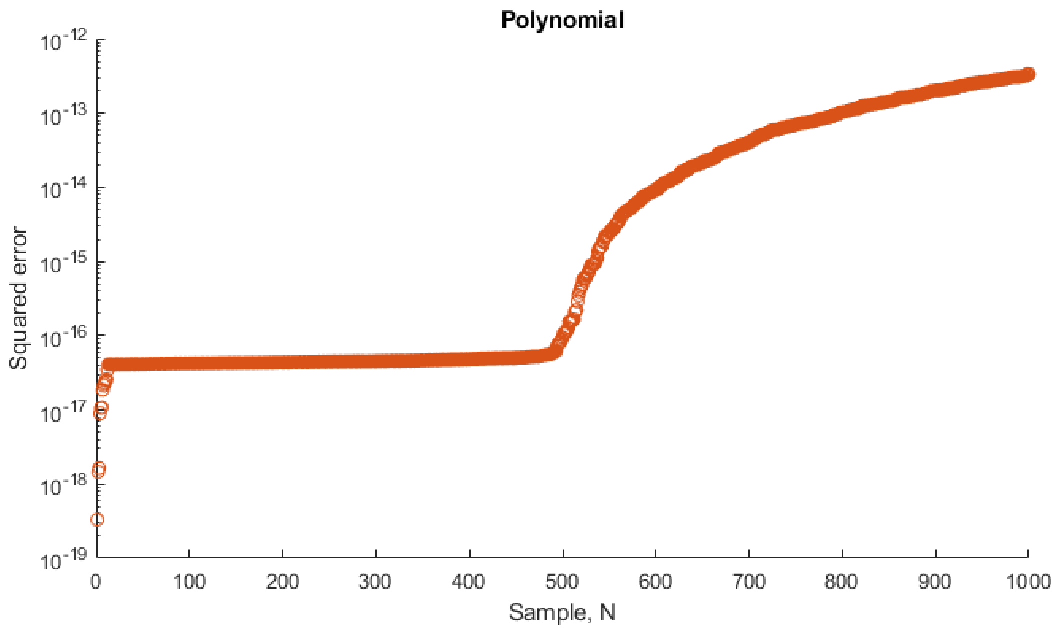


Figure 6.2 Polynomial model (6.1) thousand samples sorted optimization, where blue is all samples and red are convex samples only (As this model is 100% convex all samples and convex samples is same).

6.1.1.3. SOM analysis

As with the PCA visualization, the Latin Hyper cube sampling technique was used to obtain an array of parameters. Each sampled array of parameters is treated as a sample and they are sorted and coloured by the error between the modelled solution and the true solution. Errors are classified into high medium low and the most abundant solutions in a particular space are the dominant colour on the map. Within analysed boundaries, error values above 50% of maximum error where considered to be high error values(red). Error values between 50% and 5% of maximum error where considered to be medium error values(blue), and error values below 5% of maximum error where considered to be low error values(green). The same colour coding is consistent within all SOM analysis in this thesis.

Looking at figure 6.3 results of SOM, supports previous conclusions that system is easily solvable, but has multiple local minima. All high error values are concentrated in the centre, creating one big local minimum around it. Lowest error clusters are scattered showing you can reach similar performance with completely different set of parameters. Looking at figure 3b, each hex has multiple colours, these colours represent values that were assigned to that clusters, bigger the colour more of it was within that hex. Grey hexes around it provides value of relative distance between each cluster of coloured hexes. This further supports the multiple minima conclusion, scattered around the plane. It should be noted that none of the green values are found in the middle cluster of high error values.

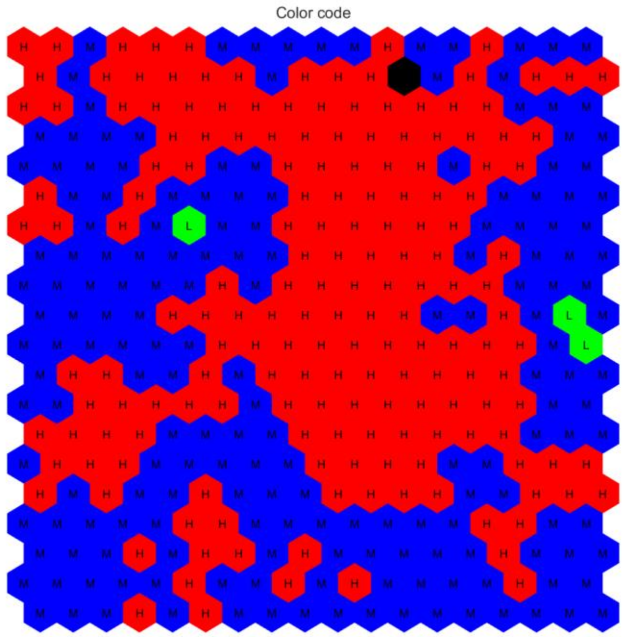
Although from the structure of the polynomial we know that none of the parameters are dominant, it is not always obvious with more complicated systems. To be able to see if how strongly parameters effect the system, we can utilise the SOM component maps and colour coded SOM. We cross correlate the patterns of component maps to overall error map to see how well they match. Matching component map and overall error map implies there is a dominant connection between the parameter and the system. Although correlation does not mean causation, with large enough sample size it is very unlikely to run into false positives. Furthermore cross-correlation need to be done twice, as parameters can be positively or negatively correlated. Therefore, a second cross-correlation is done to inverse error map.

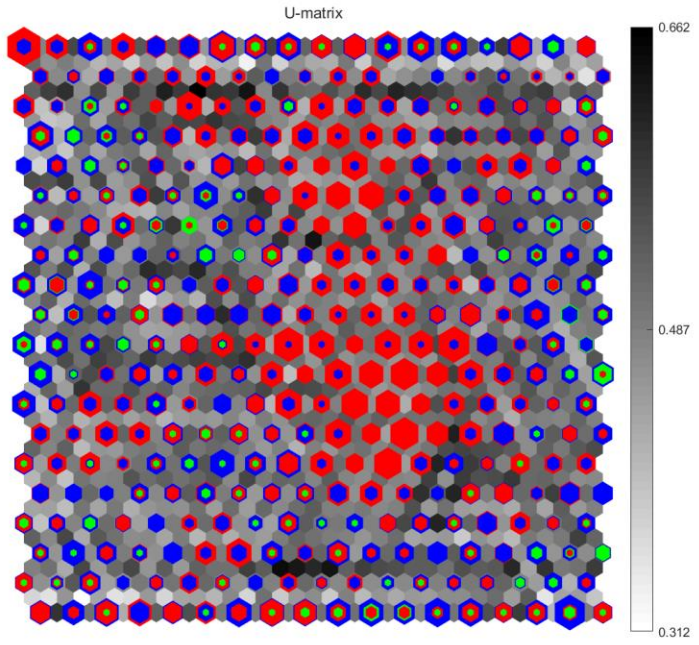
Figures 6.5 - 6.6 shows how well component maps match up versus total error maps. Positive values mean higher degree of matching, and lower values mean lower. Cross correlation factor of 1, represents perfect match and 0 represent mismatch. As expected, none of the parameters seems to be more dominant than others. These figures allow to see how well, parameter changes match overall error changes in the system and to represent a quantifiable measurement of how dominant the parameter an overall matching factor was calculated (Table 6.1).

Parameter	**Positive Matching Factor**	**Negative Matching Factor**
x1	0.66 ± 0.03	0.62 ± 0.04
x2	0.69 ± 0.02	0.60 ± 0.03
x3	0.66 ± 0.02	0.63 ± 0.03
x4	0.68 ± 0.02	0.60 ± 0.05
x5	0.72 ± 0.03	0.59 ± 0.04
x6	0.71 ± 0.02	0.58 ± 0.04
x7	0.68 ± 0.05	0.63 ± 0.03
x8	0.68 ± 0.04	0.62 ± 0.03
x9	0.69 ± 0.01	0.59 ± 0.06
x10	0.68 ± 0.05	0.60 ± 0.04
x11	0.68 ± 0.03	0.62 ± 0.03
x12	0.67 ± 0.05	0.64 ± 0.05
x13	0.72 ± 0.04	0.57 ± 0.01
x14	0.67 ± 0.03	0.62 ± 0.02
x15	0.70 ± 0.05	0.60 ±0.05
x16	0.71 ± 0.03	0.59 ± 0.02
x17	0.69 ± 0.05	0.60 ± 0.04
x18	0.67 ± 0.03	0.63 ± 0.03
x19	0.73 ± 0.02	0.58 ± 0.01
x20	0.68 ± 0.04	0.62 ± 0.04

Table 6-1 Table of positive and negative mean matching factor of each parameter for system (6.1)

We also can look at biggest difference between matching factors in table 6.1. Those are 0.068 and 0.065, respectively. This shows there is very little difference in influence between parameters confirming that none of the parameters are dominant.





SOM 10-Oct-2019

Figure 6.3 a) 2d SOM of Polynomial model (1), colour coded based of model error, where green (L) is low error, blue (M) is medium error, and red (H) is high error. b) 2d SOM with same colour code as a), but also showing relative distance in n-dimensional plane represented as colour bar and separation of different colour within each cluster.

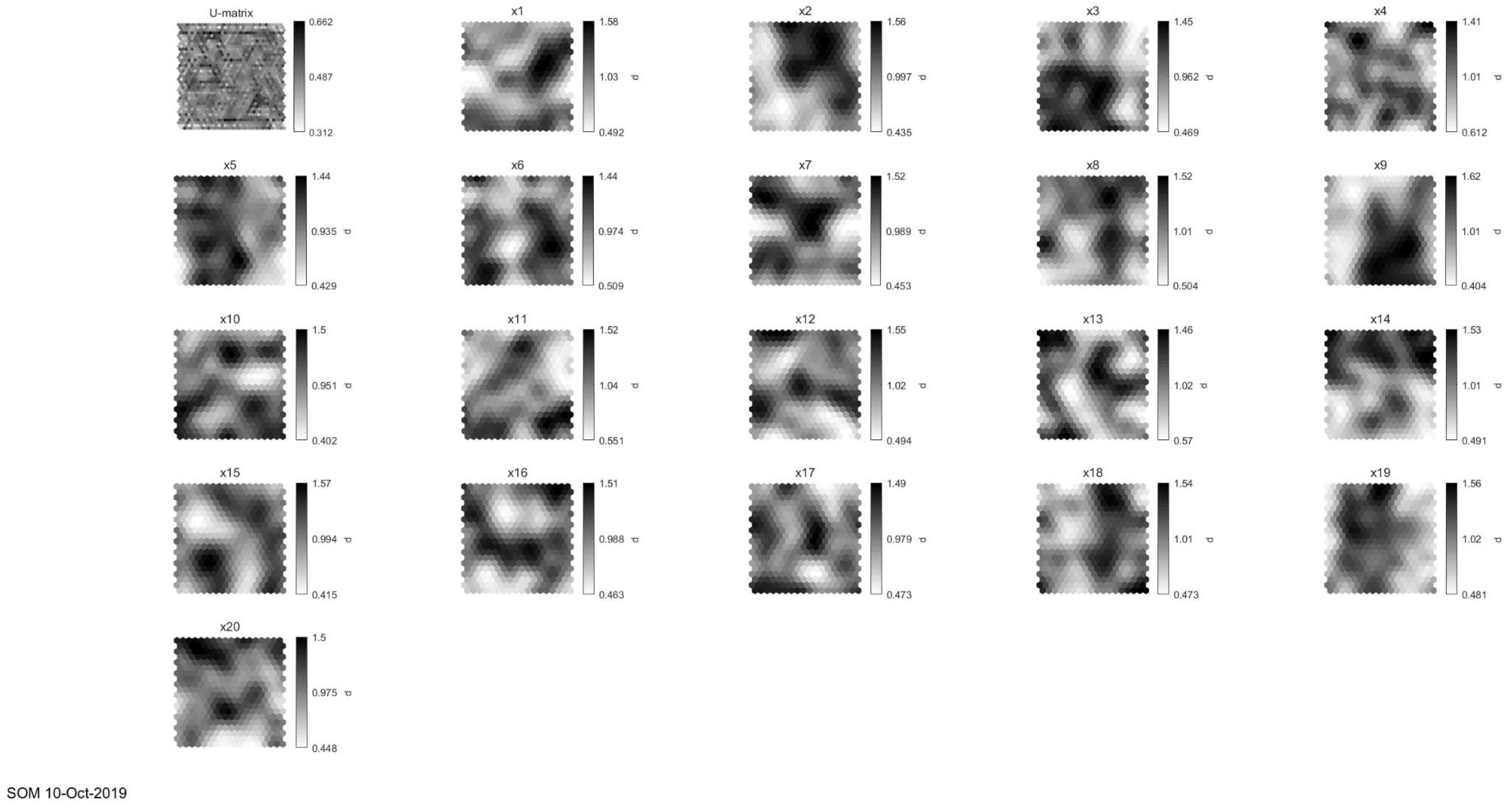


Figure 6.4 From top left: 1) n-dimensional plane represented in 2d SOM with relative distances as colour bar. 2) - 21) x1 to x20 parameters with their respective variation within n-dimensional plane, parameter values denoted in colour bar.

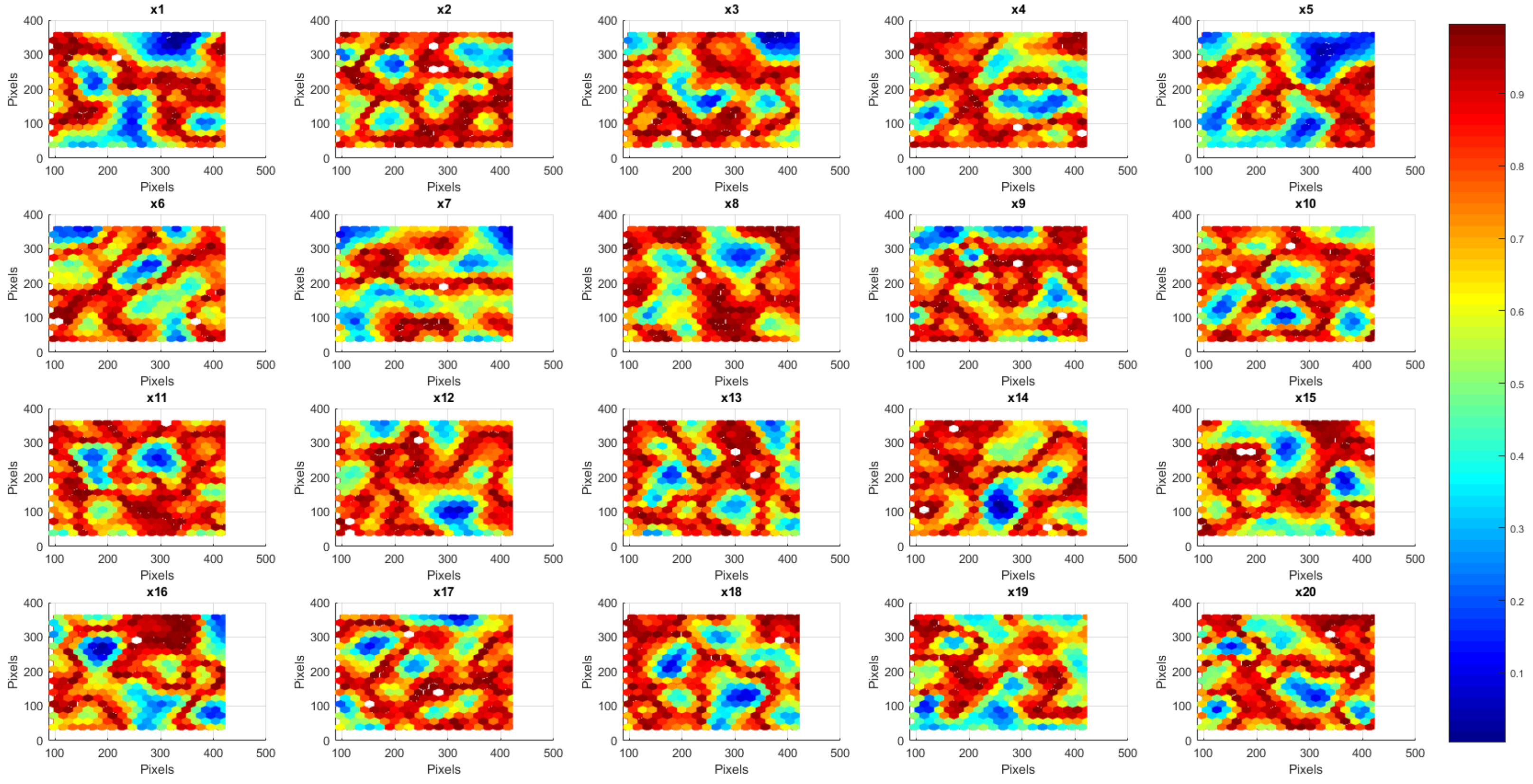


Figure 6.5 Positive cross-correlation between each parameter map and overall error map.

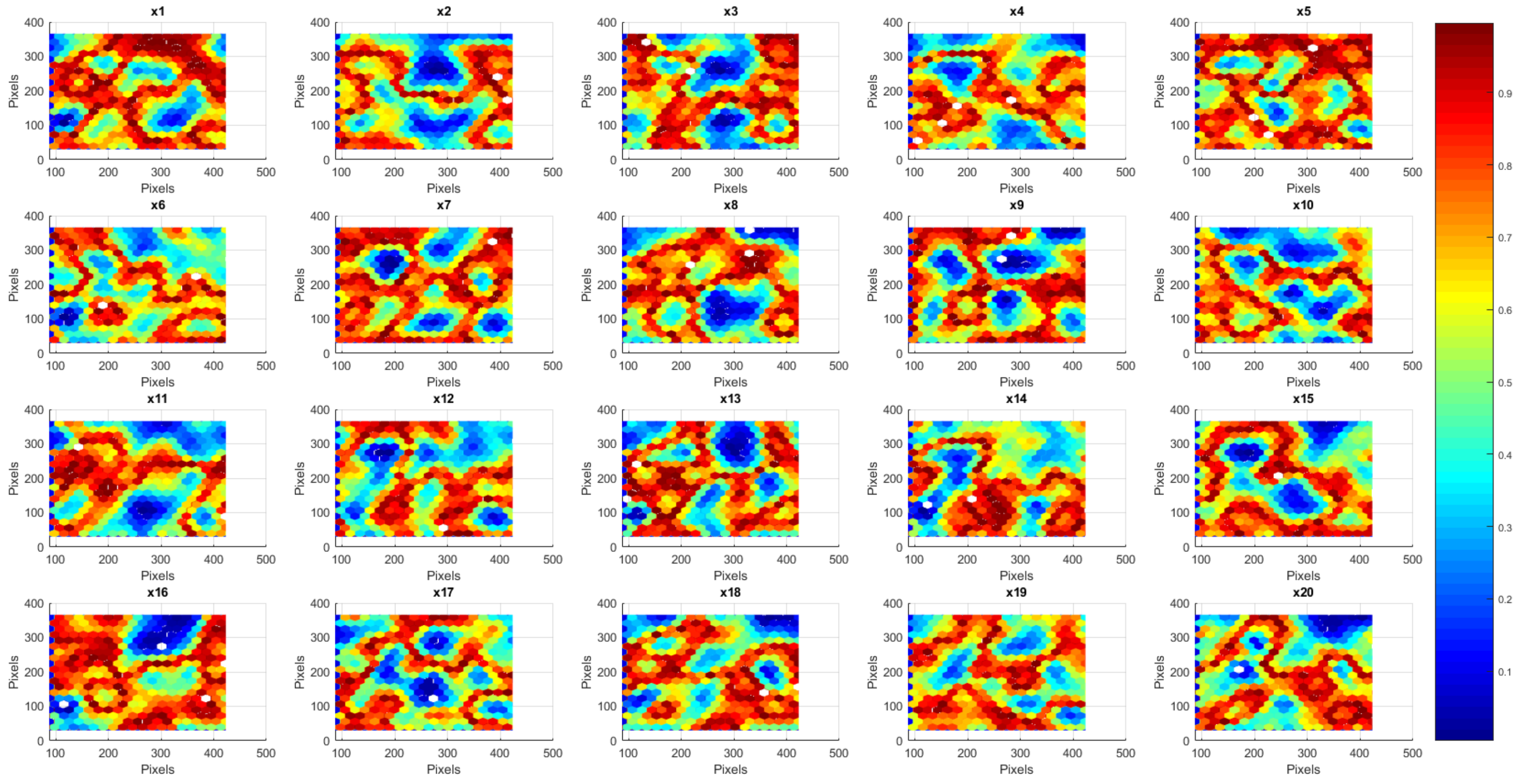


Figure 6.6 Negative cross-correlation between each parameter map and overall error map.

6.1.2. Method comparison

To evaluate the proposed state substitution method, it will be compared to the two existing methods discussed in chapter 3. The derivative estimation and Latin-hyper cube sampled multi start method. Two main criteria will be compared, model performance accuracy and the computational time between each of the methods. Although the proposed state substitution method aims to reduce computational time, model performance accuracy is also very important and cannot be completely neglected. Each method will be assessed with different random noise levels and sampling times. For each method three different state initial conditions are compared, first two (Pink and Blue) are set same for all methods and are conditions that were provided for the optimisation algorithm and third initial condition (Black) is randomised between first two initial conditions and was never seen by algorithm before. This allows to check method accuracy with unseen data sets, which are within same boundary conditions. Modelling conditions and parameter search space are summarised in tables below:

Experiment number	**Sampling Rate**	**Noise level**	**Initial conditions Pink**	**Initial conditions Blue**	**Initial conditions Black**
1	0.1h	5%	0, 0, 0	30, 30, 30	16.9, 7.1, 14.9
2	0.1h	10%	0, 0, 0	30, 30, 30	15.4, 15.6, 24.4
3	0.3h	5%	0, 0, 0	30, 30, 30	23.1, 20.6, 4.9
4	0.3h	10%	0, 0, 0	30, 30, 30	26.6, 18.0, 18.9
5	0.3h	20%	0, 0, 0	30, 30, 30	22.5, 18.1, 27.0

Table 6-2 Summary of modelling conditions for system (6.1)

Parameters	**Lower bound**	**Upper bound**
x1, x2, x3…. x20	0	2

Table 6-3 Parameter search space for system (6.1)

6.1.2.1. Experiment 1 0.1h sampling and 5% random noise

While comparing the accuracy of the model, we do not see much of a difference between each method. Furthermore, the sampling rate is high and noise level is low, allowing multiple approaches to perform accurately (Fig 6.7), without getting stuck in local minima. This result is expected. Since accuracy of model predictions are high (table 6.4) and very similar between different methods, then we should investigate computational time as the next important criteria.

Method	**Computational time, s**
Derivative estimation	223
Latin hyper cube multi-start	93
Proposed state substitution method	181

Table 6-4 Computational time of all three method for experiment 1.

Method	**Sq. Error (Blue)**	**Sq. Error (Pink)**	**Sq. Error (Black)**
Derivative estimation	2.61E-05	3.38E-05	3.20E-05
Latin hyper cube multi-start	2.39E-04	3.08E-04	3.00E-04
Proposed state substitution method	1.19E-04	1.55E-04	1.46E-04

Table 6-5 Squared error values of each method and each data set for experiment 1

When comparing computational times, it is easy to see that multi-start method is much faster, than the other two methods (table 6.3). As the system is 100% convex it works in the favour of the multi-start method as computational time is very sensitive to the convexity of the problem. Starting position needs to be in right valley to able to optimise towards global optimum. This leads to requiring a smaller size of Latin hyper cube, thus reducing computation time.

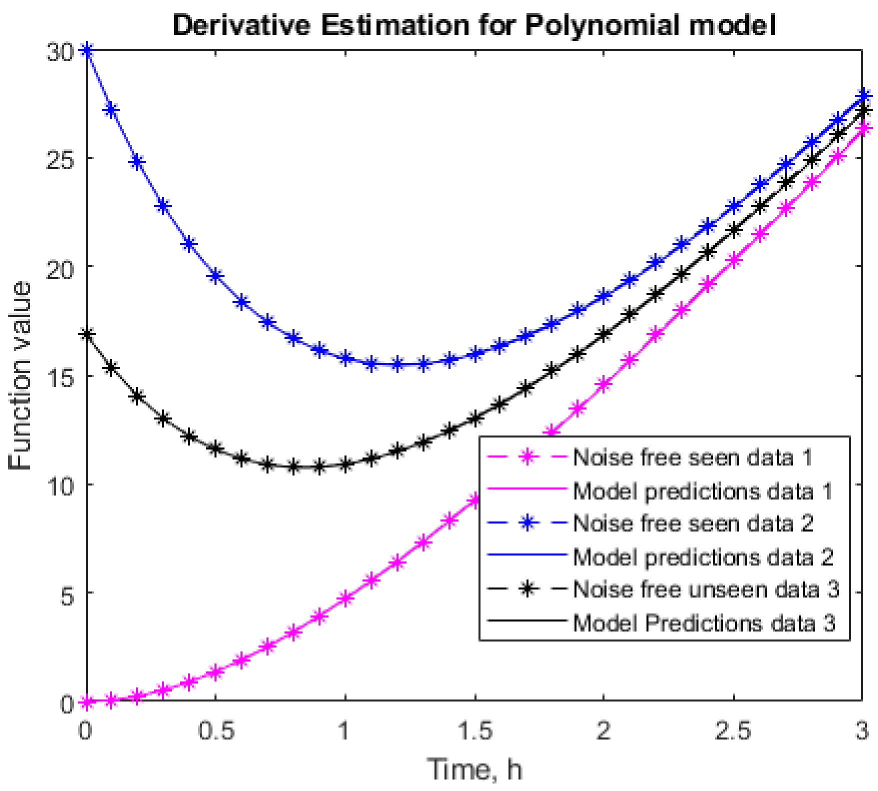
The multi-start method is faster than Derivative estimation method, because with a low complexity problem it does not require a large number samples to be able to optimise towards the correct solution, and screening for best starting points reduces the number of optimizations that need to be done. Where the Derivative estimation method solves the lower complexity algebraic equations, but the absolute number of equations significantly increases the computation time.

It is also faster than the proposed state substitution method, as it cannot take advantage of decoupling of the system as there is only one ODE. In principle the method still works, but it does additional steps that do not have any benefit for such a lower complexity system, but still required significant computational time to be calculated. It is also important to compare parameter values, that were identified with theoretical values. Table summarizing parameter values can be seen below:

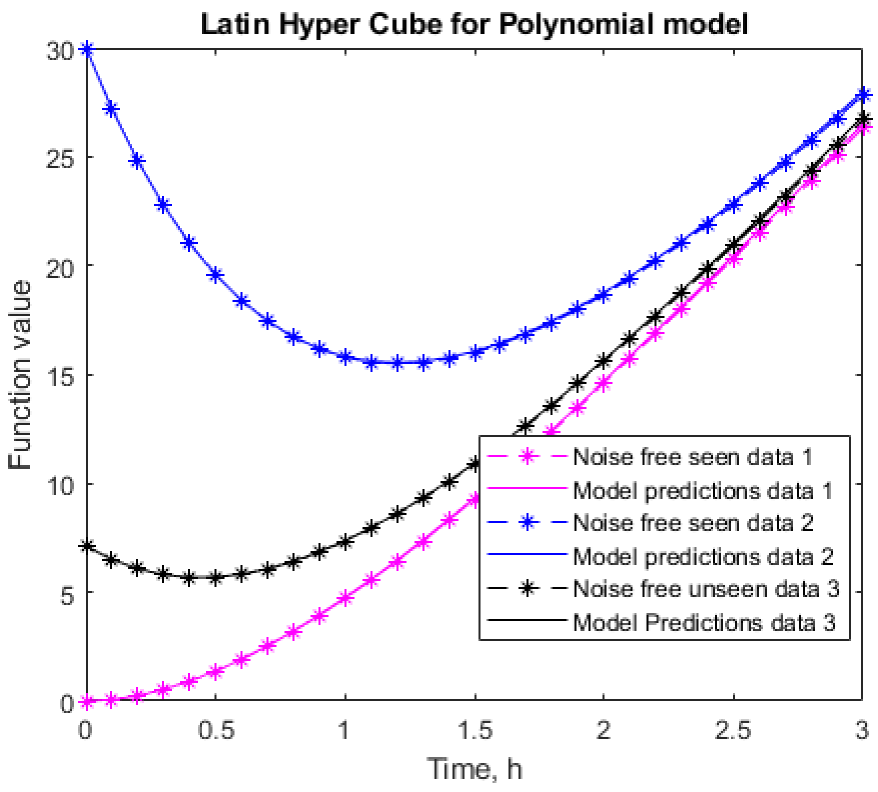
Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
x1	0.81	0.11	0.39	0.73
x2	0.91	0.68	0.92	1.28
x3	0.13	0.62	0.41	0.23
x4	0.91	0.65	0.11	0.9
x5	0.63	0.97	1.85	1.51
x6	0.1	0.05	0.56	0.75
x7	0.28	0.41	0.43	0.21
x8	0.55	0.52	0.37	0.66
x9	0.96	0.65	0.06	0.6
x10	0.96	1.72	0.42	0.92
x11	0.16	0.86	1.73	1.16
x12	0.97	1.97	0.55	0
x13	0.96	0	0.39	1.12
x14	0.49	1.23	0.84	0.15
x15	0.8	0.12	1.45	0.64
x16	0.14	0.04	0.45	0.46
x17	0.42	1.8	1.32	0.41
x18	0.92	0	0.11	0.38
x19	0.79	0.15	0.11	0.44
x20	0.96	0.25	0.46	0.34

Table 6-6 Summary of identified parameter values for each method for experiment 1

Comparing parameter values, we observe the hypothesised behaviour, based on sorted minimization in chapter 5.1.1.2. No exact parameter values match due to the infinite number of combinations that produce same result.



Derivative Estimation for Polynomial model
Function value
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3



Latin Hyper Cube for Polynomial model
Function value
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3

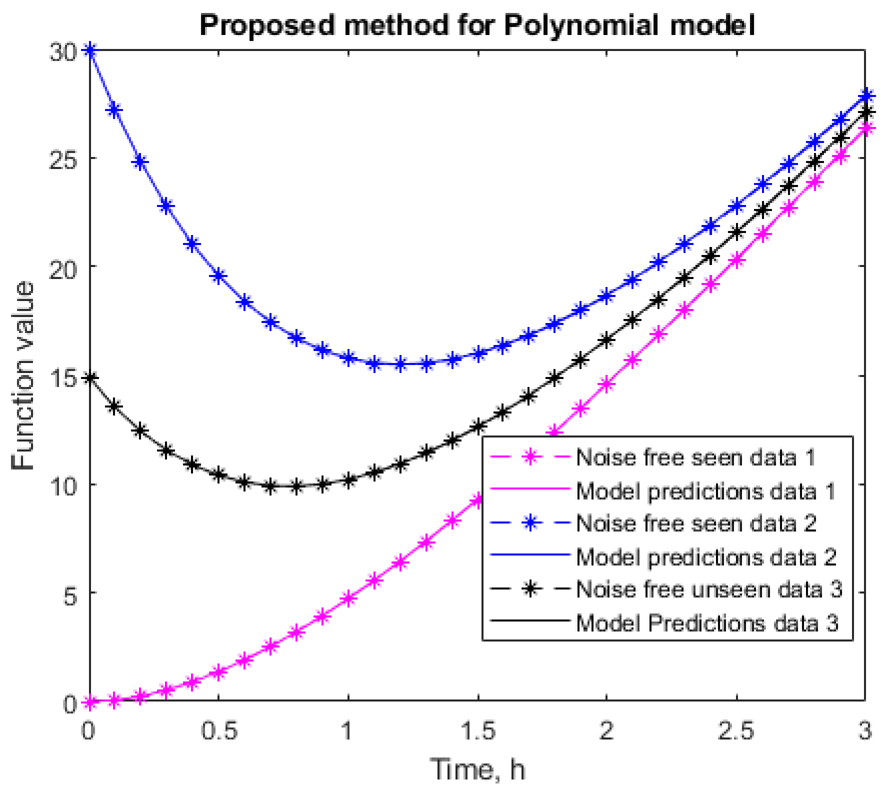


Figure 6.7 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.1h sampling and 5% random noise for measured data.

6.1.2.2. Experiment 2 0.1h sampling and 10% random noise

Even with increased noise from 5% to 10%, from previous setup, accuracy of model does not suffer in any of the methods. This is for the same reasons as explained in chapter 6.1.2.1. Problem is not complex enough to affect accuracy of the methods. It should be noted that total squared error (SSE) decreased, for the proposed and the Latin hyper cube methods, and increased for the derivative estimation method. This changed is due to methods ability to cope with noise. Although these changes are observable, they are not significant on a model scale (figure 6.8).

Method	Computational time, s
Derivative estimation	243
Latin hyper cube multi-start	107
Proposed state substitution method	180

Table 6-7 Computational time of all three method for experiment 2

Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	2.28E-04	2.98E-04	2.72E-04
Latin hyper cube multi-start	1.19E-05	1.55E-05	1.46E-05
Proposed state substitution method	1.10E-06	1.43E-06	1.35E-06

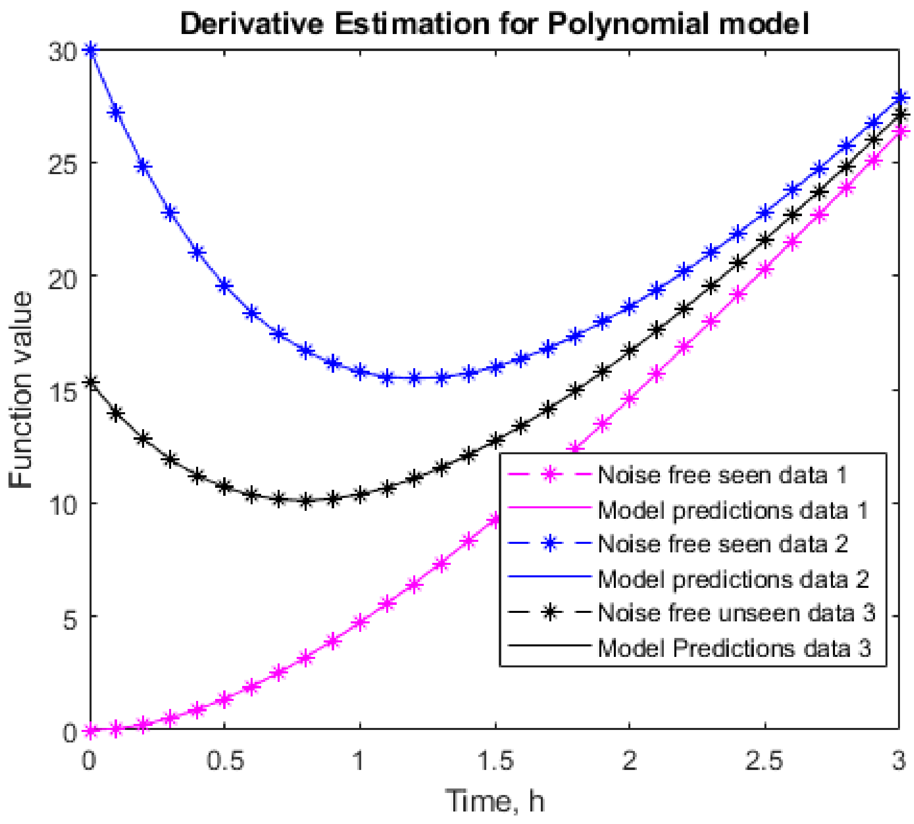
Table 6-8 Squared error values of each method and each data set for experiment 2

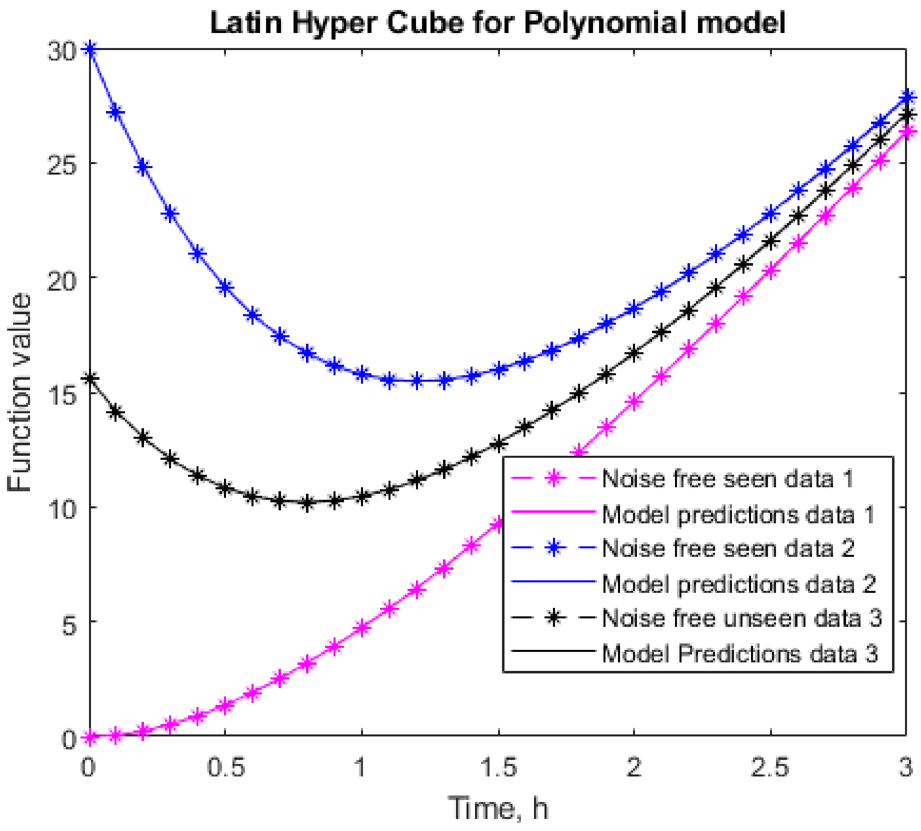
When comparing computational times there are some differences with previous setup. Firstly, all methods computational time increased. This is due to larger magnitude noise, therefore increasing uncertainty of method, as error surface becomes more disturbed. This makes each integration for each method harder. Table summarizing parameter values can be seen below:

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
x1	0.81	0.76	0.15	0.65
x2	0.91	0.24	1.24	0.71
x3	0.13	1.83	0.76	0.69
x4	0.91	0.69	0.25	0.55
x5	0.63	0.04	0.31	0.61
x6	0.1	1.29	0.03	0.74
x7	0.28	0.03	0.23	0.67
x8	0.55	0.03	1.09	0.56
x9	0.96	0.33	0.25	0.77
x10	0.96	1.05	0.95	0.55
x11	0.16	0.62	0.62	0.59
x12	0.97	0.31	1.16	0.72
x13	0.96	0.05	0.84	0.55
x14	0.49	0.31	1.81	0.57
x15	0.8	0.13	1.05	0.57
x16	0.14	0.13	0.08	0.55
x17	0.42	1.76	1.16	0.69
x18	0.92	1.04	0.65	0.73
x19	0.79	0.8	0.13	0.59
x20	0.96	1.4	0.11	0.71

Table 6-9 Summary of identified parameter values for each method for experiment 2

As before we see parameter values that are completely different than in previous setup, but still producing good performance models, due to nature of the system.


Derivative Estimation for Polynomial model
Function value
Time, h
30
25
20
15
10
5
0
0
0.5
1
1.5
2
2.5
3
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Latin Hyper Cube for Polynomial model
Function value
Time, h
30
25
20
15
10
5
0
0
0.5
1
1.5
2
2.5
3
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3

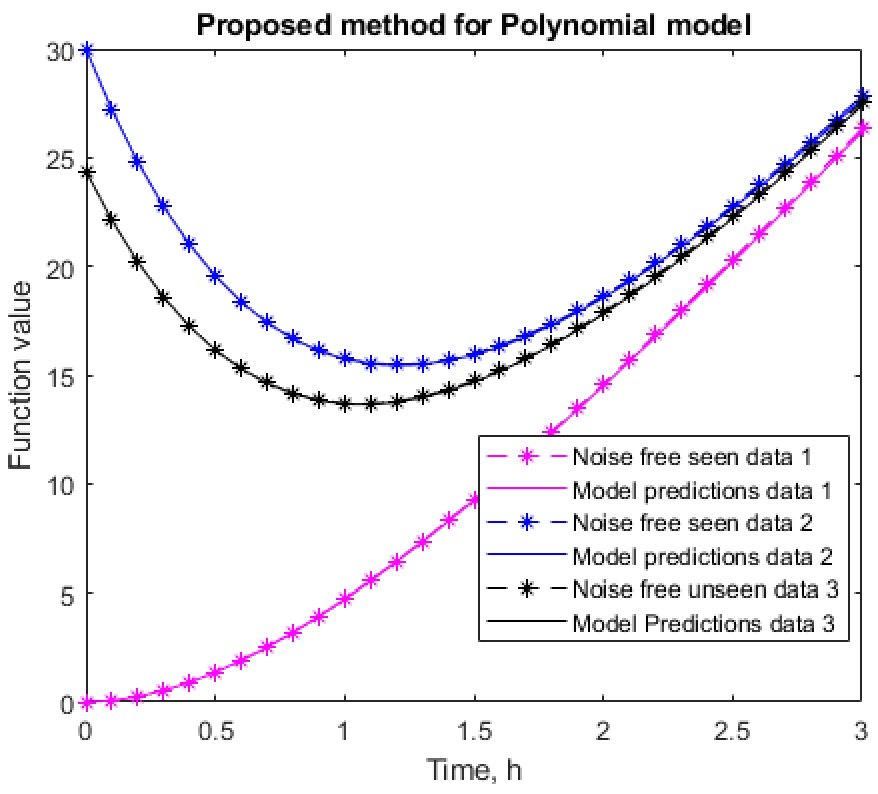


Figure 6.8 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.1h sampling and 10% noise for measured data.

6.1.2.3. Experiment 3 0.3h sampling and 5% random noise

Increasing sampling time from 0.1h to 0.3h, seem to effect derivate estimation method and proposed state substitution method in this case, while Latin hyper cube method performs at the same accuracy. Lower number of sample points provide less information for each method to optimise correctly towards solution. Still even with some inaccuracies all methods can find solution that produces acceptable performance. The Latin hyper cube method seems to outperform other methods in terms of accuracy (table 6.10).

Method	**Computational time, s**
Derivative estimation	190
Latin hyper cube multi-start	98
Proposed state substitution method	140

Table 6-10 Computational time of all three method for experiment 3.

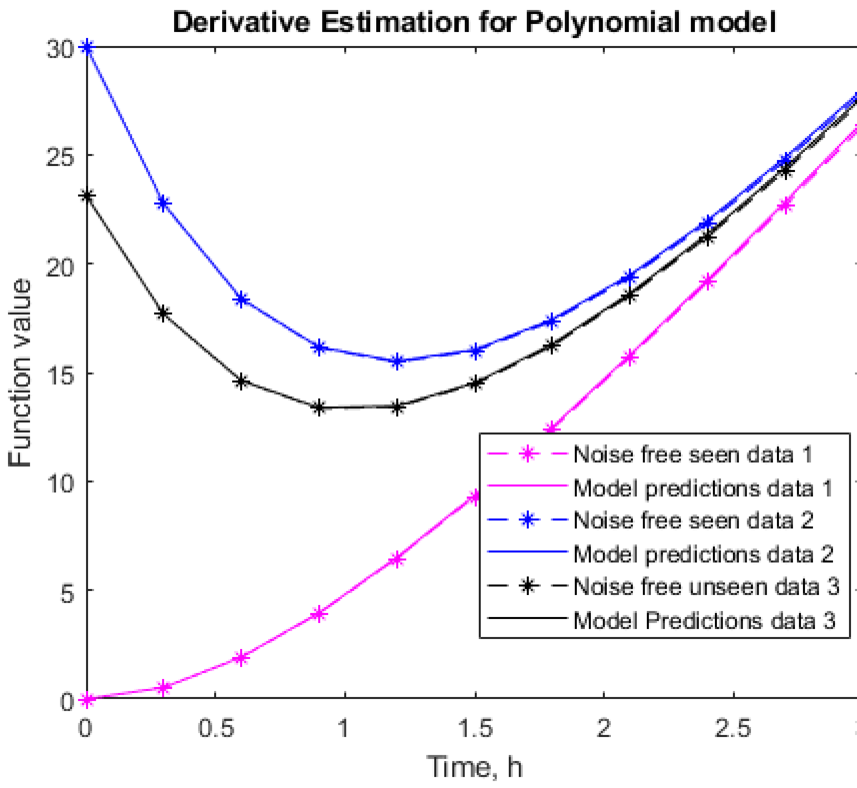
Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	3.23E-04	4.24E-04	4.16E-04
Latin hyper cube multi-start	3.42E-07	4.44E-07	4.11E-07
Proposed state substitution method	1.04E-04	1.34E-04	1.23E-04

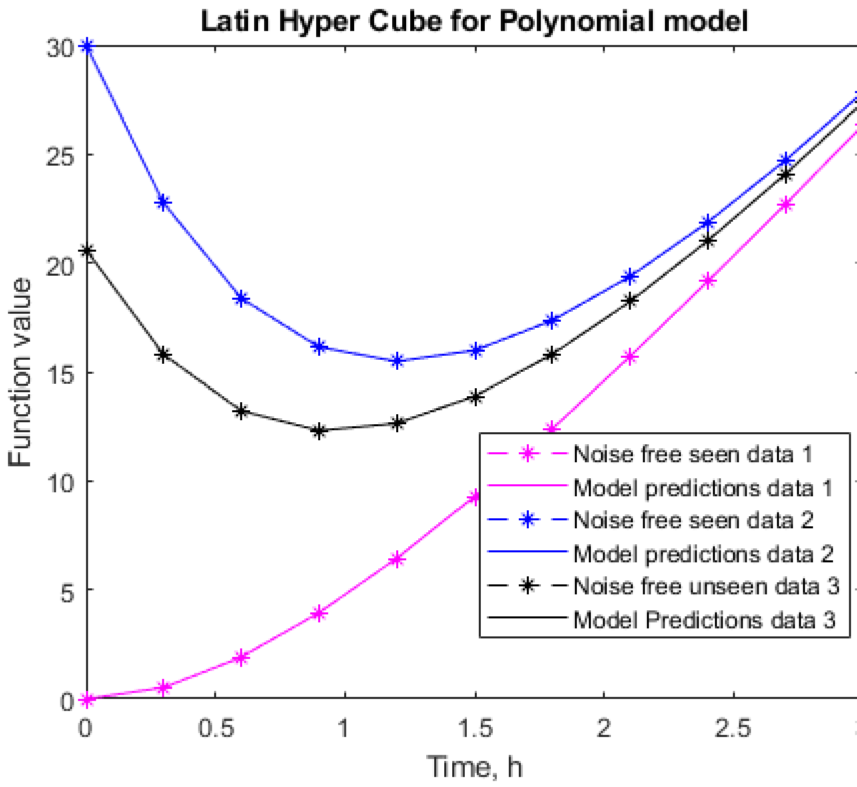
Table 6-11 Squared error values of each method and each data set for experiment 3

When comparing computational time, we can see that the overall trend stays the same of multi-start method being best followed by proposed and derivate estimation methods. Furthermore, all computational times dropped compared to experiments 1 and 2. That is because with decreased sampling frequency, there is less data to compute, making methods faster to arrive at a solution, at the cost of accuracy. The Multi-start method seems to be unaffected by this, keeping its computational time relatively similar, but at the same time it does not suffer any accuracy penalties either. Table summarising parameter values can be seen below:

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
x1	0.81	0.02	0.08	0.61
x2	0.91	0.5	1.49	0.98
x3	0.13	0.49	0.62	0.63
x4	0.91	0.37	0.41	1.39
x5	0.63	1.89	0.08	0
x6	0.1	0.19	0.76	1.11
x7	0.28	1.91	1.76	1.09
x8	0.55	0.18	0.01	0.45
x9	0.96	0.48	0.84	0.7
x10	0.96	0.21	0.9	0.59
x11	0.16	0.11	0.74	0.7
x12	0.97	1.28	0.49	0.58
x13	0.96	0.16	0.1	0.81
x14	0.49	0.33	0.04	0.8
x15	0.8	1.16	0.88	0.65
x16	0.14	0.9	1.24	0.4
x17	0.42	0.24	0.79	0.49
x18	0.92	0.39	0.67	0.27
x19	0.79	0.9	0.66	0.08
x20	0.96	1.24	1.84	0.35

Table 6-12 Summary of identified parameter values for each method for experiment 3.


Derivative Estimation for Polynomial model
Function value
Time, h
0
5
10
15
20
25
30
0
0.5
1
1.5
2
2.5
3
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Latin Hyper Cube for Polynomial model
Function value
Time, h
0
5
10
15
20
25
30
0
0.5
1
1.5
2
2.5
3
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3

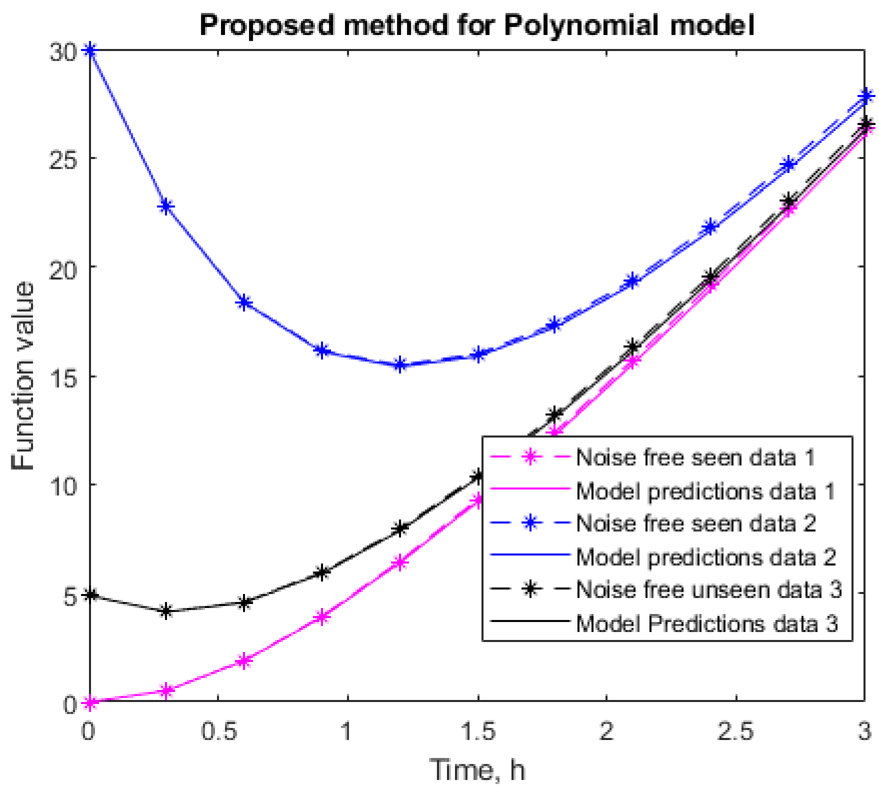


Figure 6.9 Performance results for Derivate estimation, Latin hyper cube and proposed state substitution methods, with 0.3h sampling and 5% noise for measured data.

6.1.2.4. Experiment 4 0.3h sampling and 10% random noise

With increased noise from 5% to 10%, all methods seem to suffer in accuracy. Errors are small and all methods produce acceptable performance of a model. All methods seem to perform on similar accuracy levels.

Method	**Computational time, s**
Derivate estimation	204
Latin hyper cube multi-start	99
Proposed state substitution method	176

Table 6-13 Computational time of all three method for experiment 4.

Method	**Sq. Error (Blue)**	**Sq. Error (Pink)**	**Sq. Error (Black)**
Derivative estimation	1.34E-04	1.73E-04	1.61E-04
Latin hyper cube multi-start	1.27E-04	1.65E-04	1.55E-04
Proposed state substitution method	2.39E-04	3.13E-04	2.84E-04

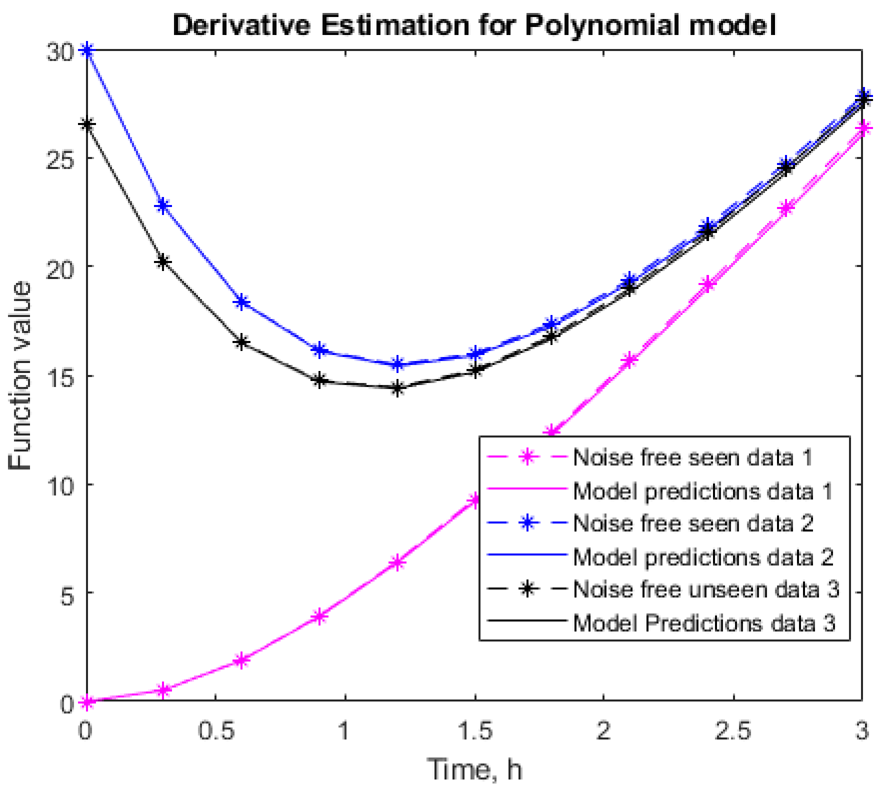
Table 6-14 Squared error values of each method and each data set for experiment 4

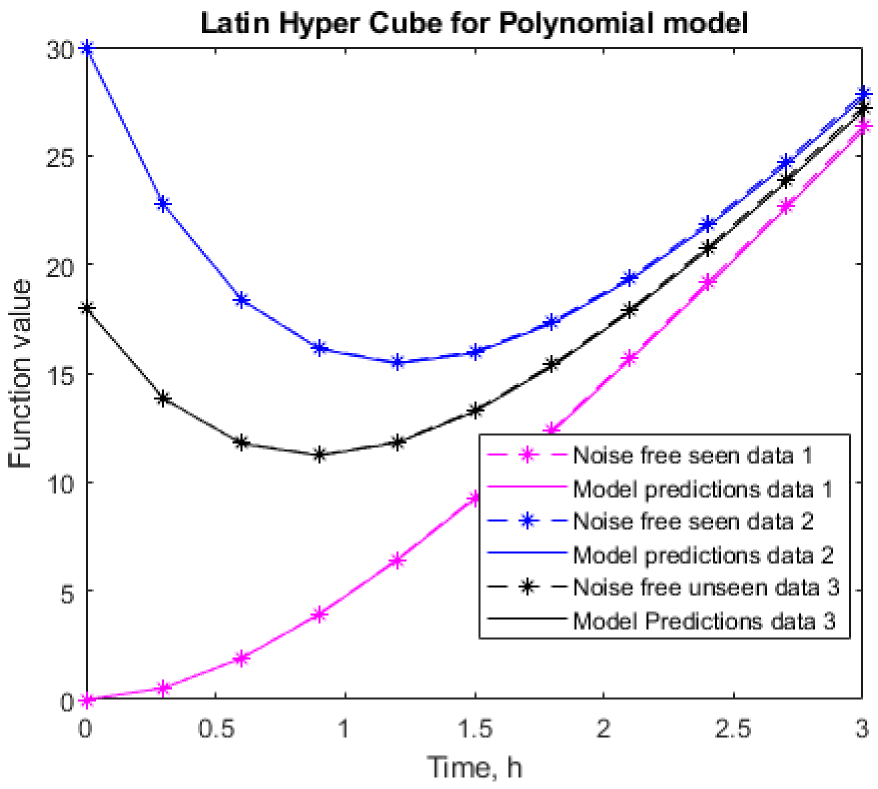
We can also see that methods accuracy degrade more than with experiment 3, from their computational time all methods, with exception of Latin hyper cube model, have significant increase in computational time. Table summarizing parameter values can be seen below:

Parameter	Theoretical value	Derivate	Latin hyper cube	Proposed
x1	0.81	1.76	0.07	0.61
x2	0.91	1.25	1.08	0.79
x3	0.13	0.21	1.69	1.04
x4	0.91	0.18	0.53	0.55
x5	0.63	0.25	0.48	0.5
x6	0.1	0.63	0.9	0.67
x7	0.28	0.13	1.25	0.43
x8	0.55	0.59	1.41	0.56
x9	0.96	0.21	0.4	0.61
x10	0.96	1.35	0.14	0.89
x11	0.16	0.35	1.3	0.46
x12	0.97	0.12	0.03	0.87
x13	0.96	1.24	0.47	0.66
x14	0.49	0.44	0.52	0.78
x15	0.8	1.29	0.92	0.62
x16	0.14	0.39	0.56	0.63
x17	0.42	1	0.24	0.48
x18	0.92	0.15	0.25	0.3
x19	0.79	0.18	0.2	0.92
x20	0.96	1	0.31	0.57

Table 6-15 Summary of identified parameter values for each method for experiment 4.

As before we see parameter value that are completely different than in previous experiments, but still producing good performance models, due to nature of the system.


Derivative Estimation for Polynomial model
Function value
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Latin Hyper Cube for Polynomial model
Function value
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3

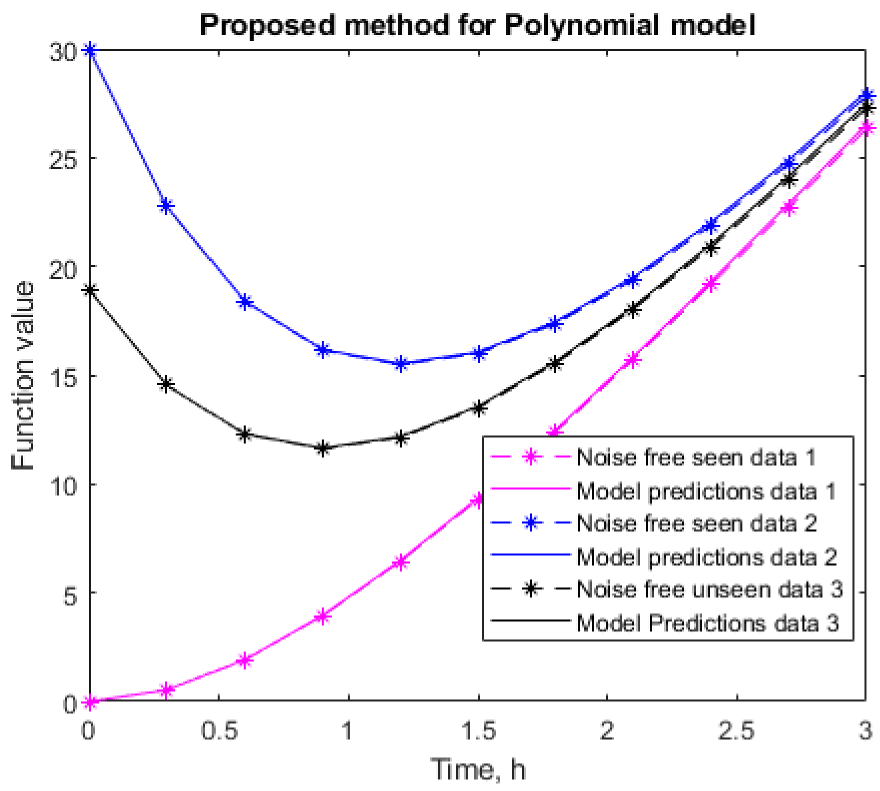


Figure 6.10 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.3h sampling and 10% noise for measured data.

6.1.2.5. Experiment 5 0.3h sampling and 20% random noise

At noise levels of 20% we start to see Further degradation in the performance of the methods. From figure 6.11, we can say that the derivate method and the proposed state substitution method have worst accuracies and the Latin hyper cube method having better accuracy. At 20% noise level a lot of parameter influence on system is hidden by large noise. Reason why methods can still be accurate, at least for the first half of the model, is because system is so forgiving with parameter selection. Making it very simple problem to identify to perform well.

Method	**Computational time, s**
Derivate estimation	256
Latin hyper cube multi-start	141
Proposed state substitution method	236

Table 6-16 Computational time of all three method for experiment 4

Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	1.60E-03	2.13E-03	1.93E-03
Latin hyper cube multi-start	3.51E-04	4.62E-04	4.31E-04
Proposed state substitution method	3.40E-03	4.25E-03	3.92E-03

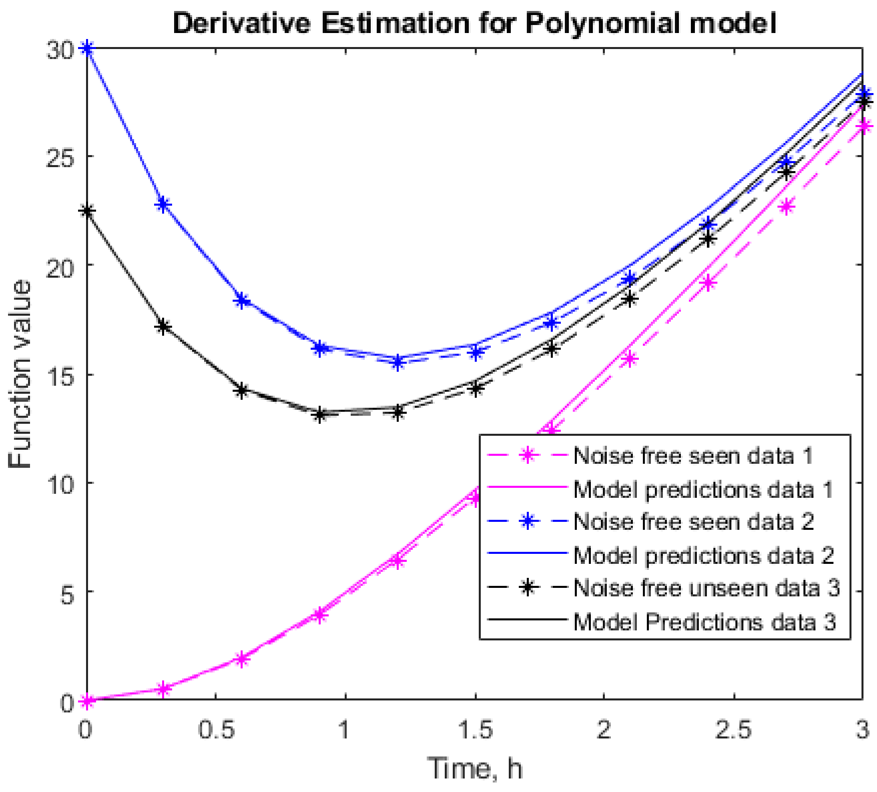
Table 6-17 Squared error values of each method and each data set for experiment 4

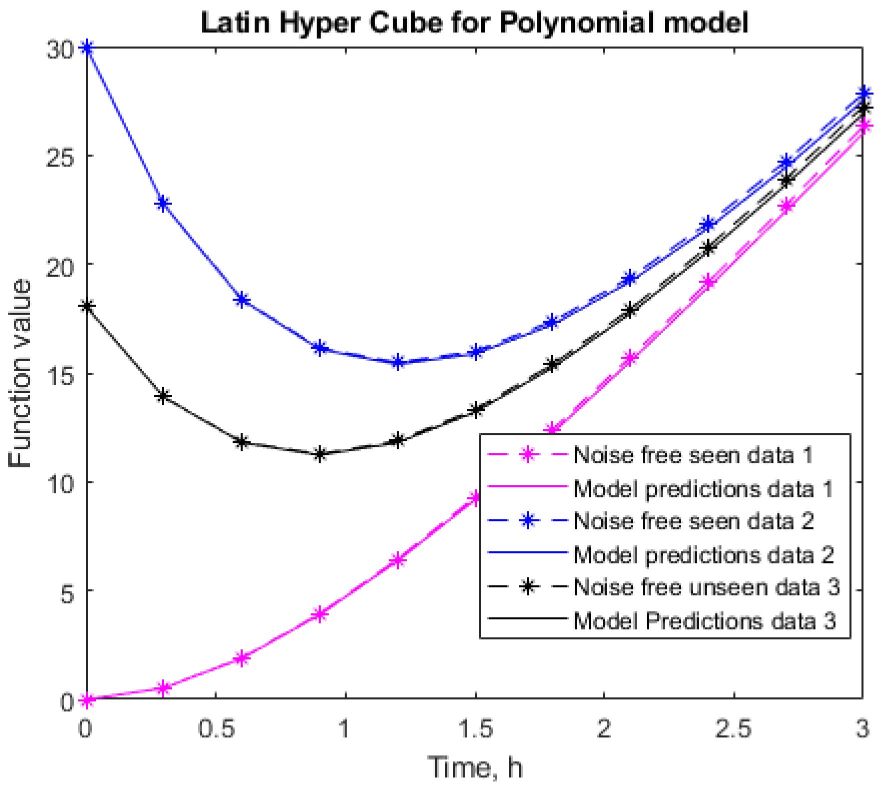
Computational time follows same trend as before. Increase in noise level increase computational time across all methods, keeping them in same order from fastest to slowest. Table summarizing parameter values can be seen below:

Parameter	Theoretical value	Derivate	Latin hyper cube	Proposed
x1	0.81	0.78	0.77	0.61
x2	0.91	0.63	0.28	0.7
x3	0.13	1.06	0.17	0.9
x4	0.91	0.53	0.68	0.16
x5	0.63	1.29	0.16	1.45
x6	0.1	0.41	1.55	0.7
x7	0.28	0.37	0.97	0.7
x8	0.55	0.98	0.31	0.03
x9	0.96	0.77	0.18	0.11
x10	0.96	0.7	0.15	0.96
x11	0.16	0.73	0.38	0.63
x12	0.97	0.83	0.22	0.68
x13	0.96	0.18	1.57	0.81
x14	0.49	0.53	0.14	0.6
x15	0.8	0.48	1.01	0.18
x16	0.14	0.31	0.58	0.62
x17	0.42	1	0.52	0.84
x18	0.92	0.19	0.83	0.73
x19	0.79	0.49	1.38	0.5
x20	0.96	1.07	0.84	0.61

Table 6-18 Summary of identified parameter values for each method for experiment 4.

As before we see parameter value that are completely different than in previous experiments, but still producing good performance models, due to nature of the system.





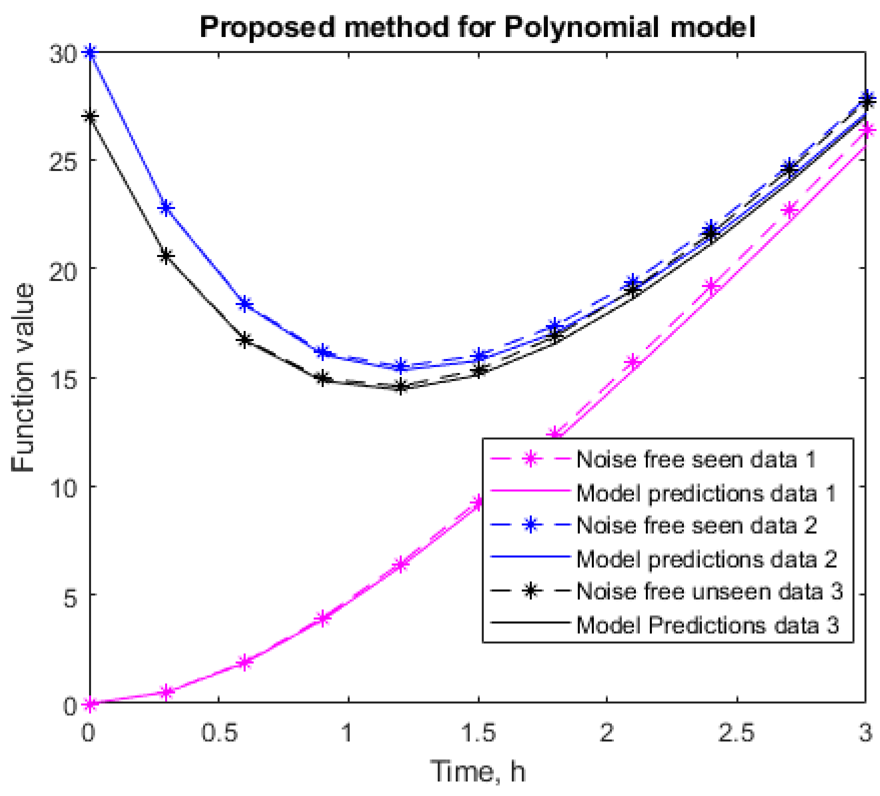


Figure 6.11 Performance results for Derivate estimation, Latin hyper cube and proposed state substitution methods, with 0.3h sampling and 20% noise for measured data.

6.1.3. Summary of results for polynomial model

This case study serves as a benchmark model, to show that complexity is not proportional with the number of parameters. It is showed that even with large number of parameters (Twenty), every method was able to provide highly accurate solution. This case study also shows how noise effect each method, although complexity is low increasing noise does decrease accuracy of the solution, for each of the models. With this low complexity model, this case study allows to assess viability of the complexity analysis tools. Each of the complexity analysis tools, provide different way to visualise complexity of the system, and together form a way to quantify the complexity of the model. This complexity measurement is not absolute, but rather a relative way to measure complexity. It allows to compare complexity of two different models, but not set a classification of the system complexity. The final decision of system complexity classification (Low, Medium, High) is up to users' interpretation. Using system, with known complexity based on prior knowledge is a good way to help establish what kind of results you should expect from this complexity analysis based on their complexity class. Using these methods complexity was assessed to be low for the polynomial model, because of several reasons. First, it consists of single ODE, which means there no coupling interactions between different ODE’s, making it easier to identify parameters, as they are all affected only by only one ODE. PCA visualization confirms this by forming a single uniform valley, that drops towards the global optimum. In addition to this convexity of this system is 100%, meaning it does not have problematic regions, that would stop system from converging. These results already suggesting this system has low complexity as it is simple to solve and has one global performance optimum. Sorted minimization reveals that system’s parameters are not possible to identify. This is expected as explained in chapter 6.1.1.2, structure of the model makes exact parameter value unidentifiable, because all parameter values are interchangeable. Model still can be optimised to perform well when compared to measured data. This also suggest that individual values of the parameters are not sensitive, for the same reason. SOM analysis confirms this by showing that system has multiple local minimums, but they all produce same level of performance. SOM analysis also provides supporting evidence of the parameters being not sensitive as cross-correlating SOM parameter maps with the overall model error SOM map is not able to identify any dominant variables. When comparing results of different parameter identification algorithms, for this system the Latin hyper-cube method is the most accurate method, in all five experimental setups. Accuracy of other two methods is comparability close for the first four experimental setups, but difference between the Latin hyper cube method increases when noise level reaches 20% mark. Accuracy levels for all methods were consistent between seen data (Pink and Blue) and unseen data (Black). Computational time is significantly different within all five experimental setups, with the Latin hyper cube method having lowest computational time, followed by other two methods. This is mainly due to small search space and simple model error hyperplane. This makes required number of samples to be low, therefore reducing computational time as well. This benchmark also shows that the proposed state substitution method is not suitable single ODE system, because decoupling cannot be used. This makes the proposed state substitution method perform worse than it would on a more complex system, when comparing with state of art methods. Although performance of identified model is satisfactory, all identified parameters values are not consistent. Identified parameter values are not consistent nor with theoretical values nor with other methods identified values. This reinforce the fact that parameter values are non-sensitive and unidentifiable, but good accuracy of the model can be still reached.

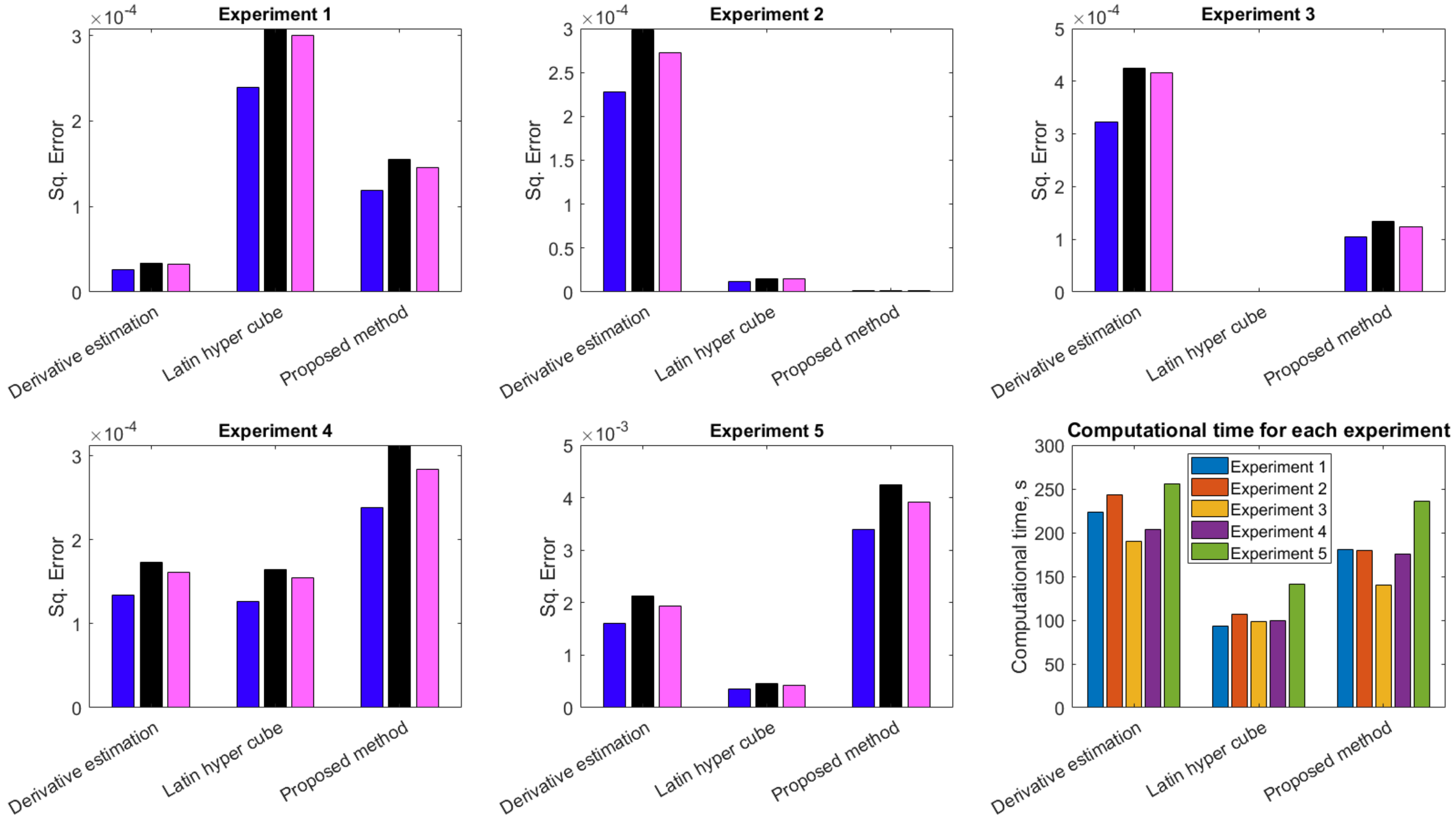


Figure 6.12 Summary of performance results for all three methods for polynomial model

6.2. Case study 2 – Monod kinetics

Monod kinetics are widely used for modelling bio-systems growth, while being relatively simple model. This makes Monod kinetics a good benchmark problem to evaluate the proposed new method. Monod kinetics also allows to observe how accurate complexity analysis is on coupled ODE system, as a lot of knowledge is known about Monod kinetics, to compare results. Monod kinetics in its simplest form, which is used for this case study, consists of two ODE's and three parameters. Both ODE's are coupled, and three parameters vary in sensitivity. Where maximum specific growth rate (Umax) and substrate consumption rate (q) values has large effects on model accuracy and half velocity constant (Ks) value, has low effects on model accuracy. Making half velocity constant hard to precisely identify. Equation 6.2 shows the system. Data was generated as described in section 5.4.

$$\begin{cases} \dfrac{dX}{dt} = \dfrac{Umax * S}{Ks + S} * X \\ \dfrac{dS}{dt} = -q * \dfrac{Umax * S}{Ks + S} * X \end{cases} \tag{6.2}$$

6.2.1. Complexity analysis

Complexity analysis will be performed in three steps as mentioned in section 6.1.1. These steps consist of performing PCA visualization with convexity calculations, sorted minimization and SOM analysis. The aim of this analysis is to compare relative complexity with other cases studies within this work, and to find problems gradient-based algorithms might run into trying to optimize this system.

6.2.1.1. PCA visualization

Figure 6.12 shows different structure, than the one described in section 6.1.1.1. Visualization of error plane shows as much more clearly defined global minimum and has non-convex regions that were not present in polynomial case study. Similar to first case study five thousand sample points, were used in PCA visualization of Monod model. These points were calculated as described in section 6.1.1.1. We can see that global optimum is well defined by sharp valley in middle of graph, but it might not be as easy to find because we can observe other two feature in this graph. A valley within which global optimum sits, and a flat area surrounding that valley. As we would expect those flat areas are non-convex, as starting in them gives an optimizer almost no indication where the minimum is. Secondly, this system also has failed integration points, which means we cannot assess their convexity, due to solver limitations. For each convex point three different values need to be evaluated (Section 4.5), failure to evaluate any of those three conditions gives failed integration/green dot. This graph suggests that the system can be difficult to optimize if the initial conditions fall within flat/red dot regions. This is also represented by overall convexity of 27%. Overall, we can conclude that the system is not a trivial problem to optimize but should not a pose big challenge if initial conditions are chosen correctly. That said, it can be tricky to get to true global optimum as it is hiding within a secondary valley that can be mistaken as the global optimum. Figure 6.12 is very good representation of the search space as PCA1 explains 90.9% variability of the system and PCA2 8.2% for a total of 99.1% variability.

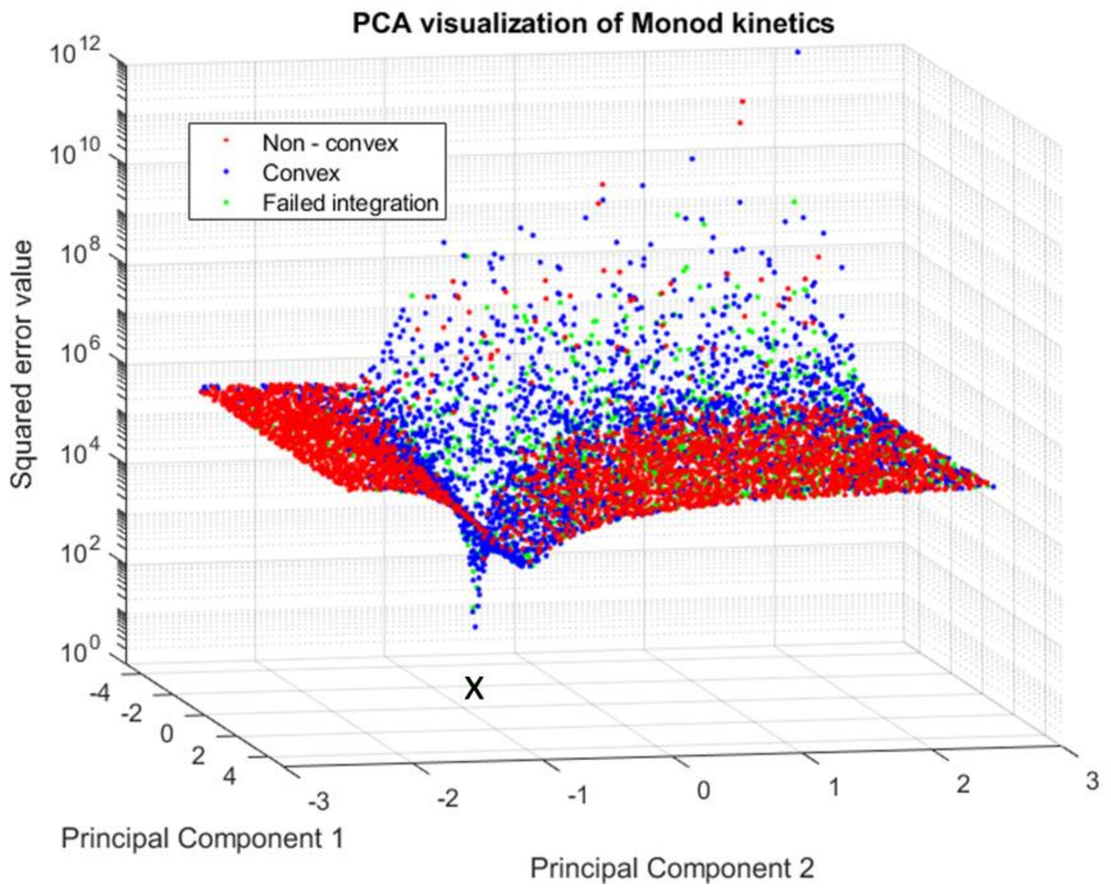


Figure 6.13 PCA error plot for Monod model (6.2), with colour coded points for convexity, where blue points are convex, red points are non-convex and green points are failed integrations. Black x marks global optimum solution.

6.2.1.2. Sorted minimization

Sorted optimization is performed same way as described in section 4.7. Parameter space is sampled for thousand initial parameter values. Optimization of these thousand samples, makes majority of samples to converge to same value, hence a flat line for figure 6.13. There is a drop down at last 200 samples but change in overall error is negligible. Still those last couple of point indicate they are approaching true global optimum, and not a valley around it which gives good performance. We can also see from figure 6.13 that even points that started at non-convex regions can convergence close to global optimum within this system. In addition, we can see that identifying convex points is beneficial as they hold same information as the full spectrum but require less points to be optimized. We can be sure that parameters values can be identified accurately, and minimization shows one dominant optimum (flat line).

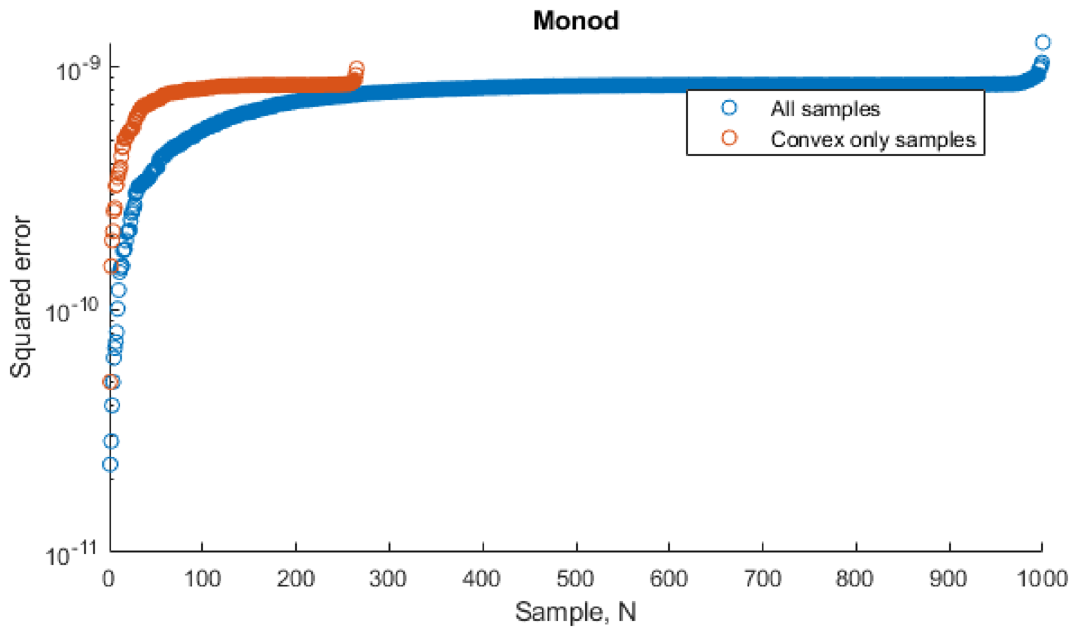


Figure 6.14 Monod model (6.2) thousand samples sorted optimization, where blue is all samples and red are convex samples only.

5.2.1.1. *SOM analysis*

Looking at figure 6.14 results of SOM, would suggests that there are multiple local minimums as lowest error green hexes(L) are not connected. With careful observation we can see green low error hexes(L) form two lines one closer to red hexes and second further away, also between these two lines distances are much smaller than anywhere else. Inspecting the U-matrix in figure 6.14 reveal that, this SOM has two places where the distance between hexes reaches very low values. One of those low distances lines are in the middle of red high error(H) hexes and second is in between green low error(L) hexes. This observation allows to see true general trend of high value on the left getting lower and lower while going to the right and finally increasing to blue medium error value(M) on the right edge again. This agrees with what we saw in a PCA visualization earlier, a valley of close to global optimum solutions.

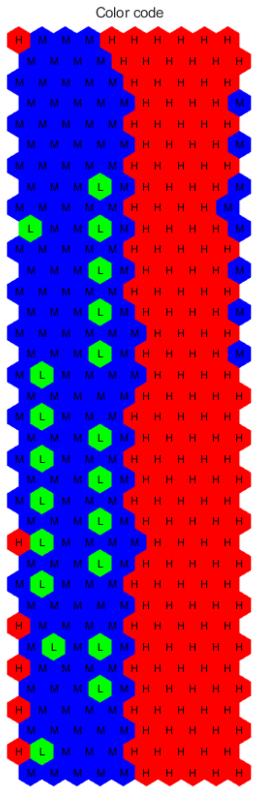
We know that with the Monod system parameter Umax is dominant and only at very low substrate values Ks become dominant parameter. To see if our SOM parameter analysis can find that we will cross correlate SOM error map with parameters maps to see which of them match best. If our cross-correlation method is correct it should show biggest matching between Umax and error fallowed by q and Ks. As before cross-correlation of maps are performed for positive and negative correlation.

Figures 6.16-6.17 shows how well component maps match up versus total error maps. Positive values mean higher degree of matching, and lower values mean lower. Cross correlation factor of 1, represents perfect match and 0 represent mismatch. As expected, none of the parameters seems to be more dominant than others. These figures allow to see how well, parameter changes match overall error changes in the system and to represent a quantifiable measurement of how dominant the parameter an overall matching factor was calculated (Table 6.18).

Parameters	**Positive Matching Factor**	**Negative Matching Factor**
Umax	0.82 ± 0.01	0.47 ± 0.00
Ks	0.61 ± 0.01	0.59 ± 0.01
q	0.66 ± 0.00	0.65 ± 0.01

Table 6-19 Table of positive and negative mean matching factor of each parameter for system (6.2)

We can observe that there is significant difference between matching factors as maximum difference between matching factors are 0.16 and 0.18, respectively. Furthermore, Umax shows the highest level of agreement with overall error map as expected. This provides additional confirmation that Umax is most dominant variable. This allows us to have more confidence when arranging order in which ODE are solved, for proposed state substitution method.



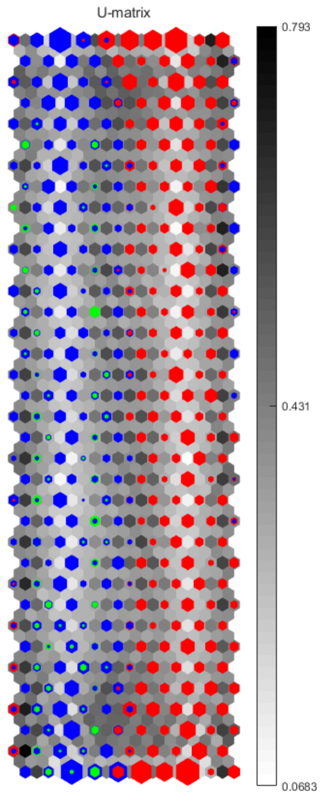


Figure 6.15 Left 2d SOM of Monod model (6.2), colour coded based of model error, where green (L) is low error, blue (M) is medium error, and red (H) is high error. Right 2d SOM with same colour code as left, but also showing relative distance in n-dimensional plane represented as colour bar and number of individual members of each cluster.

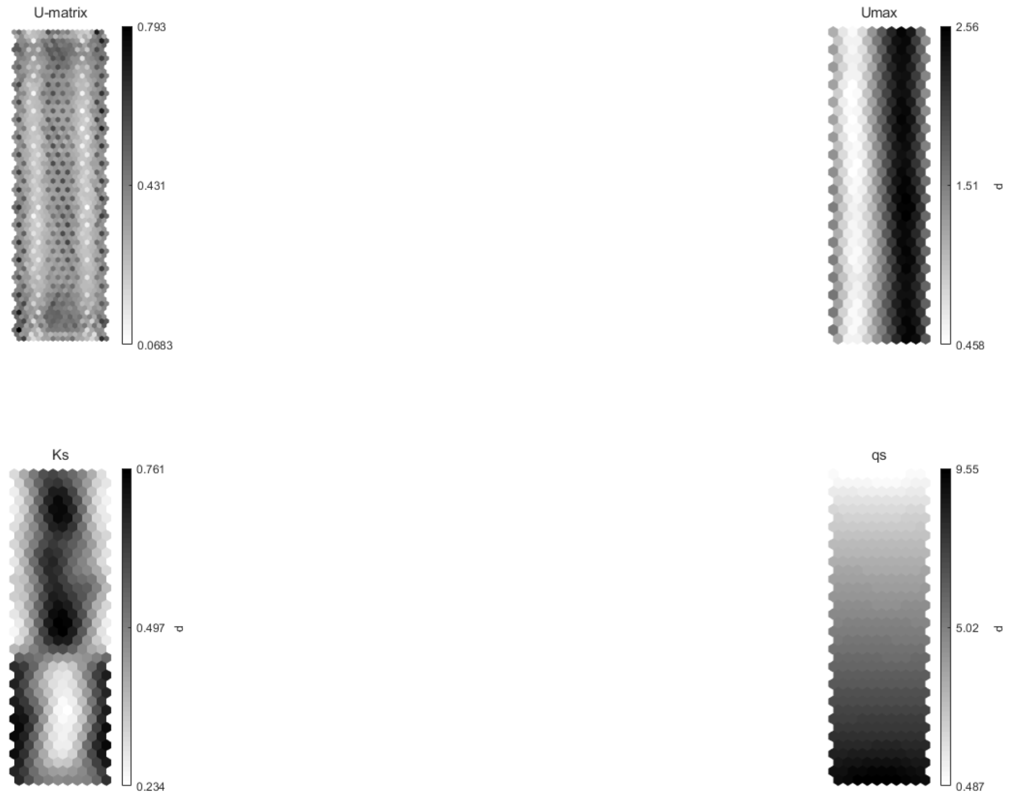
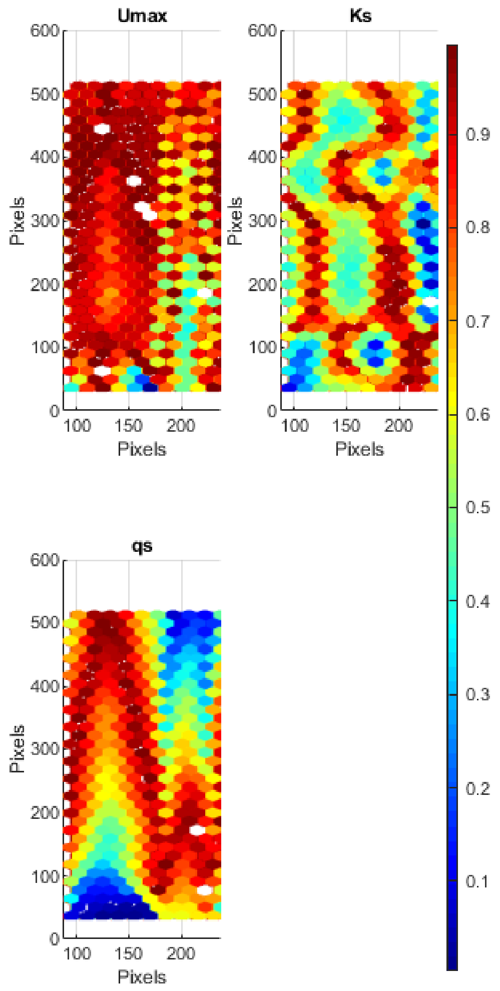


Figure 6.16 U-matrix represents N-dimensional plane as 2d SOM with relative distances as colour bar, followed by each component n-dimensional plane of its value distribution represented as colour bar

a)



b)

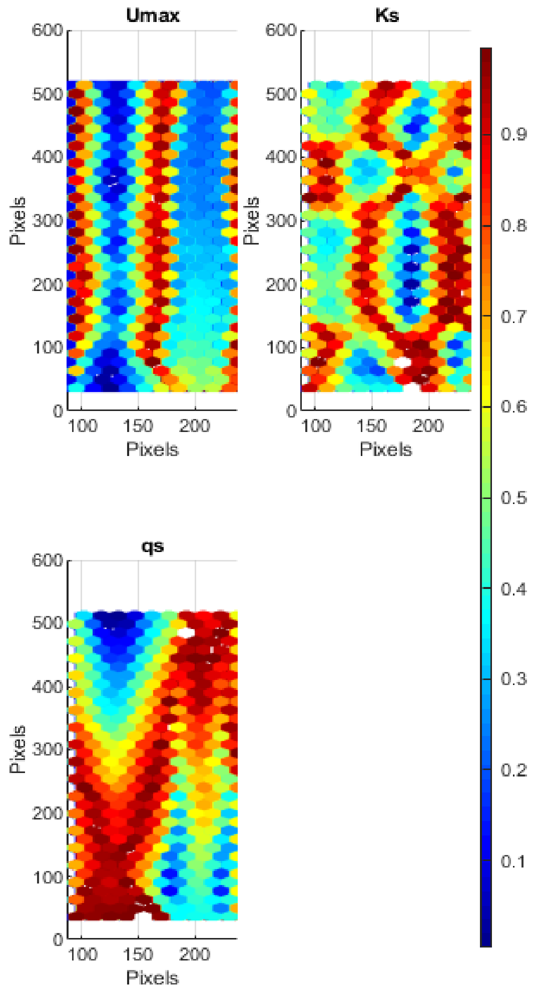


Figure 6.17 a) Positive cross-correlation between each parameter map and overall error map. b) Negative cross-correlation between each parameter map and overall error map.

6.2.2. Model hierarchy

The Monod model (6.2) consists of two coupled ODE's, which when applied to the proposed state substitution methods' decoupling algorithm leaves two independent sub-sets that can be solved in any order. Yet a specific solution hierarchy will lead to better results, because of the different levels of sensitivity from the parameters. In order to figure out the model hierarchy we need to look at the bi-partite chart (figure 6.18). This reveals that both states have same level of importance and could be solved simultaneously, but from (figure 6.16 and table 6.18), we can see that the parameter Umax, has highest correlation with model error, hence highest sensitivity. This leads to need for one of the sub-sets to be solved first to acquire the Umax value for best overall results. Although X(Biomass) and S(Substrate) sub-sets could be chosen, X sub-set is chosen, because it only has one other parameter associated with it (Ks). This means that the optimization algorithm will be more aware of Umax effects to the sub-set as it has less variables to optimize. This makes a two-level model hierarchy, with Umax value being passed on from level one to level two (figure 6.19).

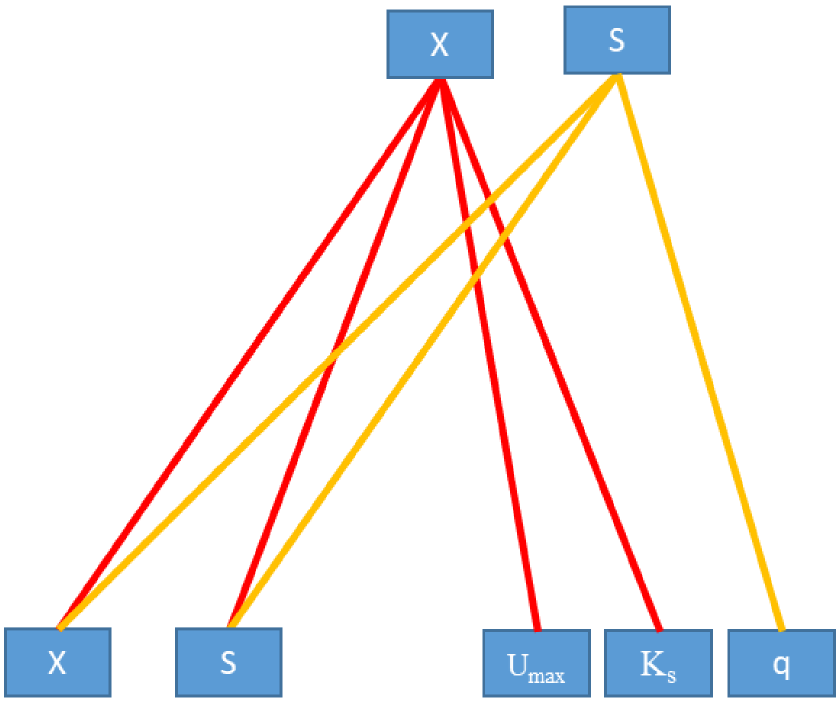


Figure 6.18 Bipartite chart of the Monod model (6.2)

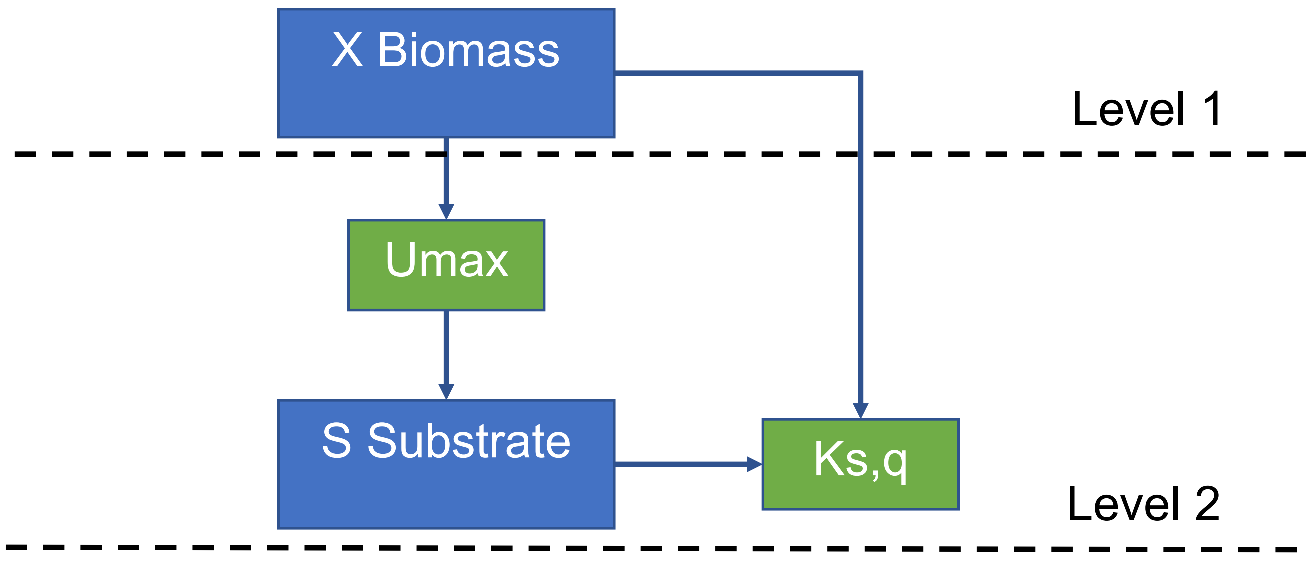


Figure 6.19 Hierarchy of the Monod model (6.2)

6.2.3. Method Comparison

To evaluate the proposed state substitution method, it will be compared to the two existing methods discussed in chapter 3. The derivative estimation and Latin-hyper cube sampled multi start method. Two main criteria will be compared, model performance accuracy and the computational time between each of the methods. Although the proposed state substitution method aims to reduce computational time, model performance accuracy is also very important and cannot be completely neglected. Each method will be assessed with different random noise levels and sampling times. For each method three different state initial conditions are compared, first two (Pink and Blue) are set same for all methods and are conditions that were provided for the optimisation algorithm and third initial condition (Black) is randomised between first two initial conditions and was never seen by algorithm before. This allows to check method accuracy with unseen data sets, which are within same boundary conditions. Only biomass data set will be presented for model performance, as biomass dictates accuracy of substrate model. This will allow to avoid unnecessary graphs while still presenting enough evidence about model performance. However, for completeness first experiment will show predictions for all states. Modelling conditions and parameter search space are summarised in tables below.

Experiment number	Sampling Rate	Noise level	Initial conditions Pink	Initial conditions Blue	Initial conditions Black
1	0.1h	5%	0.1, 0.1, 0.1	0.5, 0.5, 0.5	0.54, 0.57, 0.14
2	0.1h	10%	0.1, 0.1, 0.1	0.5, 0.5, 0.5	0.17, 0.14, 0.15
3	0.3h	5%	0.1, 0.1, 0.1	0.5, 0.5, 0.5	0.31, 0.47, 0.12
4	0.3h	10%	0.1, 0.1, 0.1	0.5, 0.5, 0.5	0.47, 0.15, 0.35
5	0.3h	20%	0.1, 0.1, 0.1	0.5, 0.5, 0.5	0.21, 0.45, 0.18

Table 6-20 Summary of modelling conditions for the system (6.2)

Parameters	Lower bound	Upper bound
Umax	0	3
Ks	0	1
q	0	10

Table 6-21 Parameter search space for the system (6.2)

6.2.3.1. Experiment 1 0.1h sampling and 5% random noise

When we look at computational times (table 6.20) we can see that proposed state substitution method already produce significant improvements, in comparison to previous case study (6.1). That is, because Monod model (6.2) has coupled states and proposed state substitution method decouples them to reduce search space, in order to reduce computational time. When comparing proposed state substitution method with Derivative estimation and Latin hyper cube method we get 107% and 38% improvements in computational time, respectively.

Method	Computational time, s
Derivative estimation	425
Latin hyper cube multi-start	190
Proposed state substitution method	129

Table 6-22 Computational time of all three method for experiment 1

In terms of accuracy of model predictions, Latin hyper cube and proposed state substitution methods seemed to be of similar accuracy, with Latin hyper cube method being slightly better in certain cases. Whereas derivative estimation method falls short of accuracy in comparison (table 6.21).

Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	7.18E-03	6.50E-05	4.01E-03
Latin hyper cube multi-start	4.00E-05	1.18E-04	6.75E-05
Proposed state substitution method	2.38E-04	1.79E-04	1.48E-04

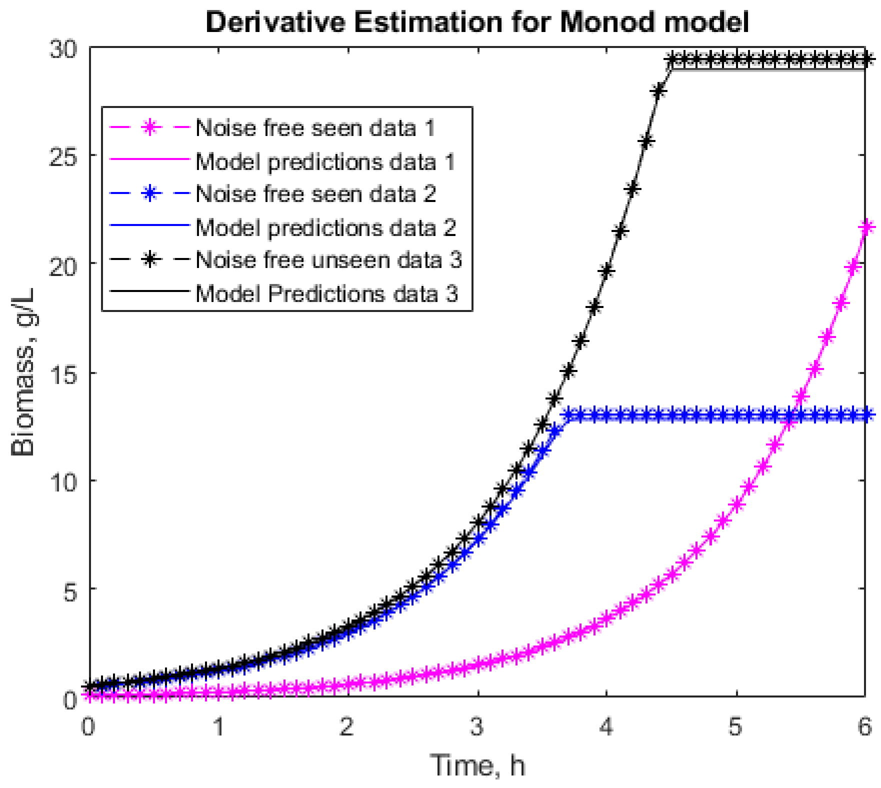
Table 6-23 Squared error values of each method and each data set for experiment 1

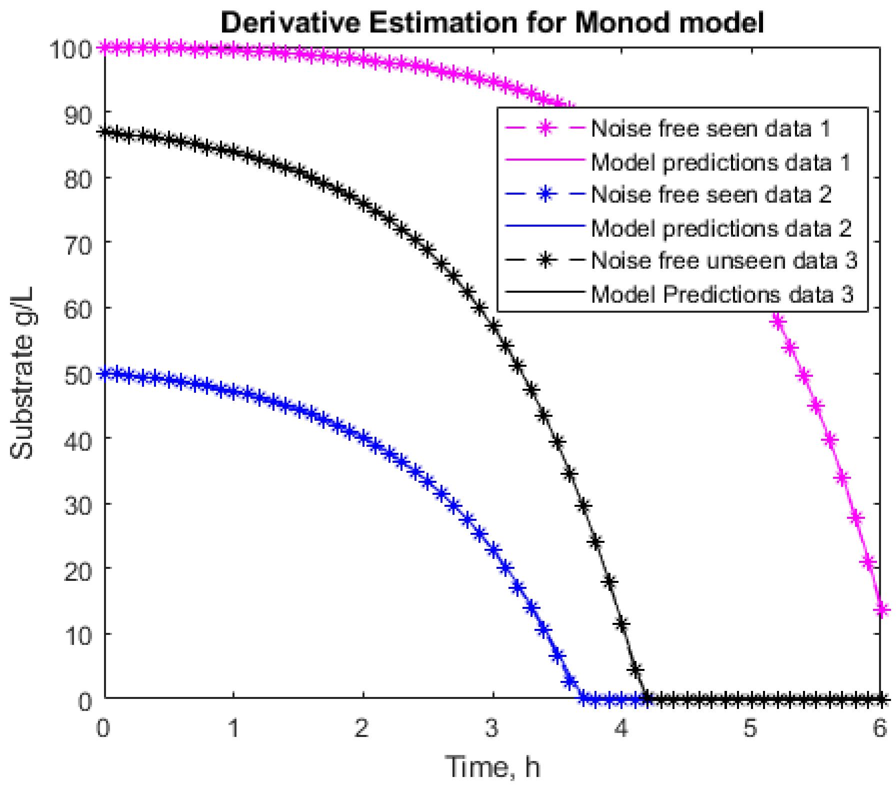
In can be noted that if we compare computational times to same experimental setup of polynomial model (6.1) (table 6.3). We will see that due to increased complexity of the system Derivative estimation and Latin hyper cube methods both roughly doubled their computational times, where proposed state substitution method decreased it. Both methods have positive correlation between complexity of a system and computational time, whereas because proposed state substitution method is using decoupling techniques, it will struggle with low complexity systems, but will outperform other methods when dealing with high complexity systems.

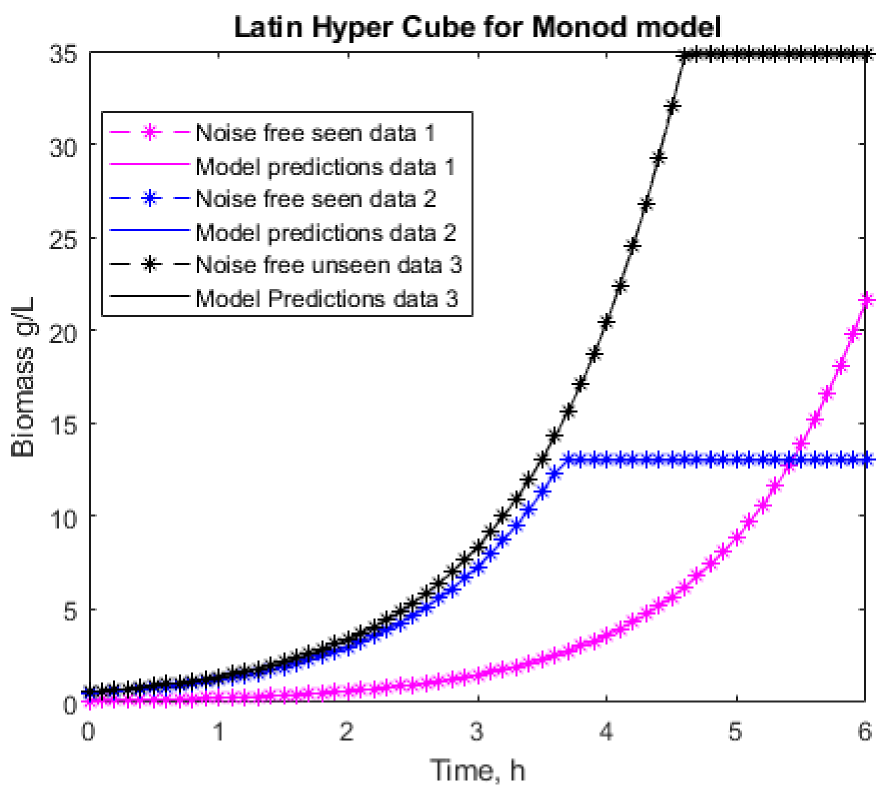
When looking at identified parameters values, we observe that as predicted from SOM analysis most important variable is Umax, as all methods identified this parameter accurately. Ks parameter is not very sensitive, because Latin hyper cube and proposed state substitution method are very close in accuracy, but Ks values is different by 24% and Derivative method has 100% change and still able to keep relatively accurate model prediction, although not as accurate as other two methods. Which is why it is most likely that q parameter is responsible for that accuracy change, as it is change from theoretical value seem to match with change in Sq. Error, between methods.

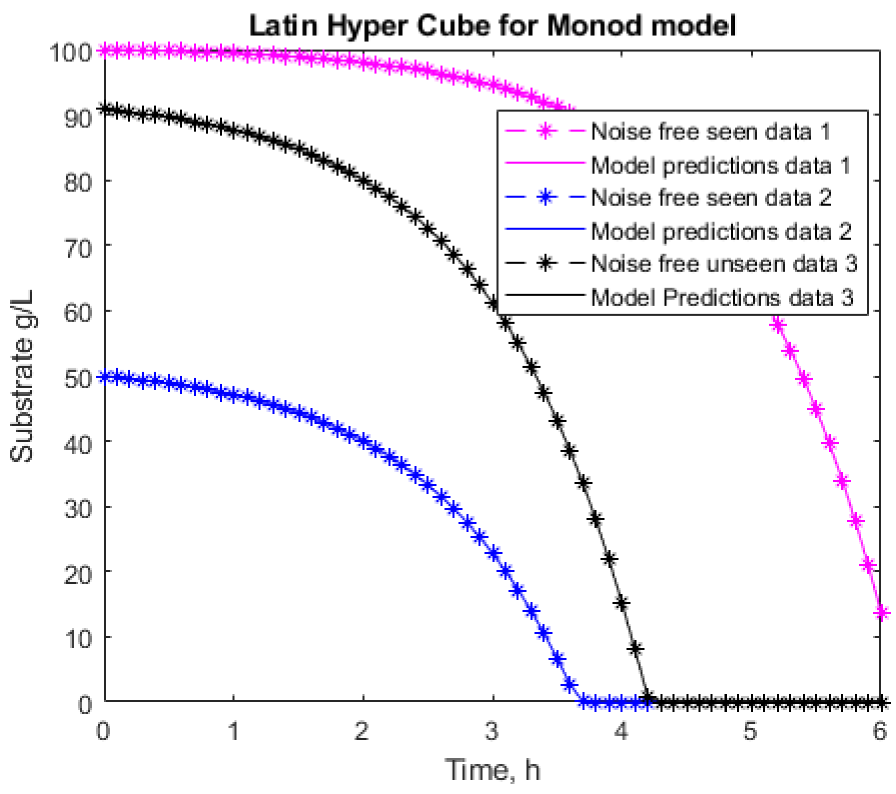
Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
Umax	0.9	0.90	0.90	0.90
Ks	0.3	0.60	0.30	0.23
q	4	4.06	4.00	4.01

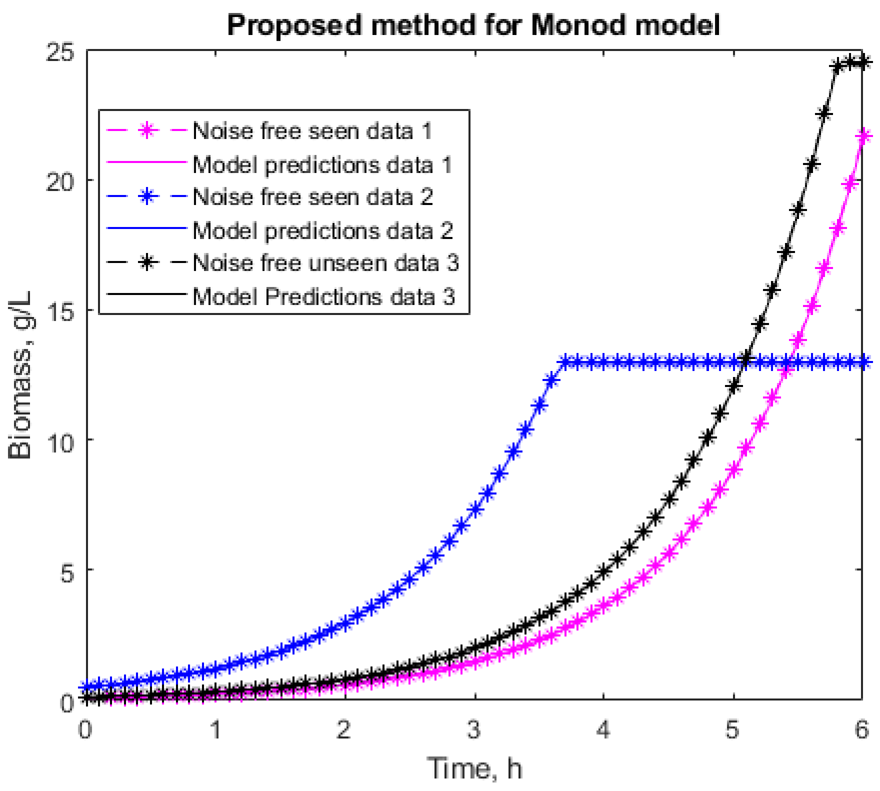
Table 6-24 Summary of identified parameter values for each method for experiment 1


Derivative Estimation for Monod model
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3
Biomass, g/L
Time, h


Derivative Estimation for Monod model
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3
Substrate g/L
Time, h


Latin Hyper Cube for Monod model
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3
Biomass g/L
Time, h
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6


Latin Hyper Cube for Monod model
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3
Substrate g/L
Time, h
100
90
80
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6



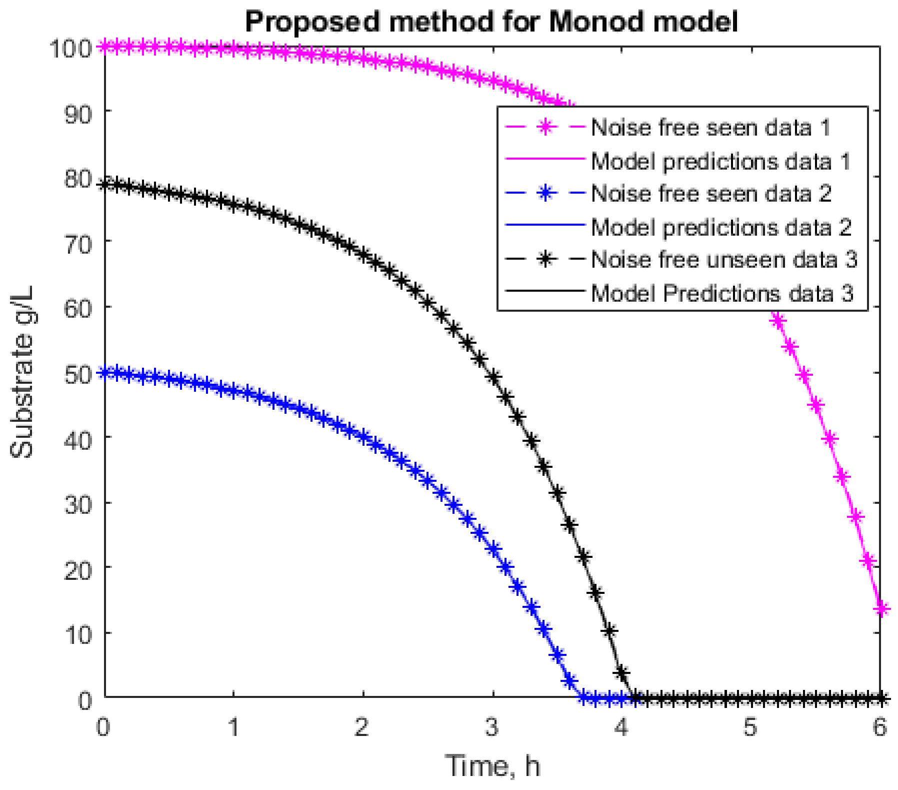


Figure 6.20 Performance results for Derivate estimation, Latin hyper cube and proposed state substitution methods, with 0.1h sampling and 5% random noise for measured data.

6.2.3.2. Experiment 2 0.1h sampling and 10% random noise

As expected with increased noise from 5% to 10% computational time of all methods should increase. Pattern of computational time stay the same as in experiment 1. Proposed state substitution method is still fastest by 107% and 34% compared to Derivative estimation and Latin hyper cube method, respectively.

Method	Computational time, s
Derivative estimation	455
Latin hyper cube multi-start	193
Proposed state substitution method	137

Table 6-25 Computational time of all three method for experiment 2

Accuracy still high of all method with Derivative estimation method providing lowest accuracy and Latin hyper cube and proposed state substitution method providing slightly better accuracy. All methods suffer loss in accuracy in comparison with experiment 1, due to increased noise level. When comparing squared Errors of experiment 2 to experiment 1, we see that experiment 2 squared error are more uniform than experiment 1. This is caused by increased error levels, making measurements on average further from real data. This creates sort of limiting band how low error of the model can be pushed.

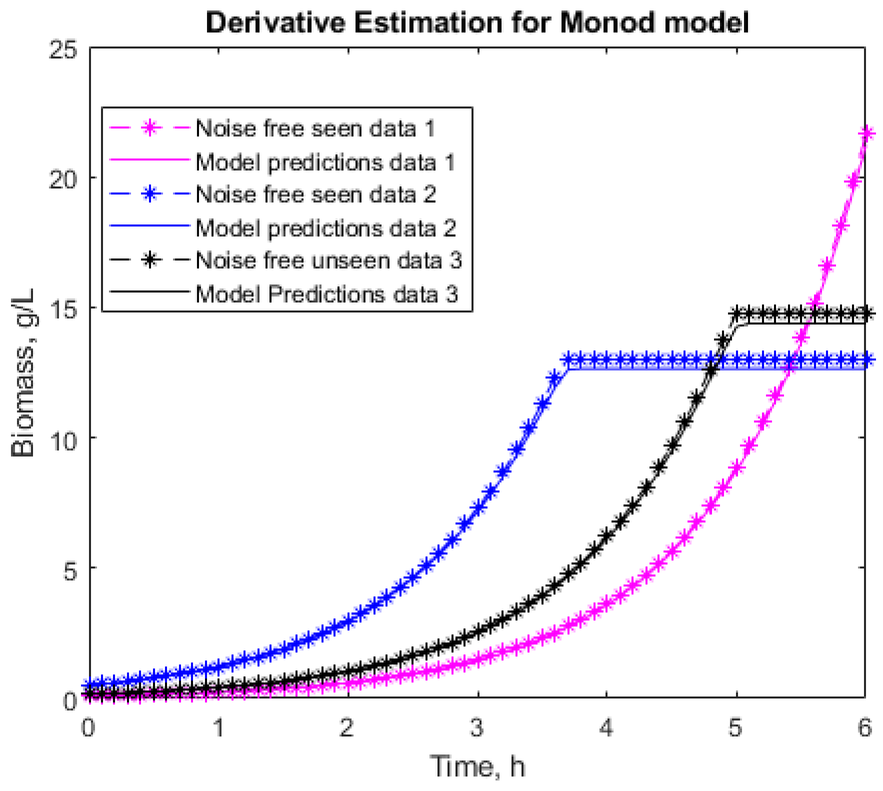
Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	2.17E-02	1.92E-03	1.29E-02
Latin hyper cube multi-start	2.09E-03	1.25E-03	1.15E-03
Proposed state substitution method	4.58E-03	3.16E-03	2.93E-03

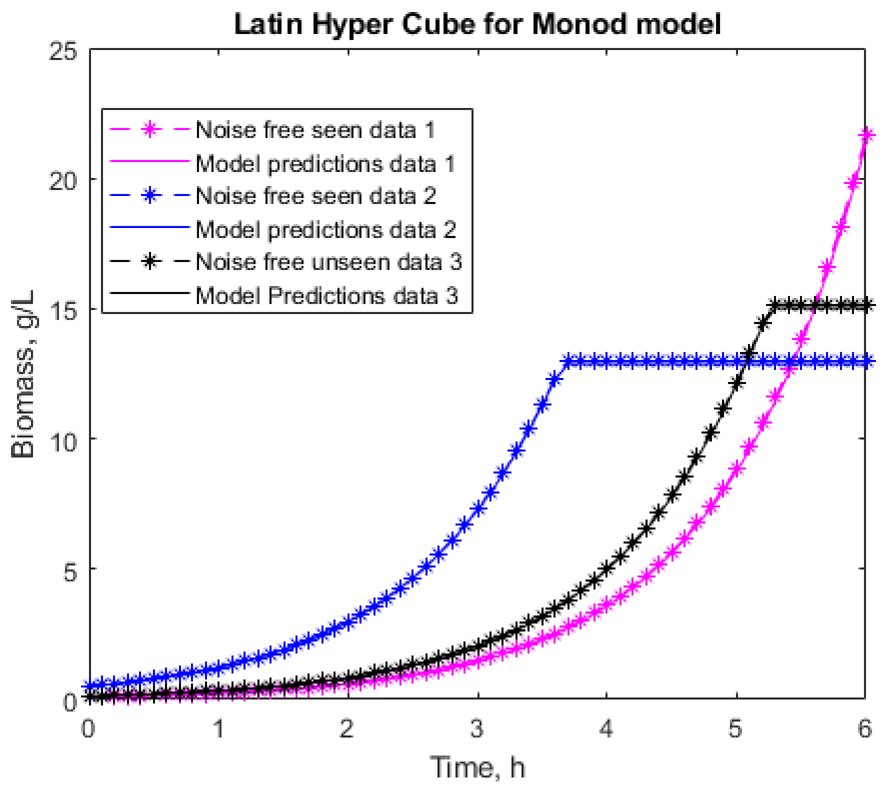
Table 6-26 Squared error values of each method and each data set for experiment 2

When looking at identified parameter values, we observe that Umax is still identified correctly by all method. This allows all methods to provide decent model prediction, because as mention before Umax is most sensitive and most impactful parameter in this system. With increased noise we see more variation in Ks and q parameters as their effect on system are slowly being hidden by noise. Ks parameter identified values vary more than q parameter, indicating that q parameters has more influence over the system than Ks.

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
Umax	0.9	0.90	0.90	0.90
Ks	0.3	0.57	0.22	0.34
q	4	4.12	4.04	3.95

Table 6-27 Summary of identified parameter values for each method for experiment





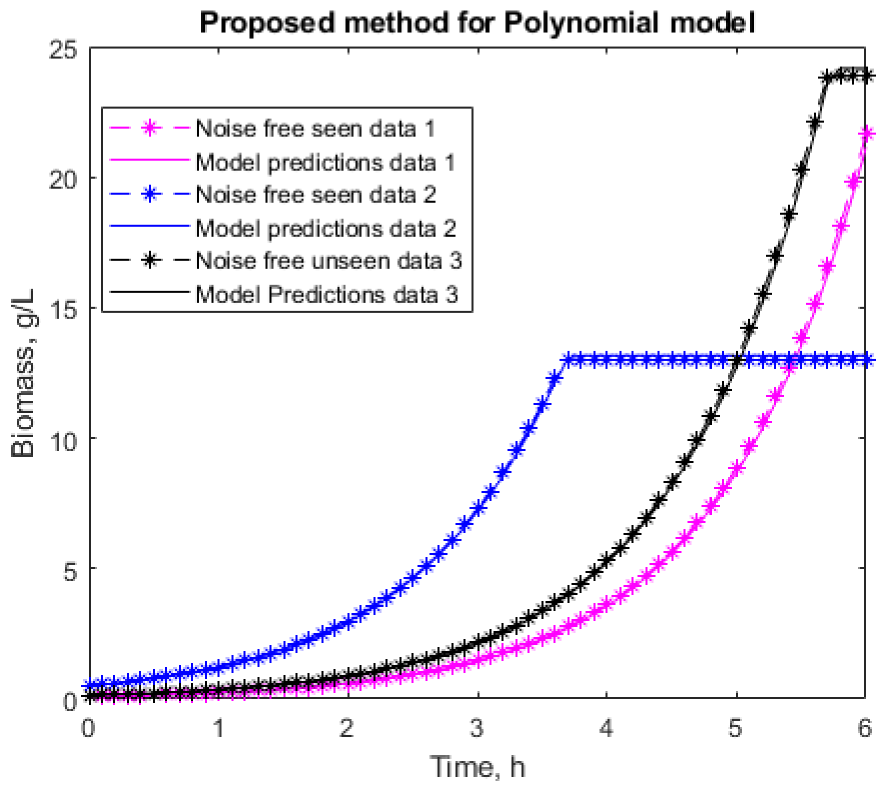


Figure 6.21 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.1h sampling and 10% random noise for measured data.

6.2.3.3. *Experiment 3 0.3h sampling and 5% random noise*

First thing we can observe in computational time that seemed to not fallow the pattern of previous experimental setups that, although higher sampling rate should result in more complex problem to identify, all the computational times have decreased. That is because with increased sampling time, there is less data points per same time span. Which reduce amount of calculation that required to be done by any method. Other than that same pattern persists of proposed state substitution method being fastest, followed by Latin hyper cube and Derivative estimation methods. Proposed state substitution method is 102% and 33% faster than other two methods, respectively.

Method	Computational time, s
Derivative estimation	377
Latin hyper cube multi-start	172
Proposed state substitution method	123

Table 6-28 Computational time of all three method for experiment 3

Accuracy overall fallows same pattern, with Latin hyper cube being most accurate, proposed state substitution method being just as accurate, but not in all three data sets, and derivative estimation method having worst accuracy. All methods are still capable to producing models that fallow general trend accurately. It should be noted that Squared error values of experiments with different sampling rates, cannot be directly compared as different number of samples affect total error value.

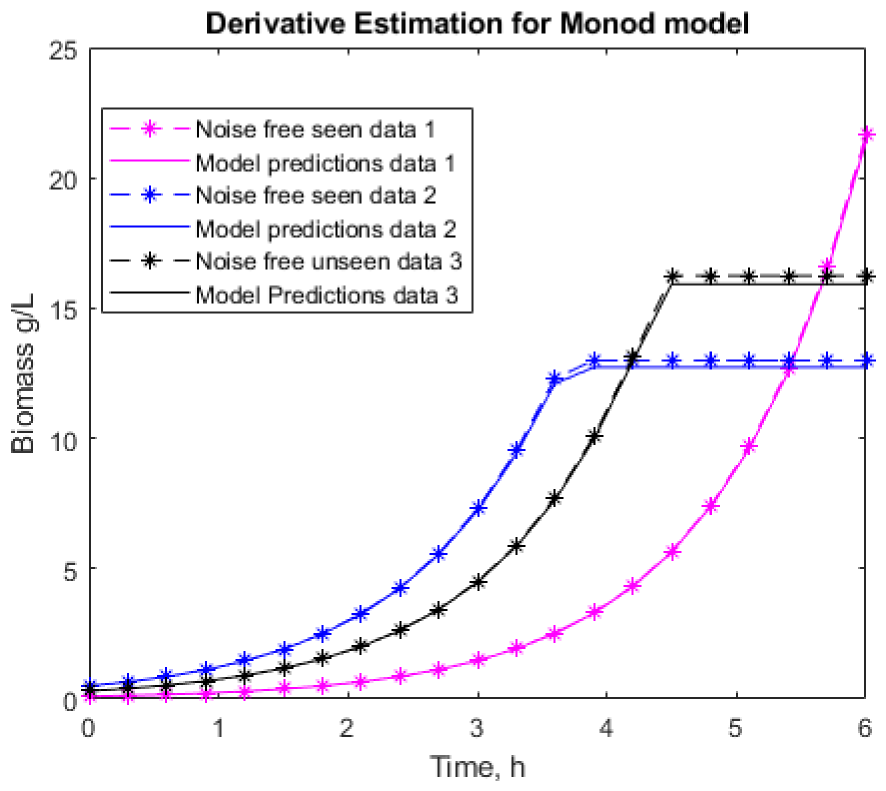
Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	3.64E-03	3.05E-04	2.72E-03
Latin hyper cube multi-start	6.86E-05	2.84E-05	4.74E-05
Proposed state substitution method	5.31E-04	1.15E-05	7.74E-05

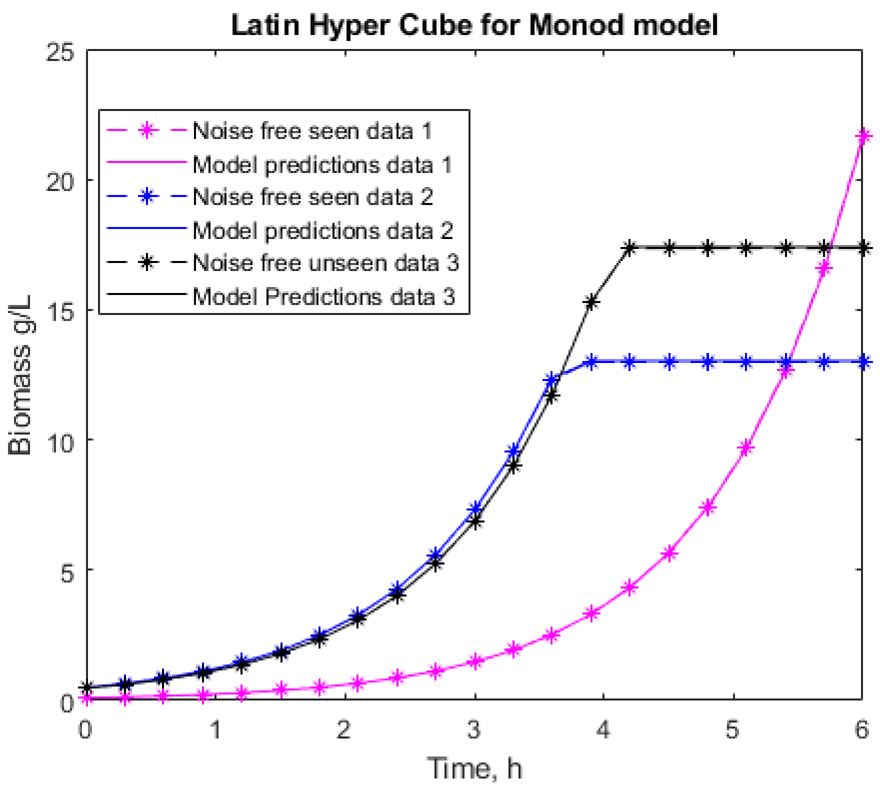
Table 6-29 Squared error values of each method and each data set for experiment 3

Umax is still being accurately identified by all methods, where q and Ks varies between methods. As in previous cases all difference in accuracy comes from difference in these less important variables, and as Latin hyper cube and proposed state substitution method have similar values of Ks and q, which leads to their accuracies being very similar too. This case all proves that q parameter has more important than Ks, as proposed state substitution method has better estimate of Ks parameter, but worse estimated in q parameter, which leads to Latin hyper cube being just a slightly more accurate than proposed state substitution method.

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
Umax	0.9	0.90	0.90	0.90
Ks	0.3	0.43	0.22	0.29
q	4	4.08	3.99	3.97

Table 6-30 Summary of identified parameter values for each method for experiment 3





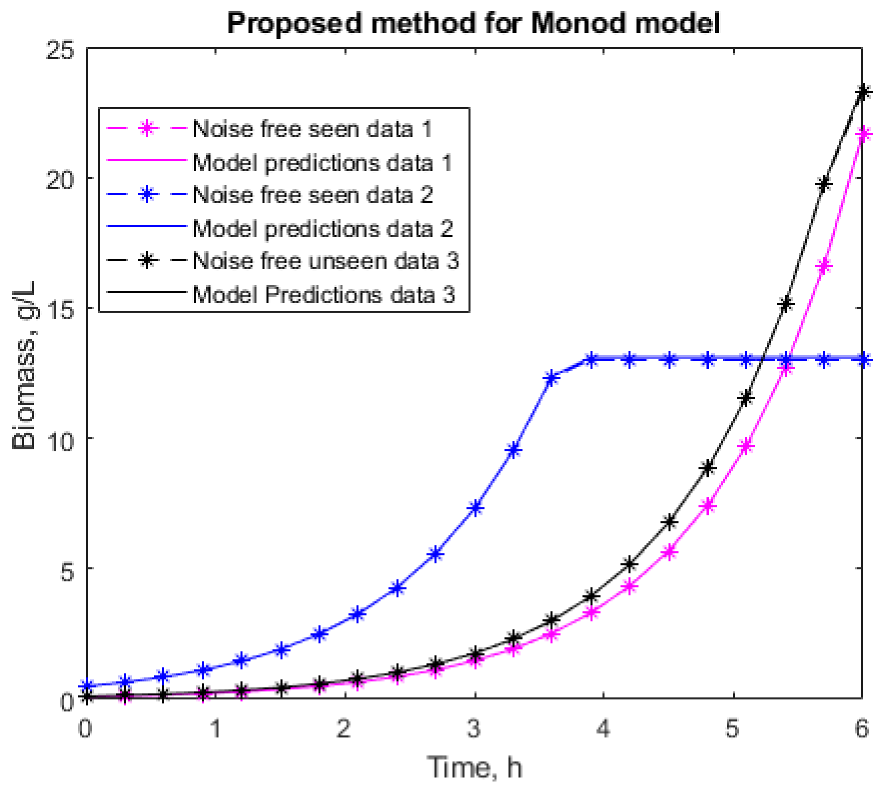


Figure 6.22 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.3h sampling and 5% random noise for measured data.

6.2.3.4. Experiment 4 0.3h sampling and 10% random noise

When noise is increased, we come back to regular pattern of all computational times increasing. Latin hyper cube seemed to be least affected by this change, but it is still slower then proposed state substitution method. Proposed state substitution method is 91% and 14% faster than Derivative estimation and Latin hyper cube methods, respectively.

Method	**Computational time, s**
Derivative estimation	410
Latin hyper cube multi-start	177
Proposed state substitution method	154

Table 6-31 Computational time of all three method for experiment 4

In terms of accuracy all methods start to be very similar although proposed state substitution method and Latin hyper cube method does seem to provide more accurate results then Derivative estimation in certain cases.

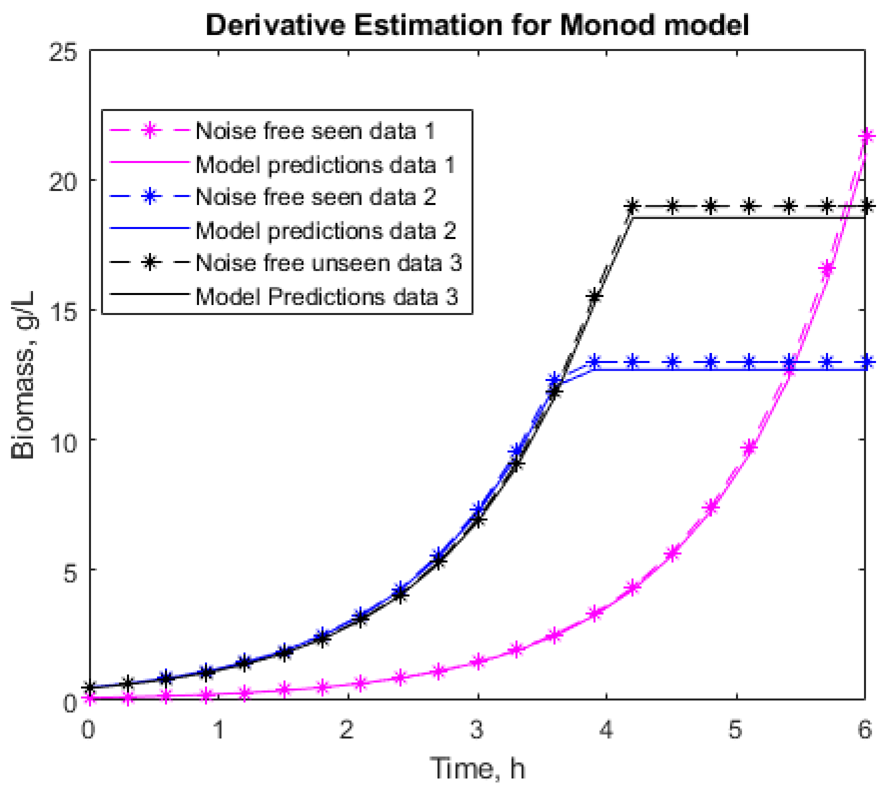
Method	**Sq. Error (Blue)**	**Sq. Error (Pink)**	**Sq. Error (Black)**
Derivative estimation	5.10E-03	2.16E-03	4.54E-03
Latin hyper cube multi-start	1.37E-03	1.29E-05	8.02E-04
Proposed state substitution method	4.66E-03	2.27E-05	4.06E-03

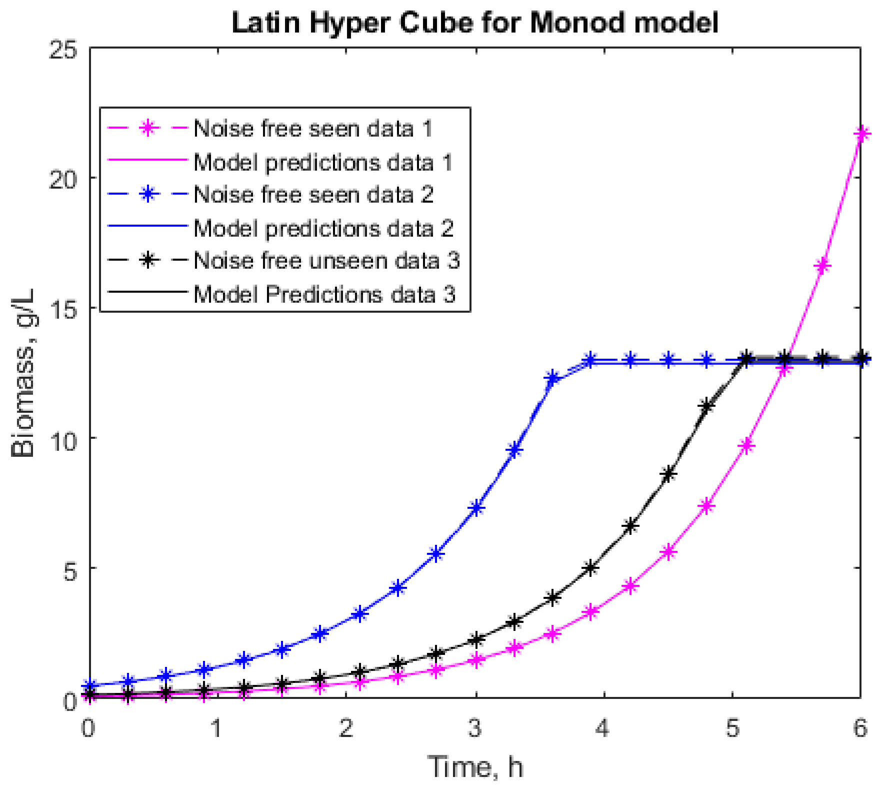
Table 6-32 Squared error values of each method and each data set for experiment 4

Identify parameter values are roughly same, just Ks values starts to vary even more. Umax parameter value also starts to be more difficult to identify. All these fluctuation in estimated parameter values is direct affect from increased noise, which slowly starts to hide parameter effects on the system.

Parameter	**Theoretical value**	**Derivative**	**Latin hyper cube**	**Proposed**
Umax	0.9	0.89	0.90	0.91
Ks	0.3	0.31	0.51	0.79
q	4	4.10	4.05	4.09

Table 6-33 Summary of identified parameter values for each method for experiment 4


Derivative Estimation for Monod model
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3
Biomass, g/L
Time, h
0
5
10
15
20
25
0
1
2
3
4
5
6


Latin Hyper Cube for Monod model
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3
Biomass, g/L
Time, h
0
5
10
15
20
25
0
1
2
3
4
5
6

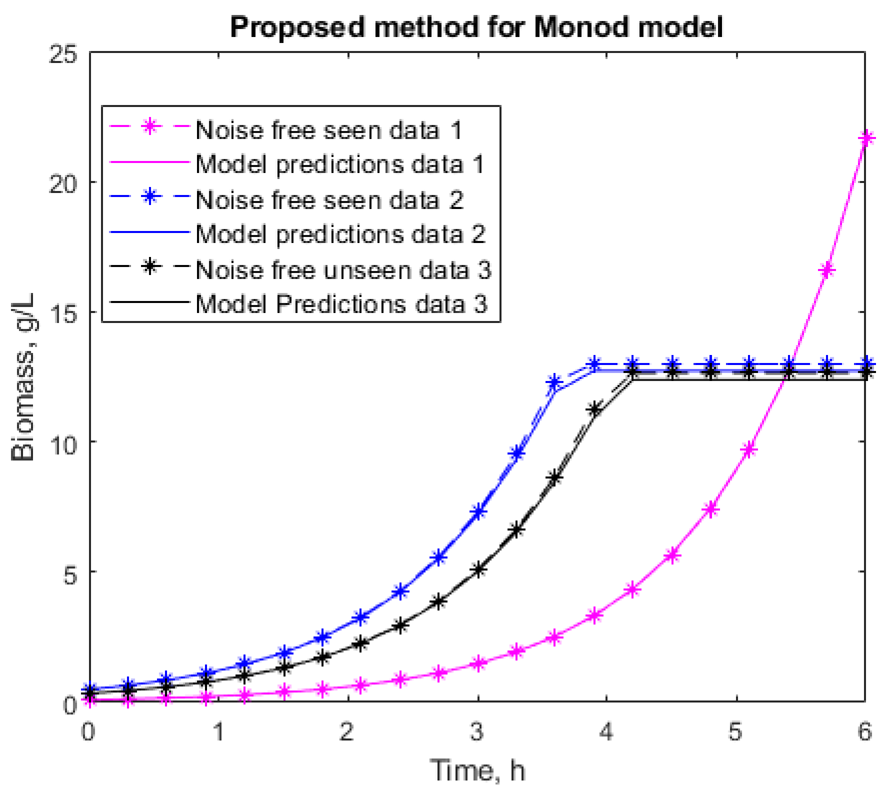


Figure 6.23 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.3h sampling and 10% random noise for measured data.

6.2.3.5. Experiment 5 0.3h sampling and 20% random noise

With increased noise from 10% to 20% computational time of all methods increased. Proposed state substitution method continues to be fastest. Proposed state substitution methods computational time is lower by 96% and 23%, when compared with Derivative estimation and Latin hyper cube methods, respectively.

Method	Computational time, s
Derivative estimation	413
Latin hyper cube multi-start	183
Proposed state substitution method	145

Table 6-34 Computational time of all three method for experiment 5

Noise also effected accuracy of all methods. Overall accuracy seems to be same but depending on specific initial condition different methods produce more accurate results. All methods are still capable to fallowing correct general trend of the system.

Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	4.88E-03	8.15E-04	5.19E-03

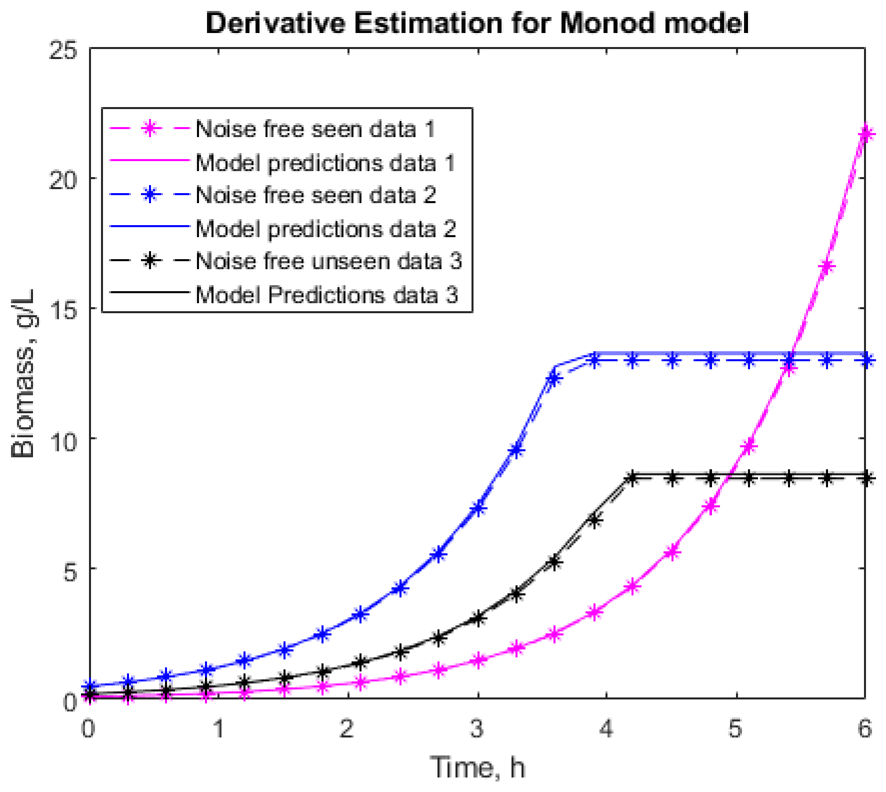
Latin hyper cube multi-start	6.64E-05	1.67E-03	5.28E-05
Proposed state substitution method	4.94E-03	2.16E-04	1.09E-02

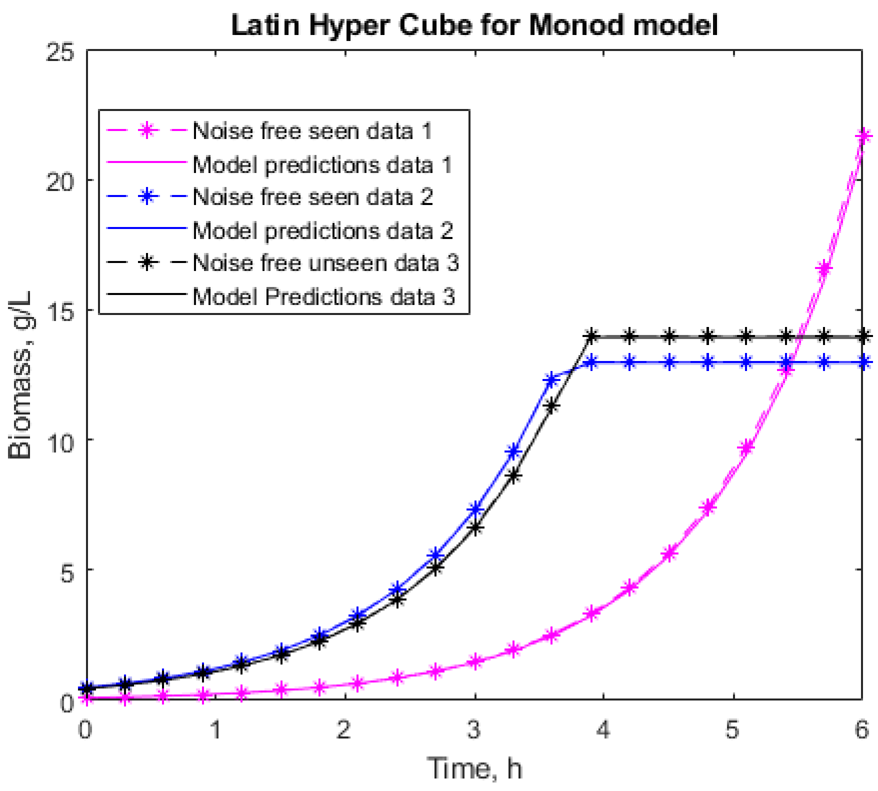
Table 6-35 Squared error values of each method and each data set for experiment 5

We can see that Derivative estimation method and Latin hyper cube method cannot identify Ks anymore, but this does not seem to have big impact on their model performances. Proposed state substitution method can identify Ks, but it is way off.

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
Umax	0.9	0.90	0.89	0.91
Ks	0.3	0.00	0.00	0.80
q	4	3.92	4.01	4.09

Table 6-36 Summary of identified parameter values for each method for experiment 5





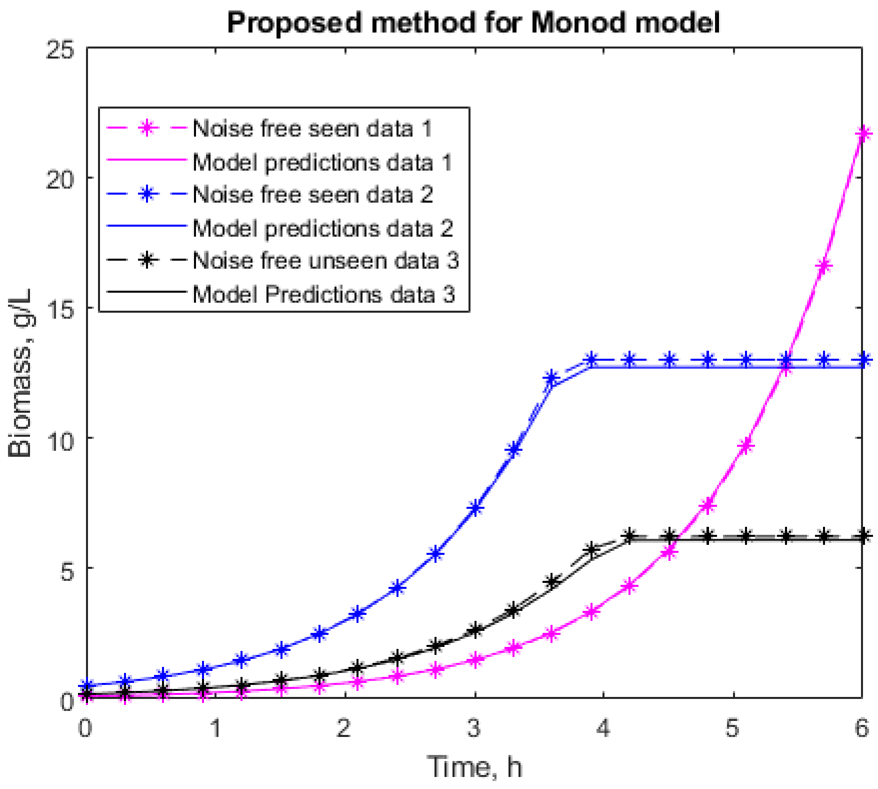


Figure 6.24 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.3h sampling and 20% random noise for measured data.

6.2.3.6. The summary of results for the Monod model

This case study is great example of small system that contains complexity, which comes from different ODE interaction. The Monod system only has three parameters, but its complexity is higher than the polynomial system that was discussed in section 6.1. As Monod kinetics is widely used, this system as studied extensively and well understood. This allows us to test complexity analysis tools, which can determinate dominant variables, as it is known that only one variable out of three is highly dominant (Umax). Using complexity analysis tools, complexity of this system was determined to be medium/low for the following reasons. First the Monod system consist of two ODEs that are strongly coupled, meaning changes in one of the are strongly reflected into other. This makes optimisation of the parameters more difficult, as parameters cannot be optimised one by one anymore and have to be optimised at the same time. PCA visualization reveal non-convex regions, which makes gradient-based optimisation methods, stop before converging into a solution. In addition, global optimum solution is surrounded by secondary valley, which can be interpreted as a local minimum. These complications are reflected in convexity measurement, which is 27%. This means that starting values of parameters heavily influence on the final result, as only roughly third of starting points will converge towards the solution. Sorted minimization shows that optimised model can reach accurate performance and accurate parameter values. The SOM analysis shows agreement with PCA visualization results, revealing a global optimum, surrounded by a secondary valley of local minimum. SOM analysis of parameter maps, shows that Umax is indeed most dominant variable, followed by q and Ks. Although SOM parameter analysis suggest that variable q is just slightly more dominant than Ks, identified values suggest that Ks is extremally non sensitive value, as it changes most between different experiments and methods but does not seem to impact the overall error of the model significantly. That being said, looking at the structure of ODEs (equation 6.1), we can see that variable Ks, impact increases as substrate concentration (S) decreases. Very small amount of data exists around low substrate values as it immediately tends to zero stopping the growth of biomass. This makes it hard to observe effect of Ks variable, but SOM parameter analysis is able to pick up that importance. When comparing results of different parameter identification algorithms, for this system the proposed state substitution method has lowest computational time across all five experiments. As there two heavily coupled ODEs the proposed state substitution method can fully utilize its decoupling technique to decrease the search space making it more efficient than other methods. Latin hyper cube method is the most accurate method, but not by significant margin when compared to the proposed state substitution method and derivative estimation method. Accuracy levels for all methods were consistent between seen data (Pink and Blue) and unseen data (Black). Identified parameter values, seem to be consistent in all five experimental setups, apart from Ks parameter. For the first three experimental setups Ks value was identifiable but had large error when compared to the theorical value, but during the experiment four and five it is not identifiable anymore. This is due to as discussed before, not being able to observe the impact Ks value has, and with increased noise levels making this observation because impossible for the algorithm. This case study is like a proof of concept for the proposed state substitution method. As complexity is not high and the system is well understood, it allows good comparison with other state of art methods. Due to ability to heavily decrease the search space, the proposed state substitution method outperforms other methods in terms of speed, but accuracy of the method could be better.

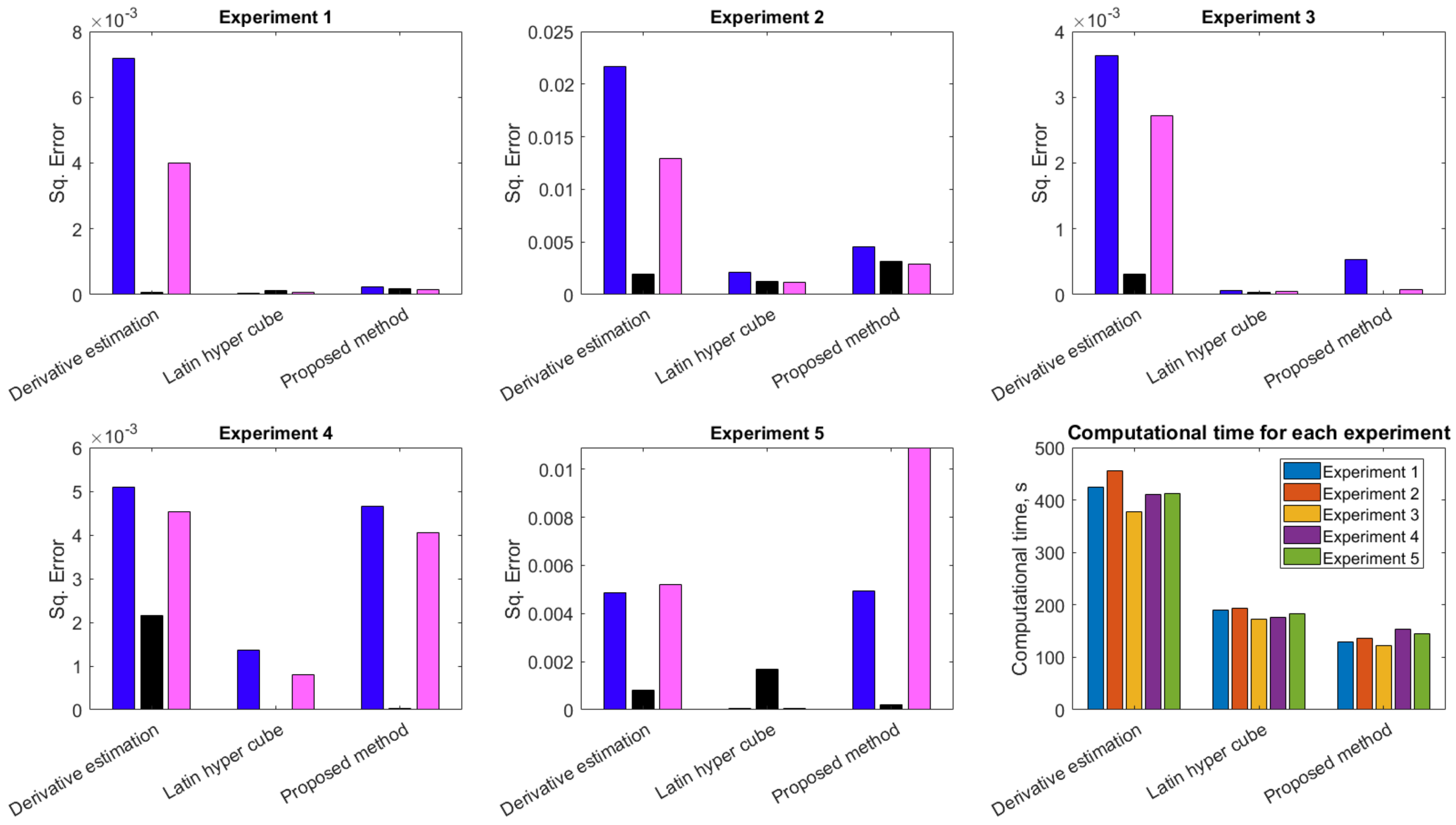


Figure 6.25 Summary of performance results for all three methods for Monod kinetics model

6.3. Case study 3 – CHO cell culture model

CCO Culture kinetics was selected as case study 3, for several factors. First it has increased complexity in model, consisting of two competing substrates, two by-products, biomass, and an antibody product. Secondly in comparison to case study 2, it has much deeper coupling, consisting of six ODE’s and sixteen parameters. First five ODEs are coupled between each other similarly to Monod kinetics in case study 2. There are four reactions happening within this model. Biomass growth with by-product production, biomass death, biomass sustain, and production of antibodies. All these reactions happen simultaneously (figure 6.25). Another unique characteristic of this model is that has several parameter ratios, which can mean both parameter values can only be identified if, one of them is known beforehand. This increase difficulty of obtaining values for all parameters. Model was obtained from (Saraiva *et al*. 2015). Equation 6.3 shows the system, later in the thesis parameters of this system will be preplaced by o1 - o16 encoding for easier reading.

$$Glucose + Glutamine \xrightarrow{Biomass} Biomass + Lactate + Ammonia$$

$$Biomass\ (alive) \rightarrow Biomass(dead)$$

$$Glucose\ +\ Biomass\ \rightarrow\ Biomass$$

$$Biomass\ \rightarrow\ Biomass\ +\ Antibodies$$

Figure 6.26 Four reaction of CHO culture model

$$\frac{d}{dt}\begin{bmatrix} Biomass(Xv) \\ Glucose(Glc) \\ Lactate(Lac) \\ Glutamine(Gln) \\ Ammonia(Amm) \\ Antibodies(MAb) \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1/Y_{Xv/Glc} & 0 & -1 & 0 \\ Y_{Lac/Glc}/Y_{Xv/Glc} & 0 & Y_{Lac/Glc} & 0 \\ -1/Y_{Xv/Gln} & 0 & 0 & 0 \\ Y_{Amm/Gln}/Y_{Xv/Gln} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} f1 \\ f2 \\ f3 \\ f4 \end{bmatrix}$$

$$f_i\ =\ u_i Xv\ , i = 1,4$$

$$u_1\ =\ \mu_{max}\frac{Glc}{(k_{Glc} + Glc)}\frac{Gln}{(k_{Gln} + Gln)} \qquad (5.3)$$

$$u_2\ = \mu_{d,max}\frac{1}{(\mu_{max} - k_{d,Lac}Lac)}\frac{1}{(\mu_{max} - k_{d,Amm}Amm)}\frac{k_{d,Gln}}{\left(k_{d,Gln} + Gln\right)}$$

$$u_3\ =\ m_{Glc}\frac{Glc}{(k_{m,Glc} + Glc)}$$

$$u_4\ =\ \beta\ + \alpha\frac{u1}{k_{\mu}\ + u1}$$

6.3.1. Complexity analysis

Complexity analysis will be performed in three steps as mentioned in section 6.1.1. These steps consist of performing PCA visualization with convexity calculations, sorted minimization and SOM analysis. Aim of this analysis is to compare its' relative complexity with other cases studies within this work, and to find problems gradient-based algorithms might run into trying to optimize this system.

6.3.1.1. PCA visualization

As this model has sixteen parameters, and high level of complexity it is required to use large number of samples for PCA visualization so first two principal components would be able to represent whole system. For this number of sample points is increased to fifty thousand. Sampling was performed same way as in section 6.1.1.1. First, we can observe large number of failed integration points (figure 6.26), this suggests that there is a lot of discontinuity in the system, leading to gradient search getting stuck very often. To have clearer picture of the error plane of the system failed integration points are removed (figure 6.27). There seem to be no clear valley leading towards global optimum, but there are number of valleys in different places. This means there are local optimums that are in separate valleys from global optimum. Another feature that can be observed is a denser plane of convex and non-convex points around error value of 40^4. This is probably, plane of separation, where majority of local minimums start to diverge. For that reason, low amount points can be observed above this plane, and very high number of points reside on the plane. This signifies a first barrier for optimization algorithm to pass, in order to reach any optimal solution. Combining all these obstacles, system achieves overall convexity of 7%, around three times lower than Monod model (6.2). Overall, it can be concluded that system is of a high complexity level, and involved multiple obstacles, towards global optimum. We can be certain that this representation of search space is accurate as both PC1 and PC2 explains 48.4% each, for a total of 96.8% variability.

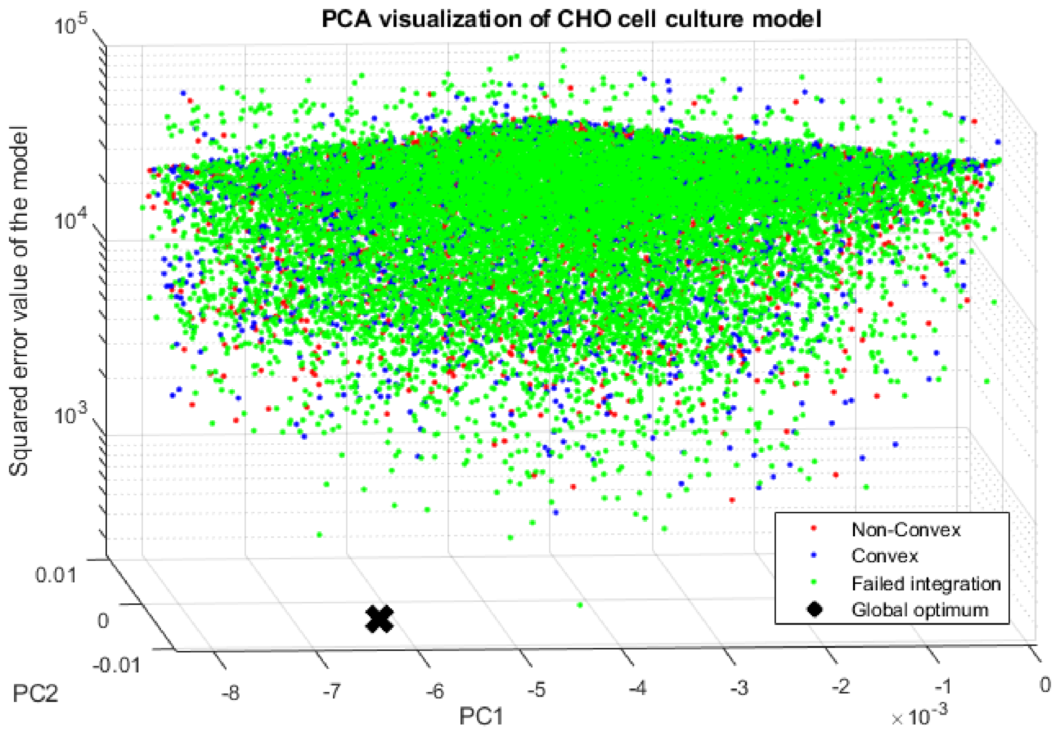


Figure 6.27 PCA error plot for CHO cell culture model (6.3), with colour coded points for convexity, where blue points are convex, red points are non-convex and green points are failed integrations. Black x marks global optimum solution.

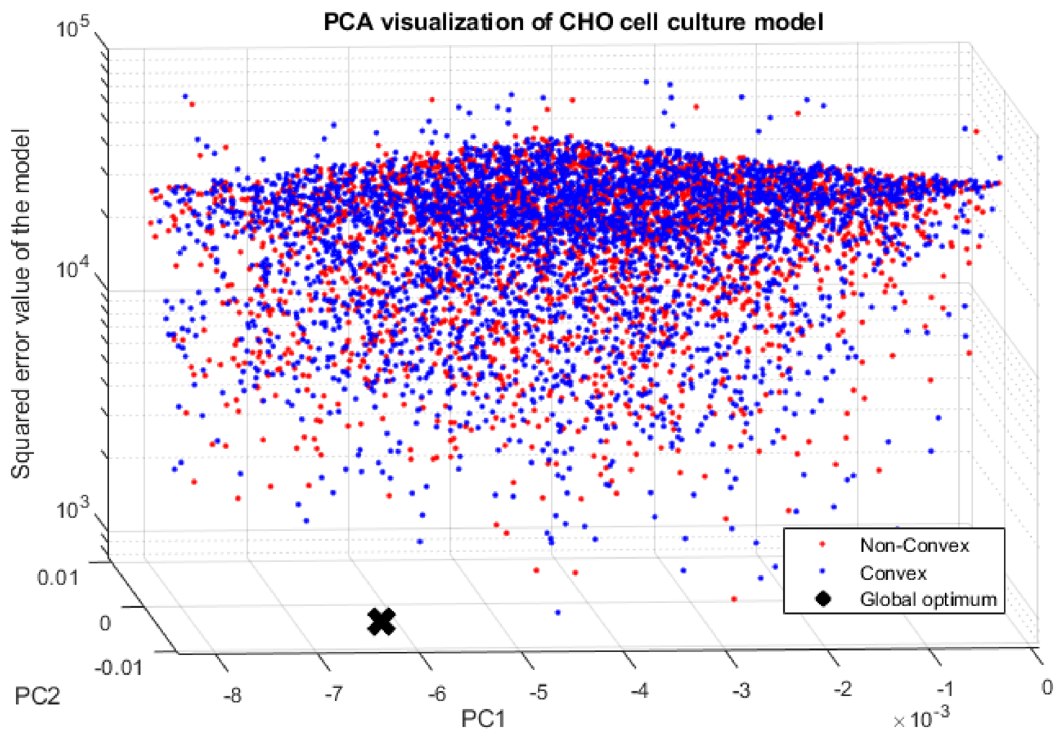


Figure 6.28 PCA error plot for CHO cell culture model (6.3), with colour coded points for convexity, where blue points are convex and red points are non-convex. Black x marks global optimum solution.

6.3.1.2. Sorted minimization

Sorted optimization is performed same way as described in section 4.7. Parameter space is sampled for thousand initial parameter values. When optimizing thousand samples for this system, it shows a lot of different local minimums. Although it would seem, that more than half samples converge to same value (samples from 1000 to around 450), they do not. This is just high error value of 10^{10} that is assigned to failed integrations. Which means over a half of starting positions cannot be optimized at all. This agrees with PCA visualization results which show large amount of failed integration implying there are a lot of discontinuities in this model. This makes model extremely problematic to most optimization techniques. Only about hundred initial points can reach decent error values, of below 10^{-4} which is about 10% of starting points. Points that start in non-convex regions, get to the low error values much faster, as expected. This emphasises how crucial it is to have your starting condition in convex regions for fast and optimal results, when using gradient based optimization algorithms. This sorted optimization graph (figure 6.28), also provides us with information that although the systems parameters are hard to identify, they can be identified correctly, as difference between lowest and highest error points is clearly visible. Still, your result accuracy will be heavily dependent on your starting points.

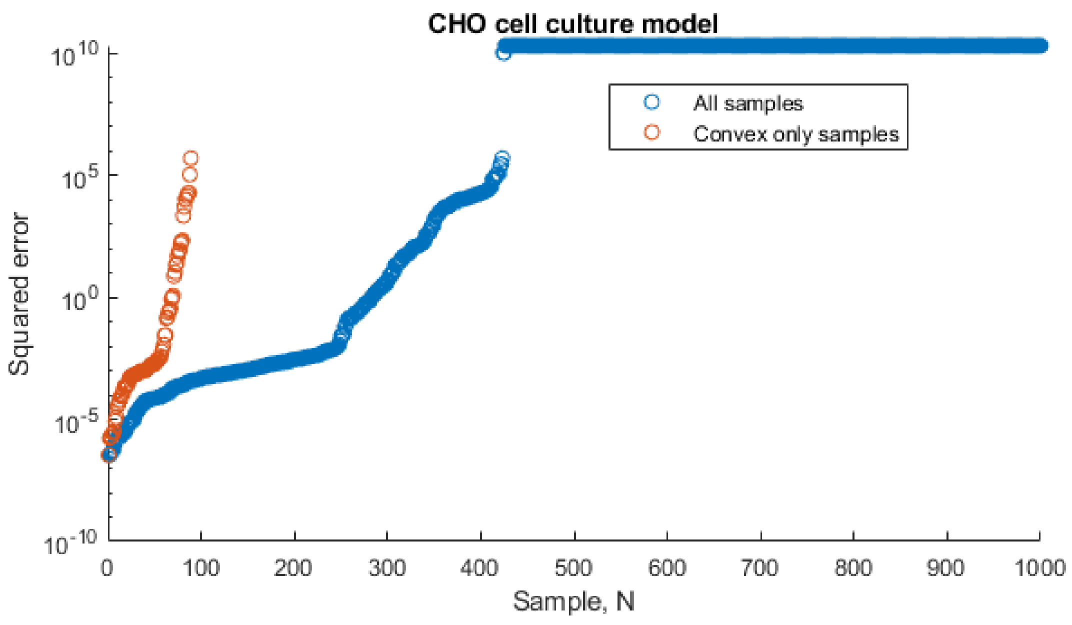


Figure 6.29 CHO cell culture model (6.3) thousand samples sorted optimization, where blue circles is all samples and red circles are convex samples only

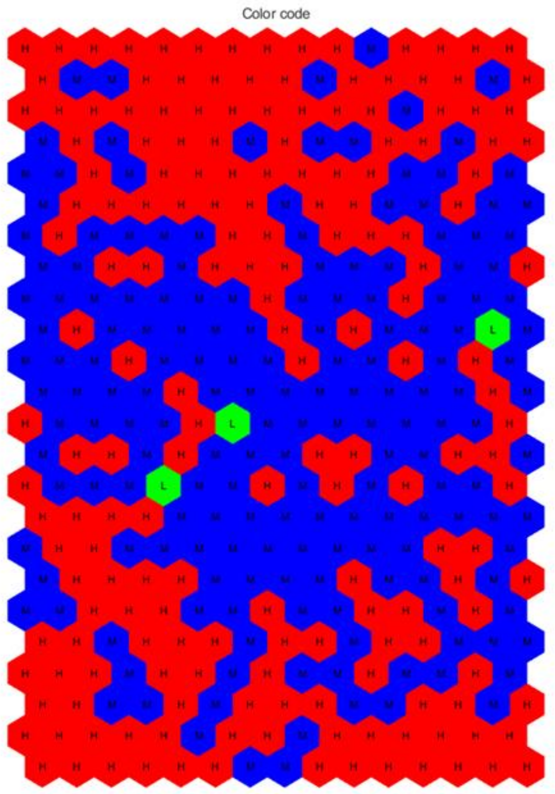
6.3.1.3. SOM analysis

Looking at figure 6.29 results of SOM, shows couple of local minimums scattered around the plane, but if we look at the U-matrix there are much more not as pronounced local minimums scatter all around the place. This is like what we saw in PCA visualization earlier of lots of local minimums scattered all around the plane and several deeper valleys of local minimums which are more distinctive. Although there is a general valley towards the middle, some of the smaller local minimums are even scattered in high error zones. This kind of layout makes gradient-based algorithm easily stuck in wrong local minimums, which there are plenty of. Looking at individual component map hints to couple things, only parameters o11 and o15 have continuous patterns, where all other are chaotic. Continuous patterns were observed with Monod model (6.2), where there was clear dominant variable and identifiable parameters. Whereas chaotic nature of these maps where observed in Polynomial model (6.1), where there was no dominant parameter and parameter values where not identifiable due to nature of the model. This could be due to several factors: a) chaotic component maps are due to large amount of discontinuity in model b) only parameters o11 and o15 are dominant c) large number of parameters leads to chaotic component maps, during the training of SOM.

To see if we can answer why most component maps have no continuous patterns to them, we need to look at importance of factor and how well they match with overall error map. Figures 6.31-6.32 shows how well component maps match up versus total error maps. Positive values mean higher degree of matching, and lower values mean lower. Cross correlation factor of 1, represents perfect match and 0 represent prefect mismatch. Cross correlation maps seem to indicate some parameters are more dominant than others, but not by large amount. This is confirmed by matching factors. Most dominant parameter is o1, that said it is by small margin. It can only be dominant variable because of its low variability. This would imply that with larger number of variables matching factors become more similar making it hard to distinguish dominant variables from the rest. Maximum difference between matching factors is 0.05 and 0.03, respectively (table 6.35). This is a very small difference, but it has low variability in comparison with Polynomial model (6.1).

Parameters	Encoded parameters	Positive Matching Factor	Negative Matching Factor
μ_{max}	o1	0.68 ± 0.01	0.64 ± 0.02
k_{Glc}	o2	0.63 ± 0.01	0.66 ± 0.01
k_{Gln}	o3	0.66 ± 0.03	0.63 ± 0.02
$\mu_{d,max}$	o4	0.65 ± 0.02	0.63 ± 0.02
$k_{d,Lac}$	o5	0.66 ± 0.03	0.63 ± 0.03
$k_{d,Amm}$	o6	0.65 ± 0.02	0.65 ± 0.02
$k_{d,Gln}$	o7	0.65 ± 0.01	0.64 ± 0.01
$Y_{Xv/Glc}$	o8	0.64 ± 0.01	0.66 ± 0.01
$Y_{Lac/Glc}$	o9	0.65 ± 0.02	0.63 ± 0.02
m_{Glc}	o10	0.65 ± 0.04	0.65 ± 0.03
$k_{m,Glc}$	o11	0.65 ± 0.01	0.64 ± 0.02
$Y_{Xv/Gln}$	o12	0.65 ± 0.02	0.65 ± 0.01
$Y_{Amm/Glc}$	o13	0.65 ± 0.01	0.64 ± 0.02
β	o14	0.63 ± 0.02	0.66 ± 0.03
α	o15	0.61 ± 0.01	0.66 ± 0.03
k_{μ}	o16	0.64 ± 0.02	0.65 ± 0.01

Table 6-37 Table of positive and negative mean matching factor of each parameter for system (6.3)



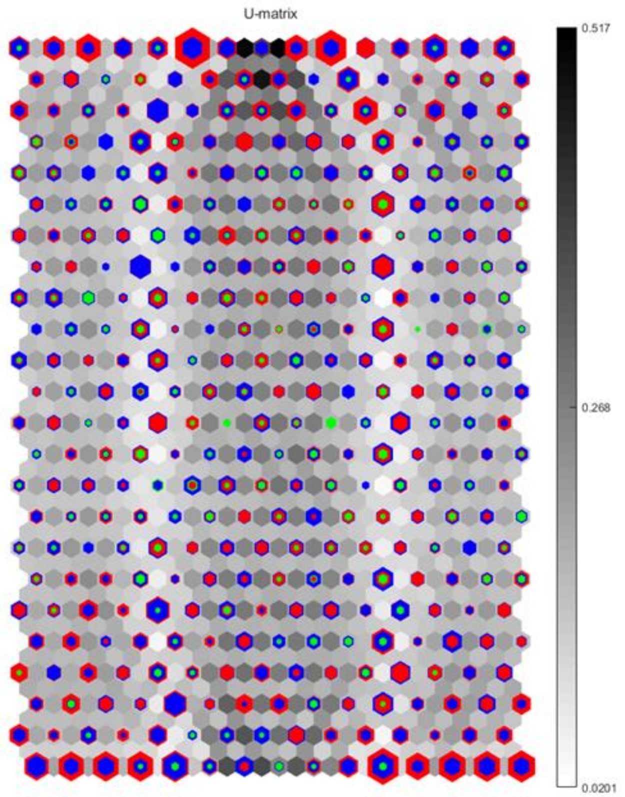


Figure 6.30 Left 2d SOM of CHO cell culture model (6.3), colour coded based of model error, where green (L) is low error, blue (M) is medium error, and red (H) is high error. Right 2d SOM with same colour code as left, but also showing relative distance in n-dimensional plane represented as colour bar and number of individual members of each cluster.

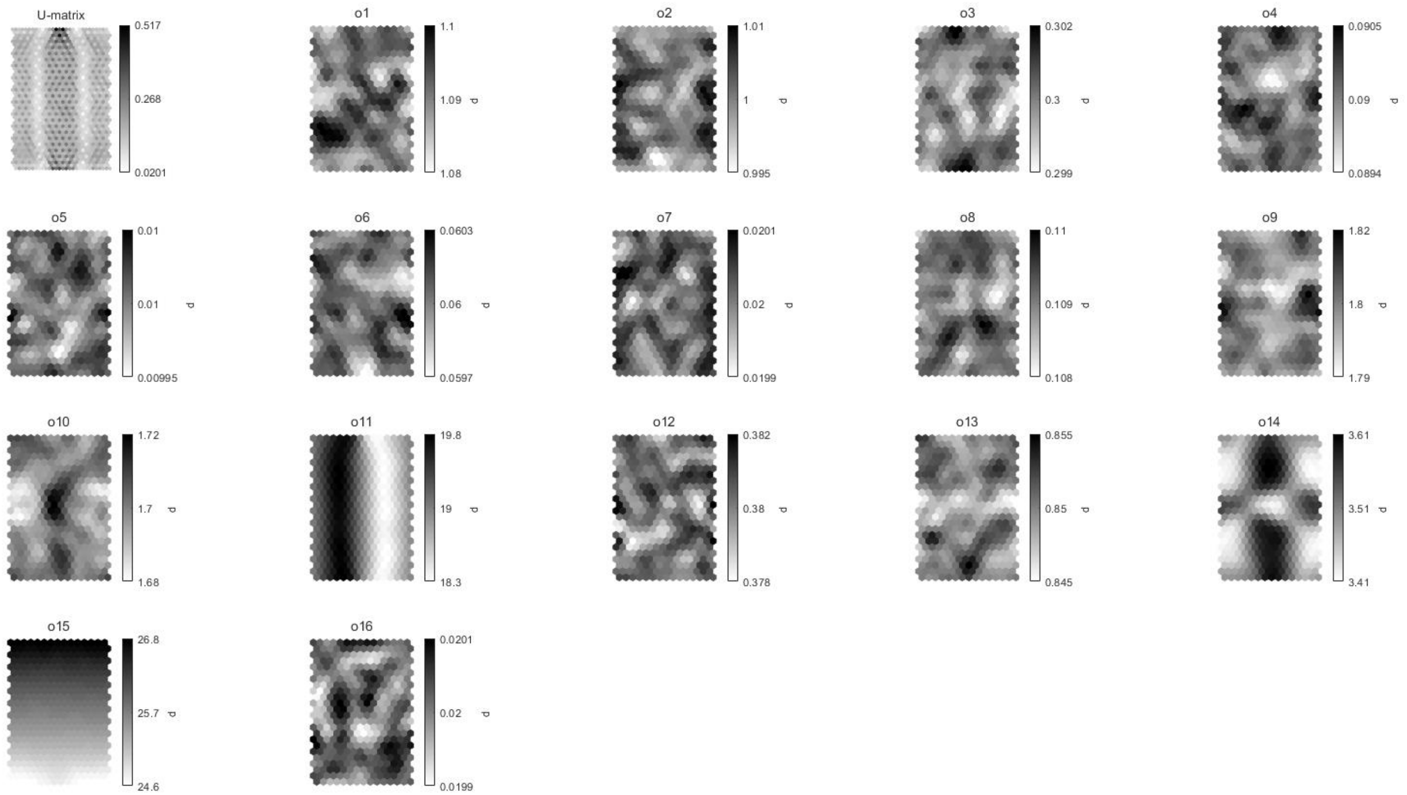
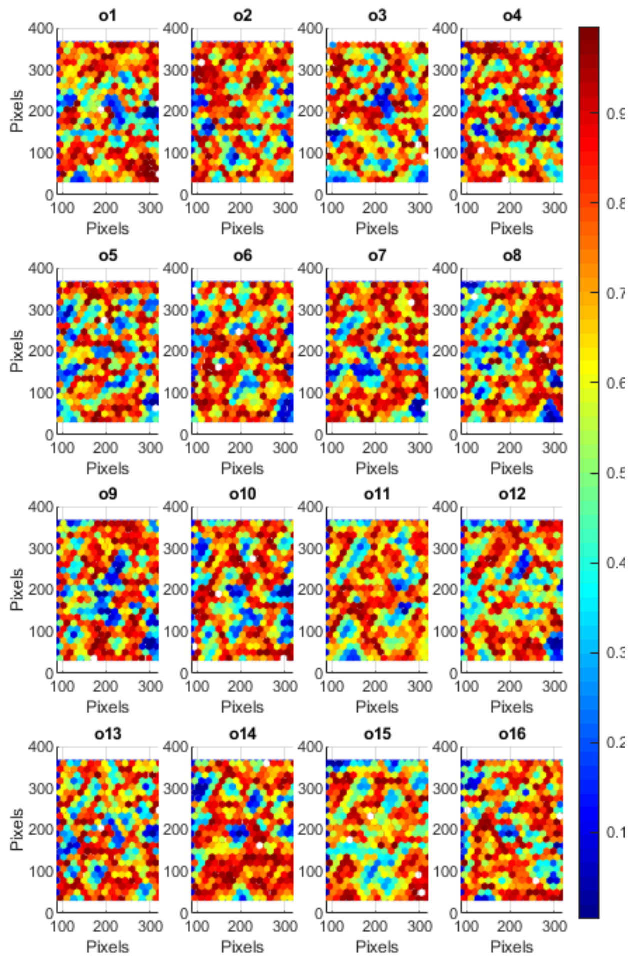


Figure 6.31 U-matrix represents N-dimensional plane as 2d SOM with relative distances as colour bar, followed by each component n-dimensional plane of its value distribution represented as colour bar.

a)



b)

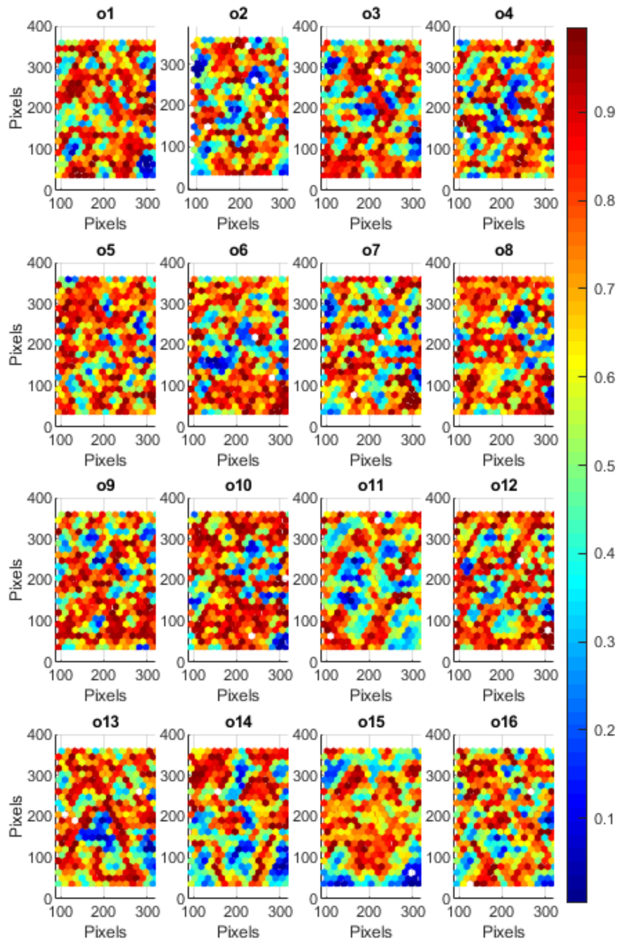


Figure 6.32 a) Positive cross-correlation between each parameter map and overall error map. b) Negative cross-correlation between each parameter map and overall error map.

6.3.2. *Model hierarchy*

The CHO cell culture model consists of six coupled ODE's, which when applied proposed state substitution methods' decoupling algorithm leave six independent sub-sets that can be solved in any order. Yet specific solution hierarchy will lead to better results, because of the different levels of sensitivity from the parameters. To figure out best hierarchy of the model we need to use additional tools like bi-partite chart (figure 6.34) and SOM analysis of parameter importance (figures 6.32-31, table 6.35). When combined these tools reveal which order of solving individual subsets will lead best results. Bi-partite chart of the CHO cell culture model shows that there are three most important states biomass, glucose, and glutamine. This separates six subsets into two levels of biomass, glucose, glutamine and lactate, ammonia, antibodies. SOM component analysis was only able to confirm that o1 parameter is most dominant one, this means we would need to solve for o1 first to pass it on for best results. Parameter o1 can be calculated from any of the six subsets, but it is best to use biomass to calculate o1, because o1 is specific growth rate of biomass. This would separate model hierarchy into three levels, by raising biomass one level above both substrates. Lastly there is a problem of parameter ratios, we need to make sure that where parameter ratios appear one of the ratio components is already known. Ratios within CHO cell culture model are as follows: a) o1/o8 b) o1/o12 c) o9/o8 d) o13/o12. Luckily with three level approach this is not a problem as o1 is solver in level one, then o8 and o12 can be solved in level two, finally with o8 and o12 known we can solve for o9 and o13 in level 3.

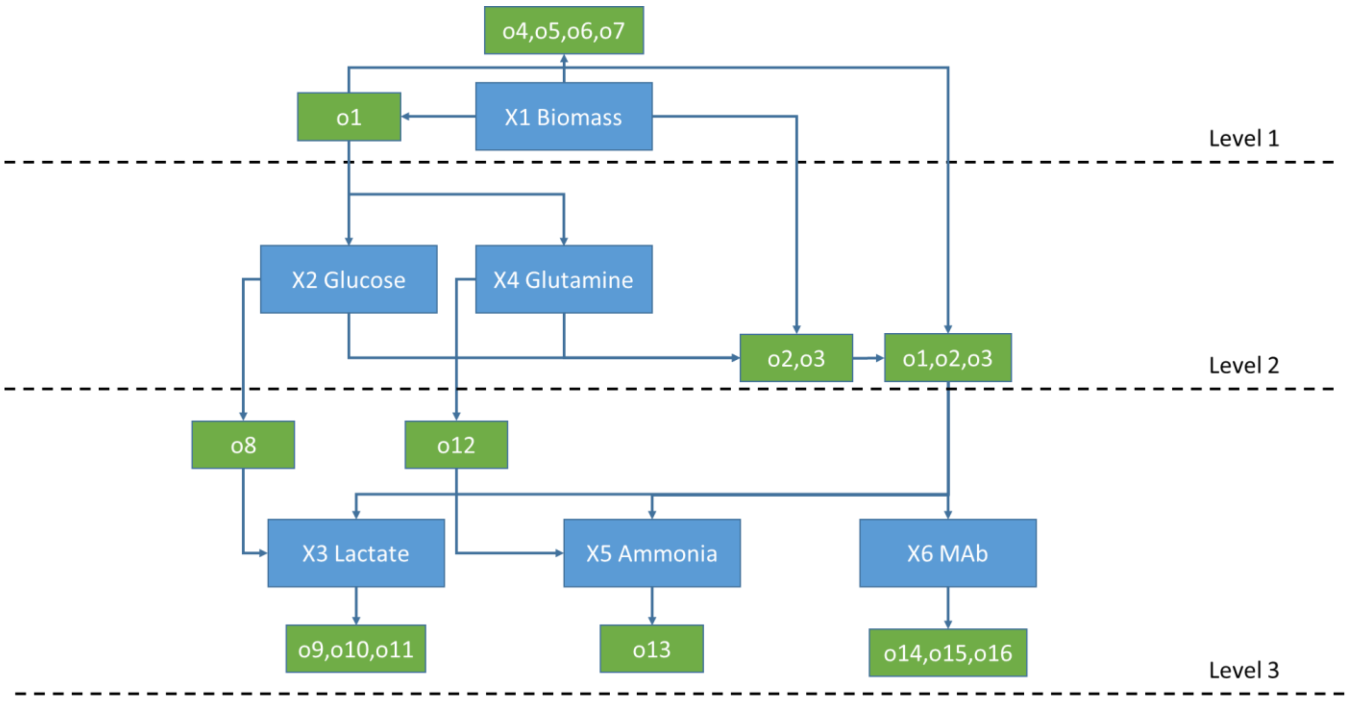


Figure 6.33 Hierarchy of the CHO cell culture model (6.3)

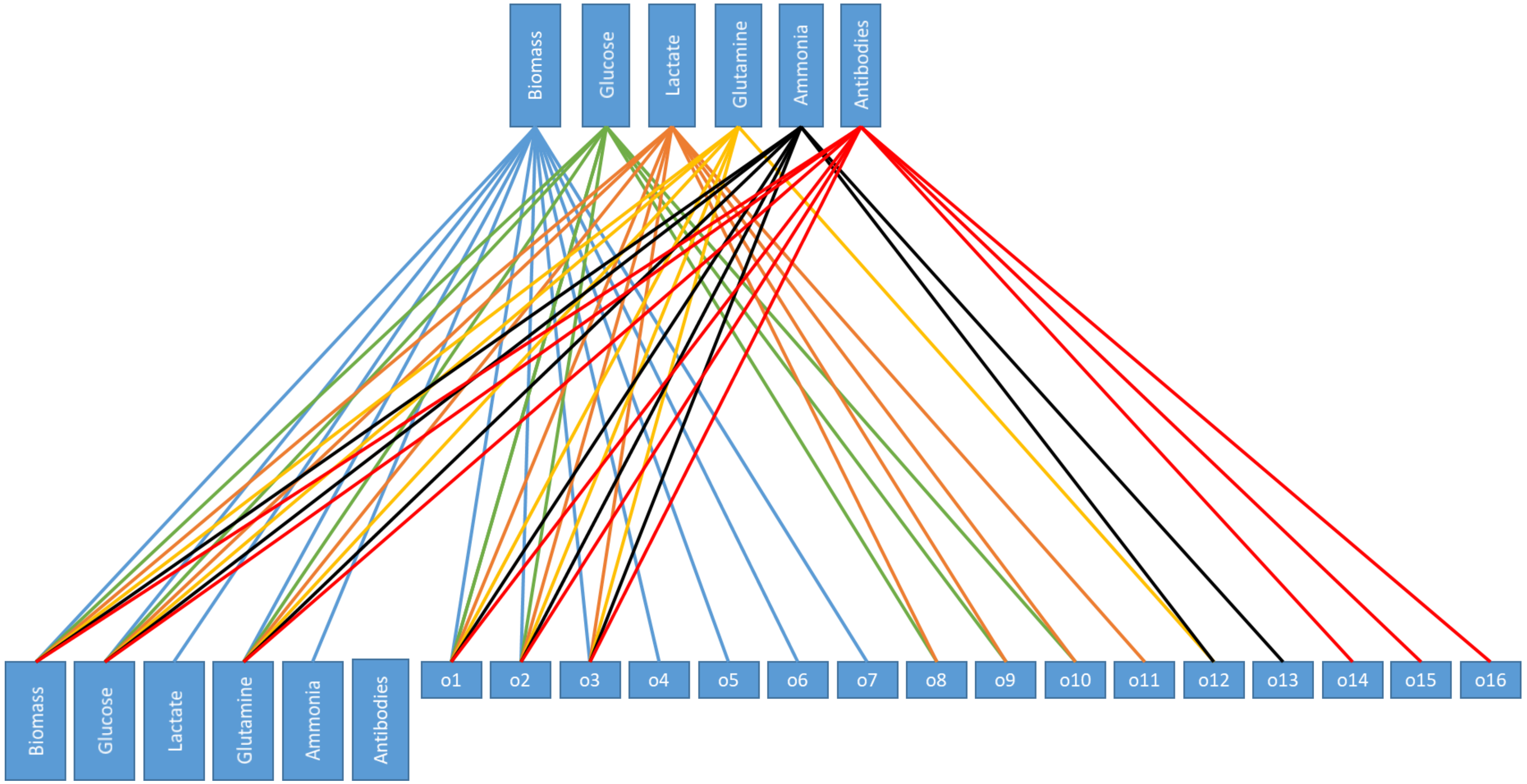


Figure 6.34 Bipartite chart of the CHO cell culture model (6.3)

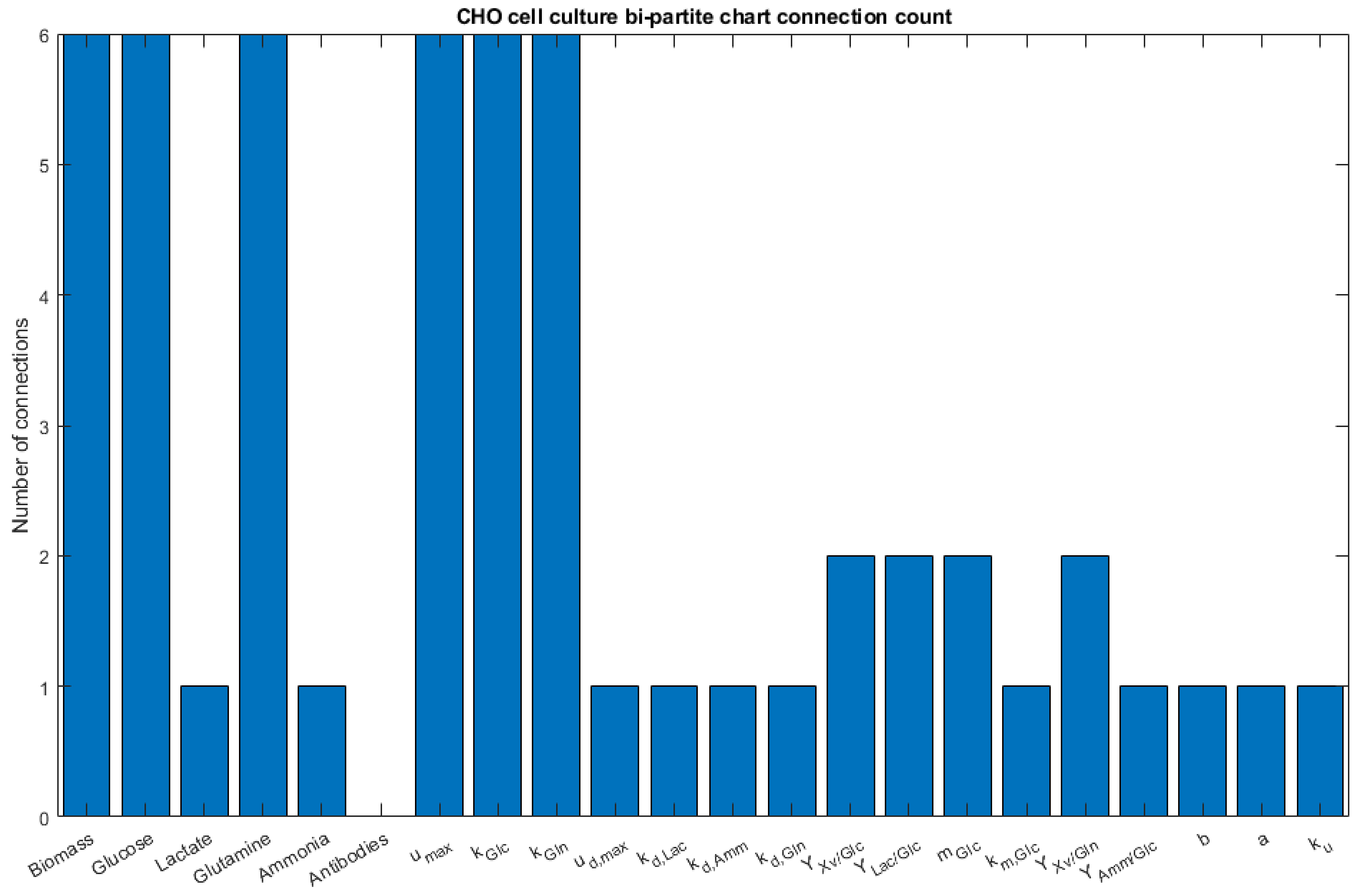


Figure 6.35 Bar graph of connections of CHO culture bipartite chart

6.3.3. Method comparison

To evaluate the proposed state substitution method, it will be compared to the two existing methods discussed in chapter 3. The derivative estimation and Latin-hyper cube sampled multi start method. Two main criteria will be compared, model performance accuracy and the computational time between each of the methods. Although the proposed state substitution method aims to reduce computational time, model performance accuracy is also very important and cannot be completely neglected. Each method will be assessed with different random noise levels and sampling times. For each method three different state initial conditions are compared, first two (Pink and Blue) are set same for all methods and are conditions that were provided for the optimisation algorithm and third initial condition (Black) is different from the first two initial conditions and was never seen by algorithm before. This allows to check method accuracy with unseen data sets, which are within same boundary conditions. Only biomass data set will be presented for model performance, as biomass dictates accuracy for the rest of the states. This will allow to avoid unnecessary graphs while still presenting enough evidence about model performance. However, for completeness first experiment will show predictions for all states Modelling conditions and parameter search spaces are summarised in tables below.

Experiment number	**Sampling Rate**	**Noise level**	**Initial conditions Pink**	**Initial conditions Blue**	**Initial conditions Black**
1	0.1h	5%	0.3, 0.3, 0.3	0.3, 0.3, 0.3	0.2, 0.2, 0.2
2	0.1h	10%	0.3, 0.3, 0.3	0.3, 0.3, 0.3	0.2, 0.2, 0.2
3	0.3h	5%	0.3, 0.3, 0.3	0.3, 0.3, 0.3	0.2, 0.2, 0.2
4	0.3h	10%	0.3, 0.3, 0.3	0.3, 0.3, 0.3	0.2, 0.2, 0.2
5	0.3h	20%	0.3, 0.3, 0.3	0.3, 0.3, 0.3	0.2, 0.2, 0.2

Table 6-38 Summary of modelling conditions for the system (5.3)

Parameters	Lower bound	Upper bound
μ_{max}	0	2
k_{Glc}	0	2
k_{Gln}	0	1
$\mu_{d,max}$	0	0.5
$k_{d,Lac}$	0	0.5
$k_{d,Amm}$	0	0.5
$k_{d,Gln}$	0	0.5
$Y_{Xv/Glc}$	0	0.5
$Y_{Lac/Glc}$	0	5
m_{Glc}	0	5
$k_{m,Glc}$	0	50
$Y_{Xv/Gln}$	0	1
$Y_{Amm/Glc}$	0	1
β	0	10
α	0	50
k_{μ}	0	0.5

Table 6-39 Parameter search space for the system (6.2)

6.3.3.1. Experiment 1 0.1h sampling and 5% random noise

When we compare computational times (table 6.37) we can see that although the proposed state substitution method is slower than the derivative estimation method it is faster than the Latin hyper cube method. The proposed state substitution method is faster by 49% compared to the Latin hyper cube method and slower by 32% than the Derivative estimation method.

Method	Computational time, s
Derivative estimation	1371s
Latin hyper cube multi-start	3109s
Proposed state substitution method	1887s

Table 6-40 Computational time of all three methods for experiment 1

Although the proposed state substitution method is slower than the Derivative estimation method it is considerably more accurate, when predicting all three data sets. The Latin hyper cube method seems to be able to predict general trend right, but its accuracy is not as good as in previous case studies. This is most likely is caused by CHO cell culture model having a lot of discontinuities in error plane. Main advantage of the Latin hyper cube method is that is has large number of starting positions, but when discontinuities separate whole error plane into small pieces, getting good starting location becomes difficult and unreliable.

Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	9.66E-03	2.21E-02	3.53E-03
Latin hyper cube multi-start	1.24E-02	7.91E-03	1.32E-02
Proposed state substitution method	1.76E-02	2.22E-04	5.33E-04

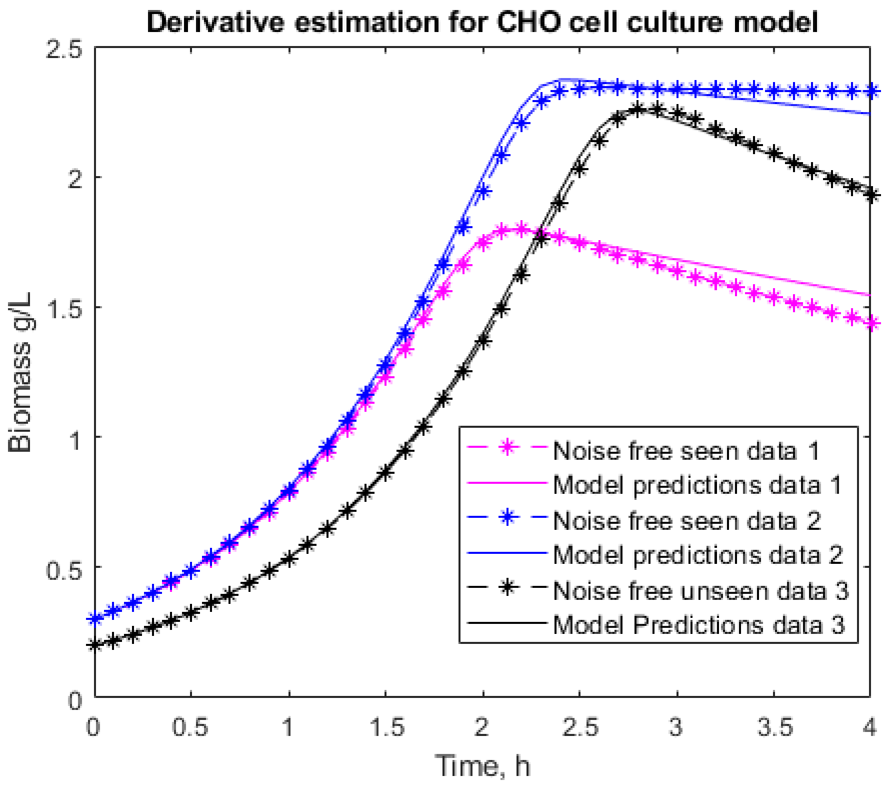
Table 6-41 Squared error values of each method and each data set for experiment 1

As complexity analysis suggested this model (5.3) is considerably harder to optimize than previous two cases studies, and it is confirmed by much higher computational time in all three methods. Due to increased complexity proposed state substitution method outperforms Latin hyper cube method in terms of speed and accuracy. Nevertheless, it must sacrifice some of its speed to maintain accuracy making it solver than the Derivative estimation method. This leads to think that for initial optimization the Derivative estimation method could be better choice provided noise levels and sampling time is low.

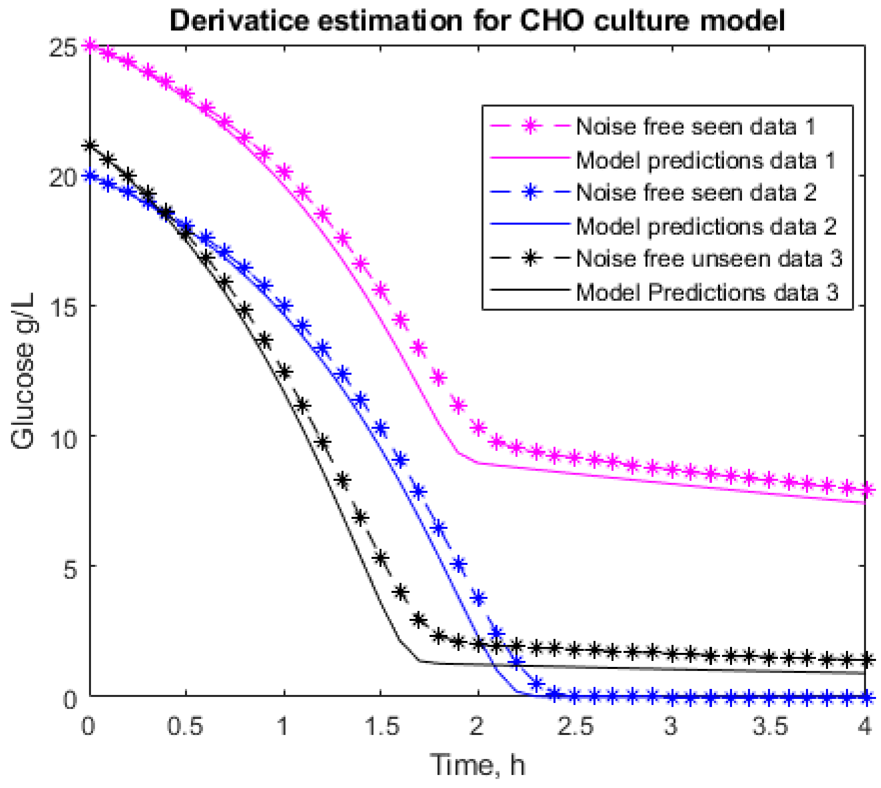
When comparing identified parameter values, we observe that parameter values that are closest to their theoretical values are o1, o5, o8, o9, o12, o13 and o15. Out of all these variables only o1 was picked up to be dominant variable in SOM component analysis, confirming that with large number of variables it is very hard to pick all dominant variables using this technique. On other hand we can see that parameters o10 and o11 have very large variation across all methods implying they have very low impact over the system.

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
μ_{max}	1.09	1.04	1.12	1.01
k_{Glc}	1.00	0.79	0.91	0.56
k_{Gln}	0.30	0.11	0.27	0.18
$\mu_{d,max}$	0.09	0.00	0.11	0.02
$k_{d,Lac}$	0.01	0.03	0.01	0.02
$k_{d,Amm}$	0.06	0.19	0.00	0.09
$k_{d,Gln}$	0.02	0.50	0.50	0.01
$Y_{Xv/Glc}$	0.11	0.10	0.12	0.11
$Y_{Lac/Glc}$	1.80	1.74	1.83	1.81
m_{Glc}	1.70	2.17	0.79	3.27
$k_{m,Glc}$	19.00	33.10	1.94	46.75
$Y_{Xv/Gln}$	0.38	0.37	0.41	0.37
$Y_{Amm/Glc}$	0.85	0.87	0.82	0.86
β	3.50	6.76	2.87	2.67
α	25.70	21.03	25.73	27.56
k_{μ}	0.02	0.00	0.01	0.00

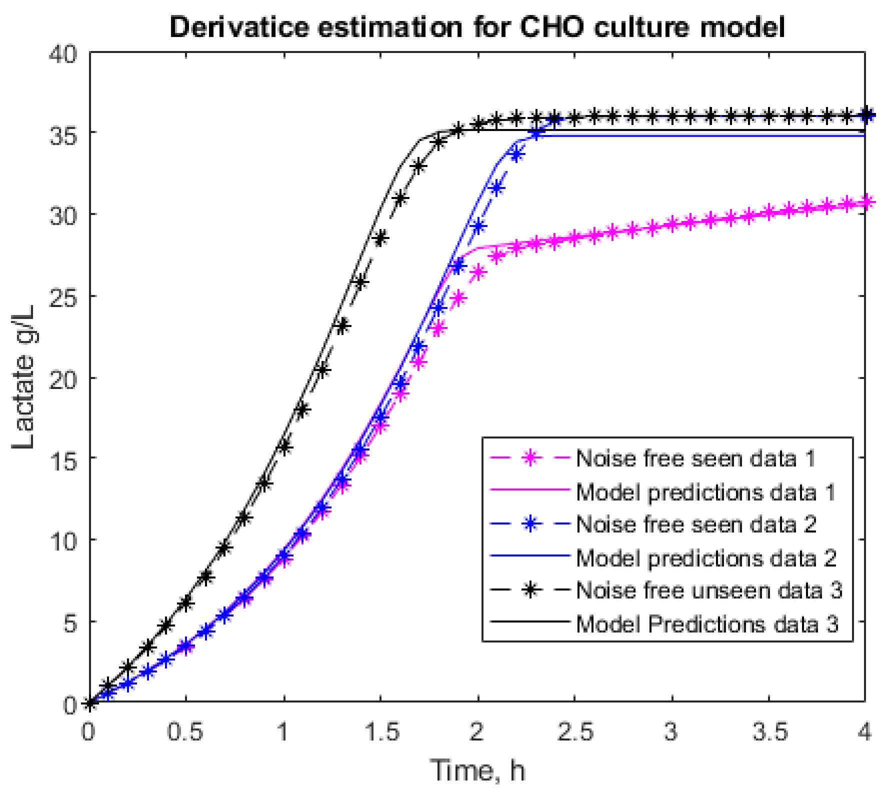
Table 6-42 Summary of identified parameter values for each method for experiment 1

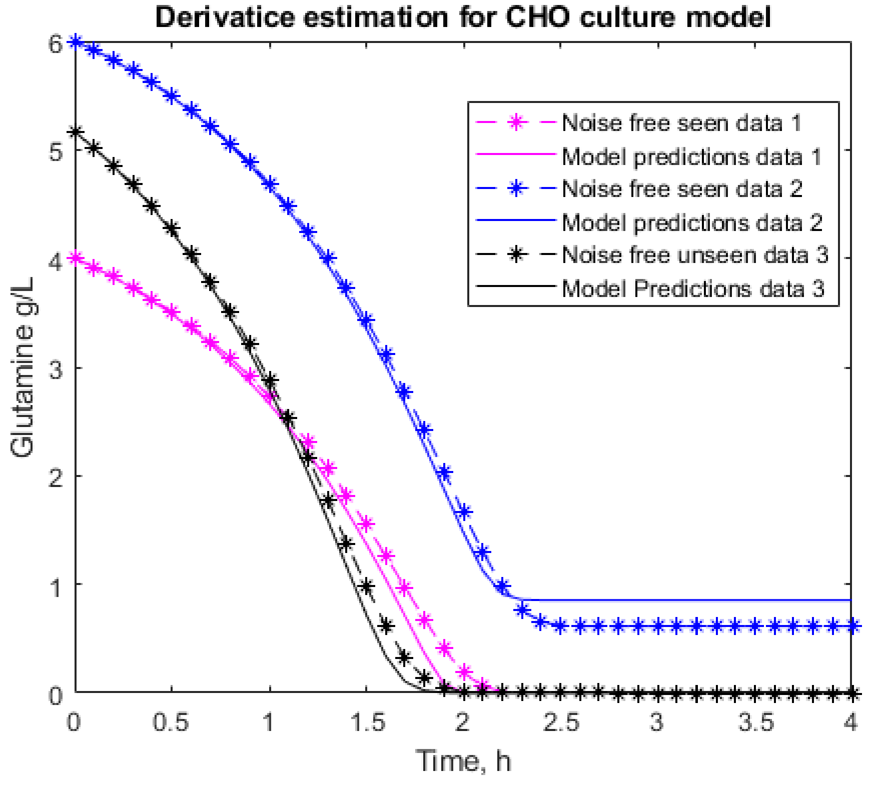


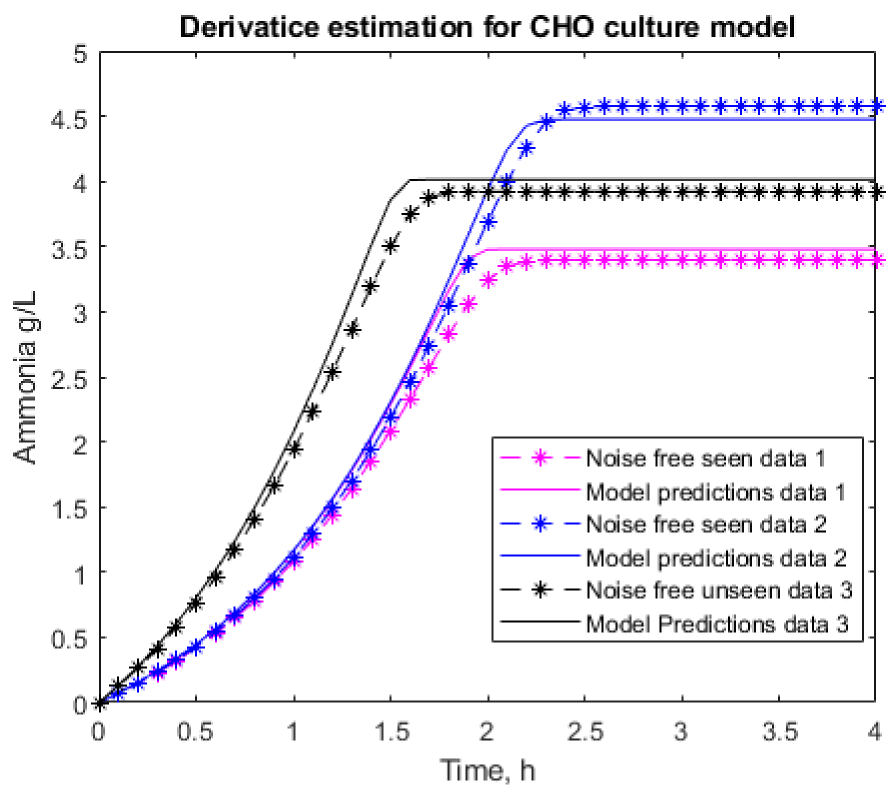
Derivative estimation for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3



Derivatice estimation for CHO culture model
Glucose g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Derivatice estimation for CHO culture model
Lactate g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Derivatice estimation for CHO culture model
Glutamine g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3



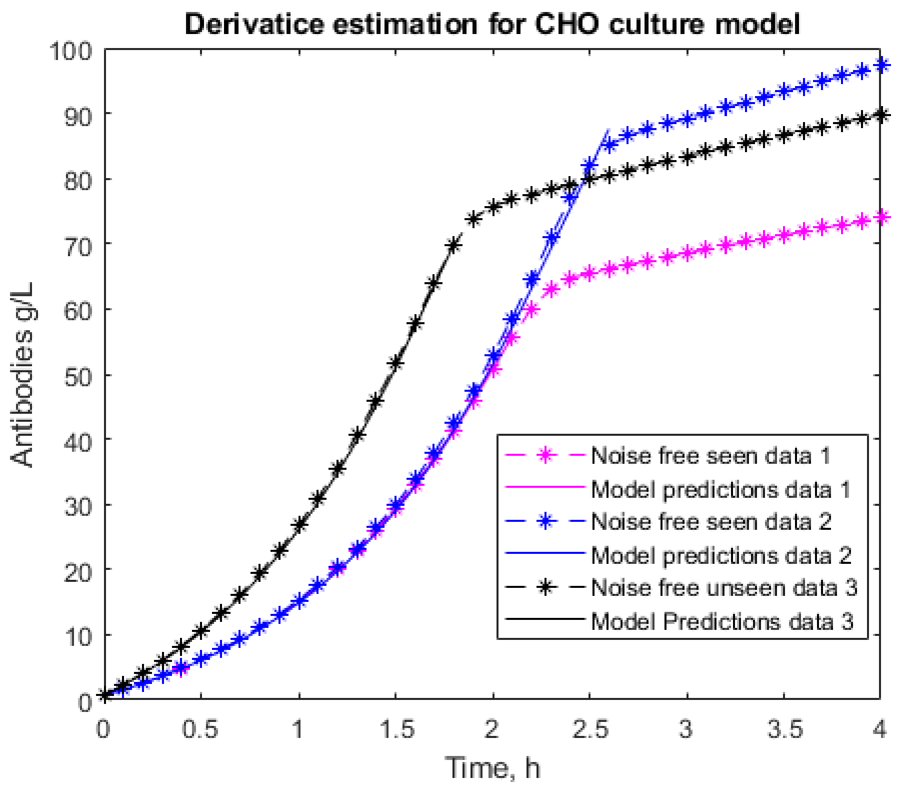
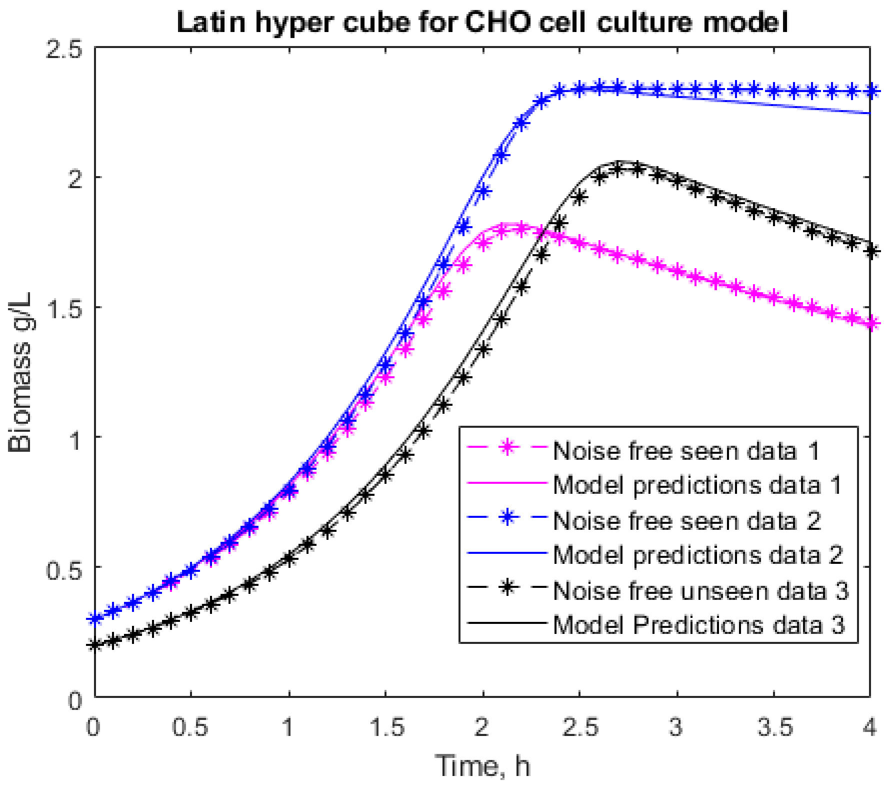
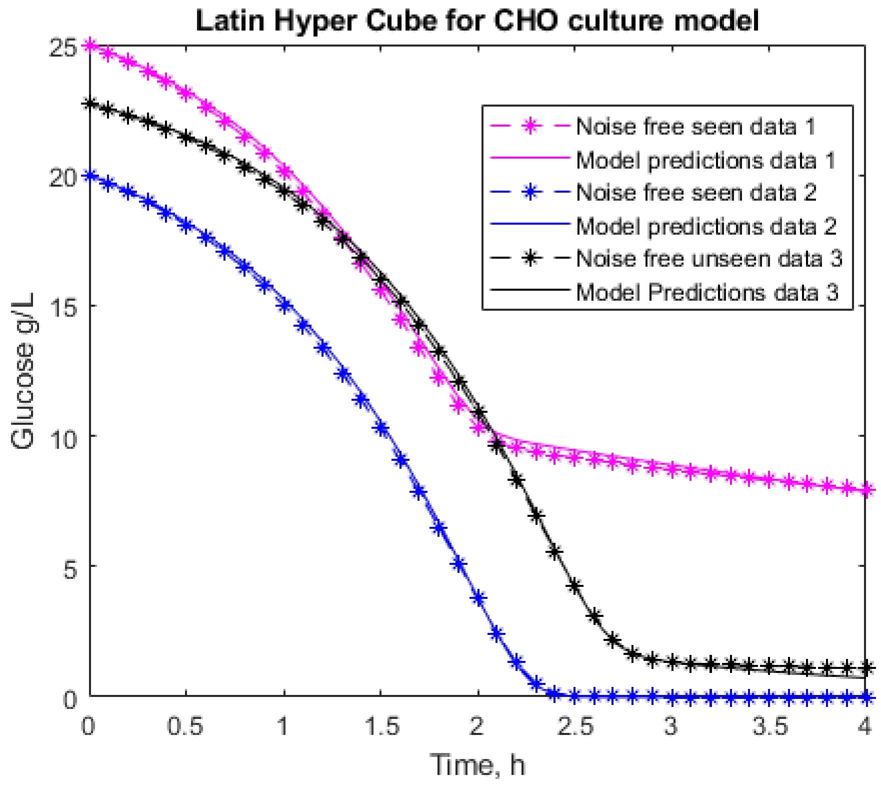
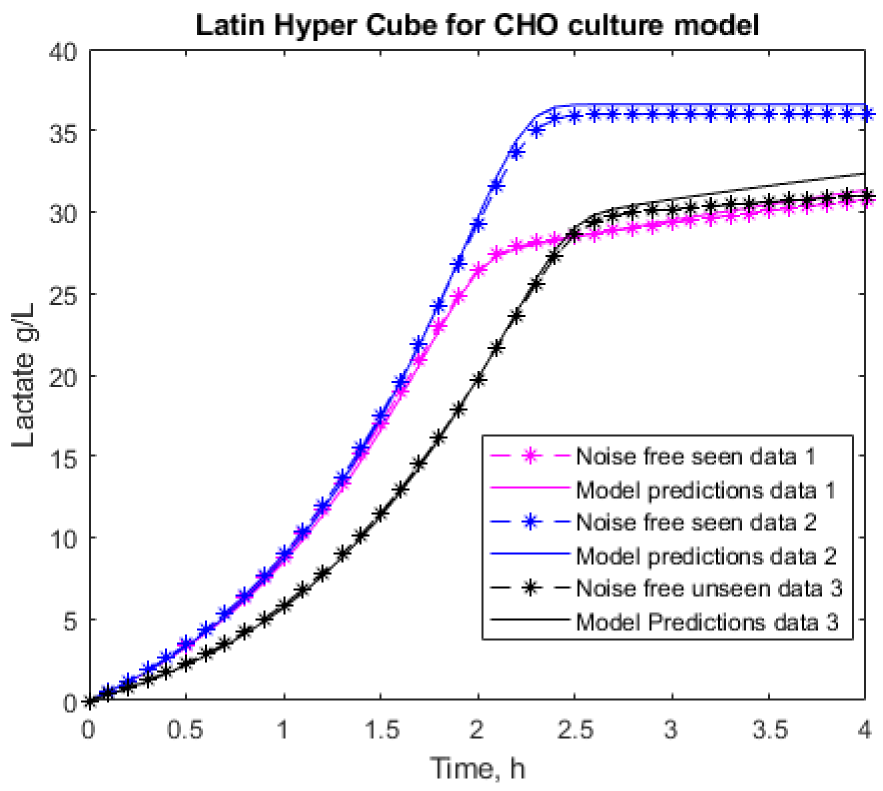
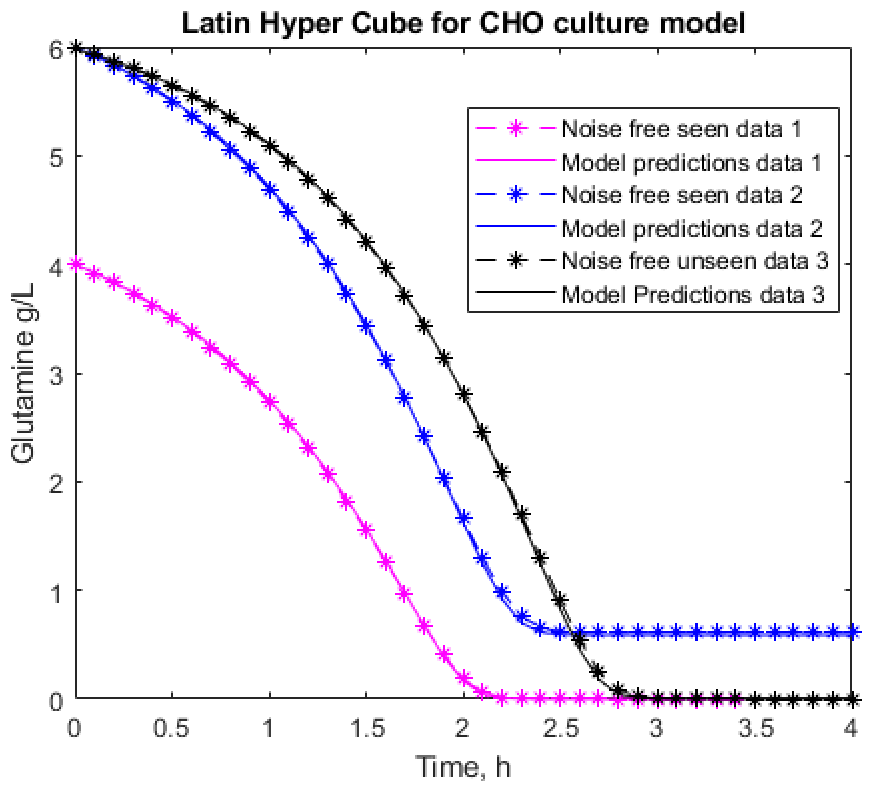


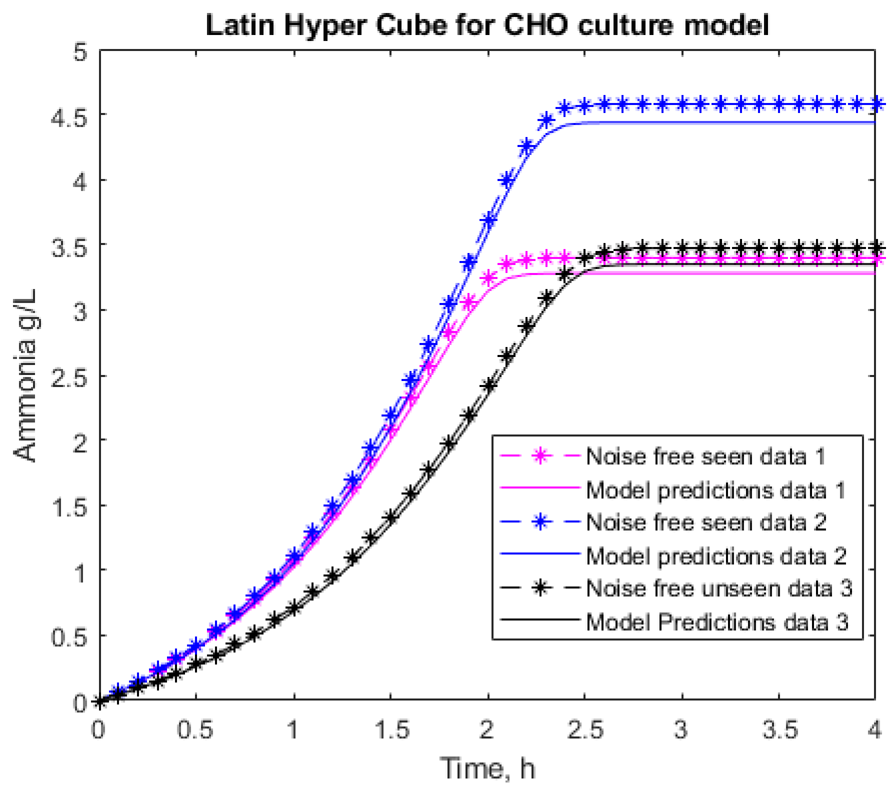
Figure 6.36 Performance results for Derivate estimation with 0.1h sampling and 5% random noise for measured data.


Latin hyper cube for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Latin Hyper Cube for CHO culture model
Glucose g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Latin Hyper Cube for CHO culture model
Lactate g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3
0
5
10
15
20
25
30
35
40
0
0.5
1
1.5
2
2.5
3
3.5
4


Latin Hyper Cube for CHO culture model
Glutamine g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3
0
1
2
3
4
5
6
0
0.5
1
1.5
2
2.5
3
3.5
4



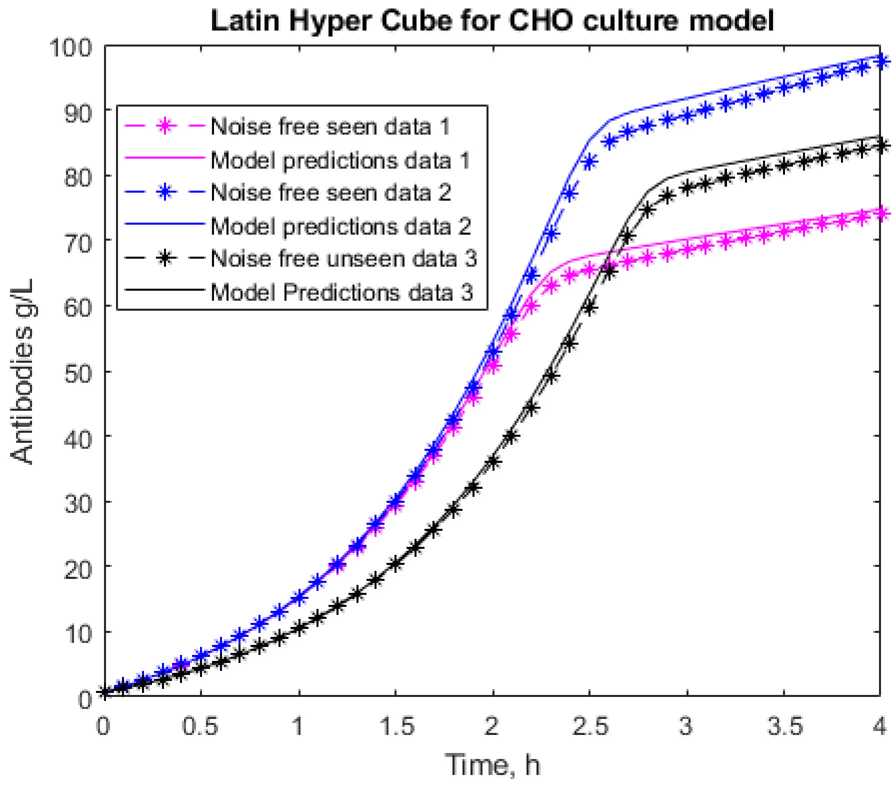
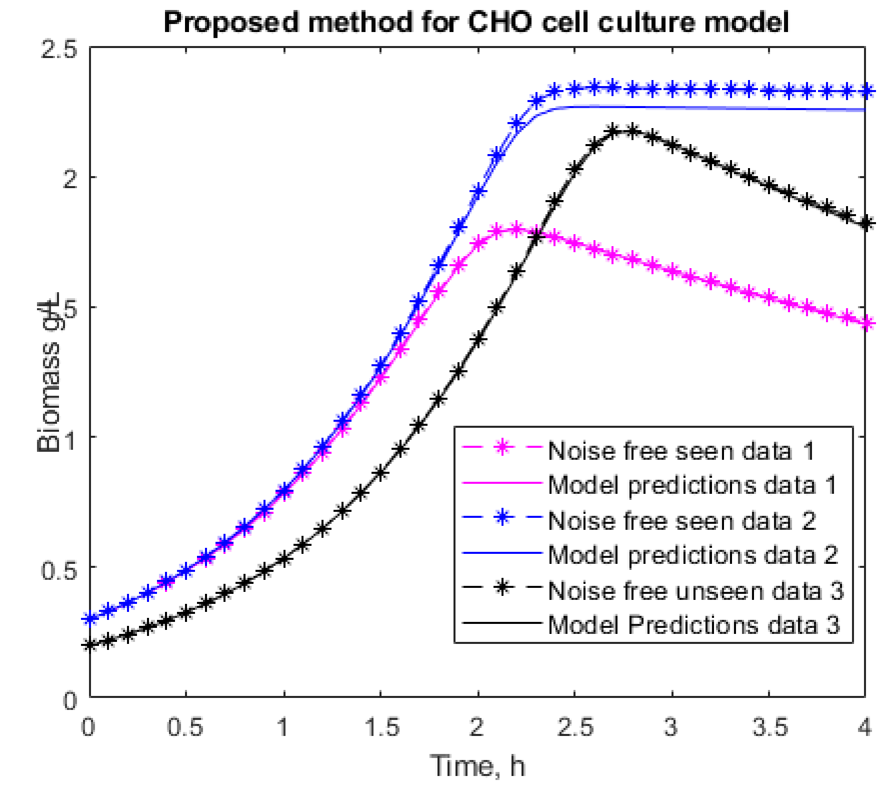
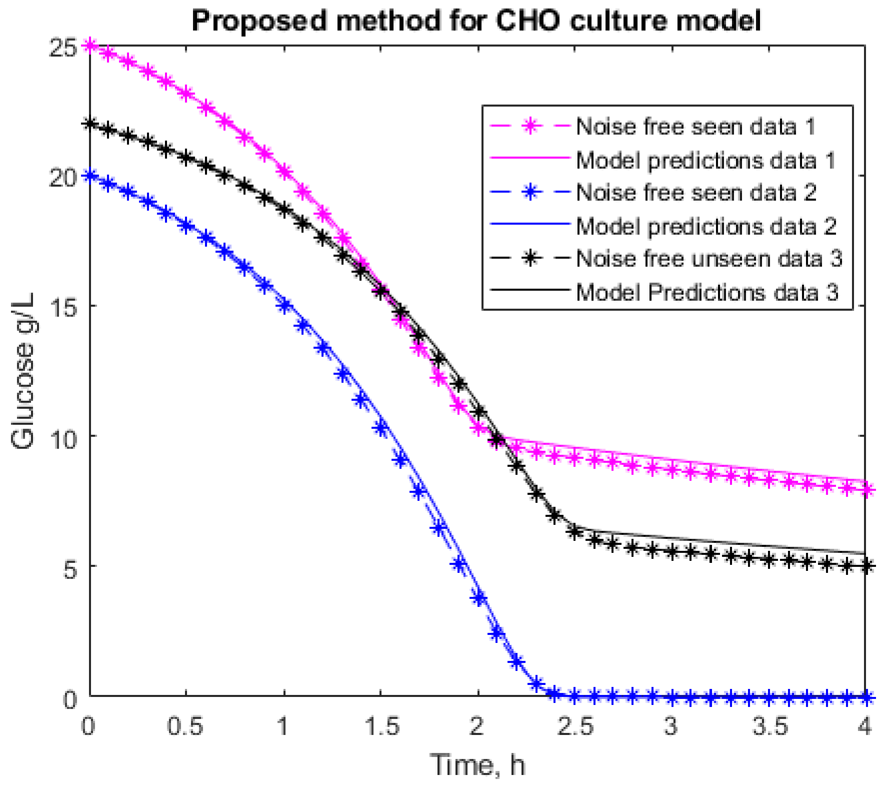


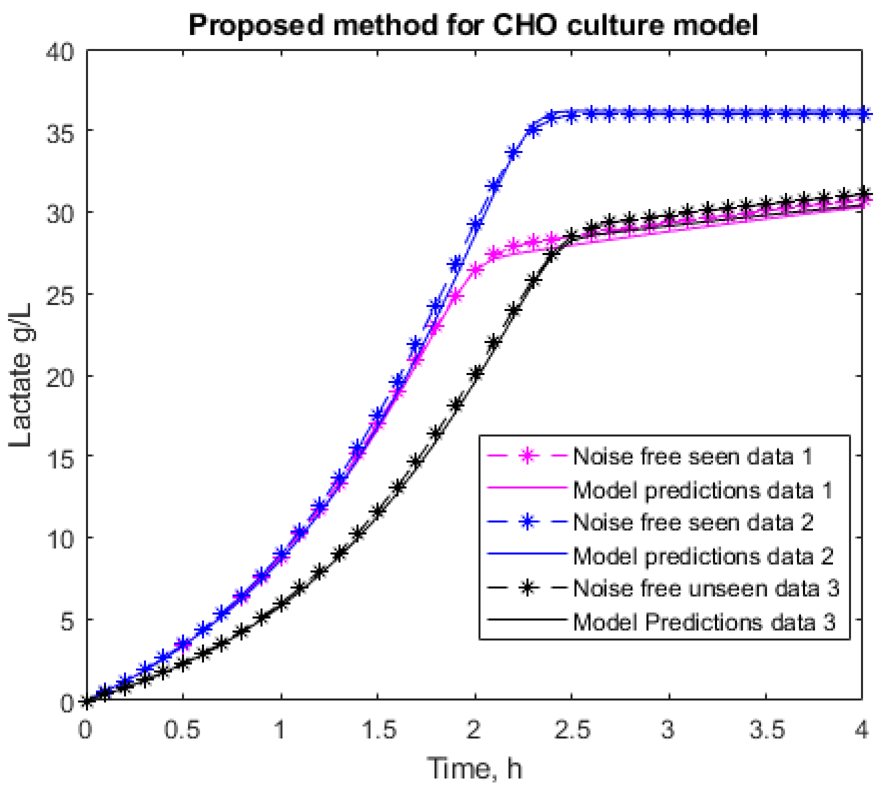
Figure 6.37 Performance results for Latin hyper cube method with 0.1h sampling and 5% random noise for measured data.

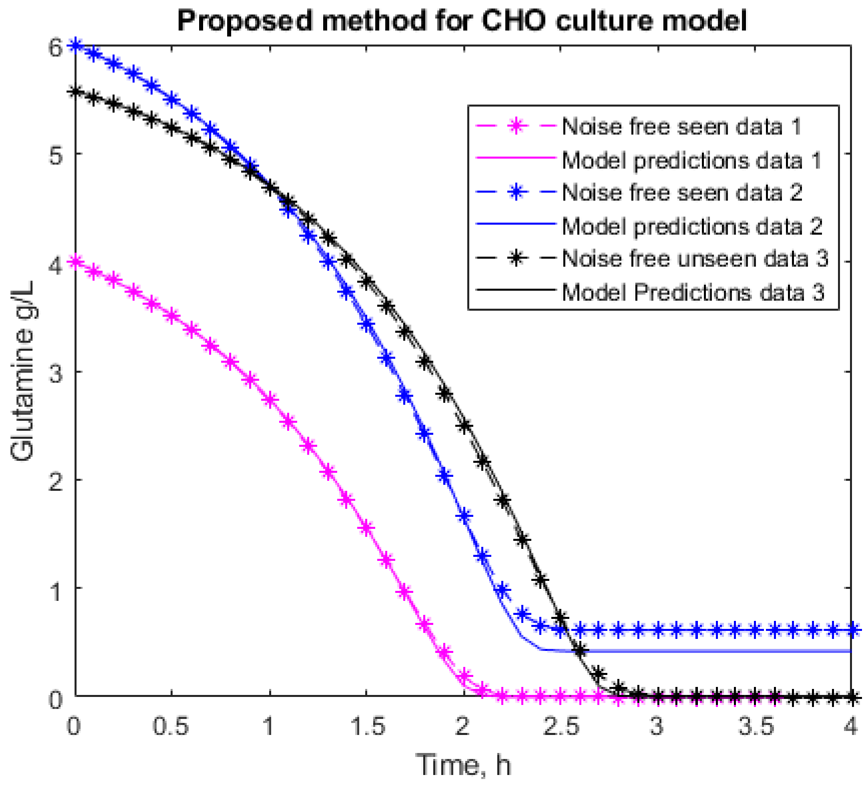


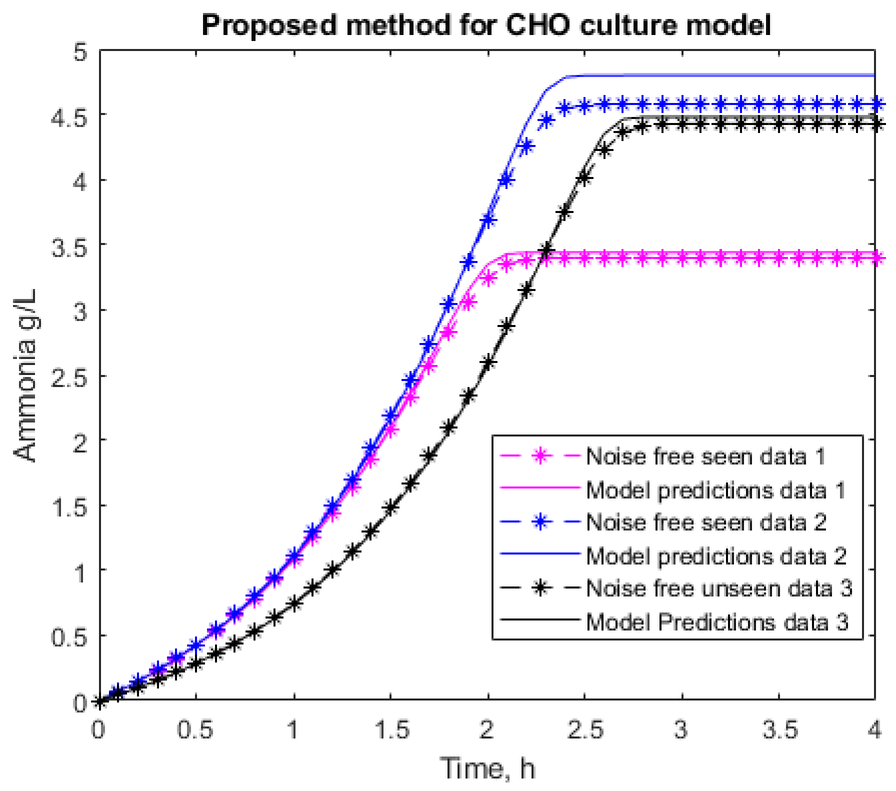
Proposed method for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3



Proposed method for CHO culture model
Glucose g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Proposed method for CHO culture model
Lactate g/L
Time, h
0
0.5
1
1.5
2
2.5
3
3.5
4
5
10
15
20
25
30
35
40
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Proposed method for CHO culture model
Glutamine g/L
Time, h
0
0.5
1
1.5
2
2.5
3
3.5
4
1
2
3
4
5
6
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3



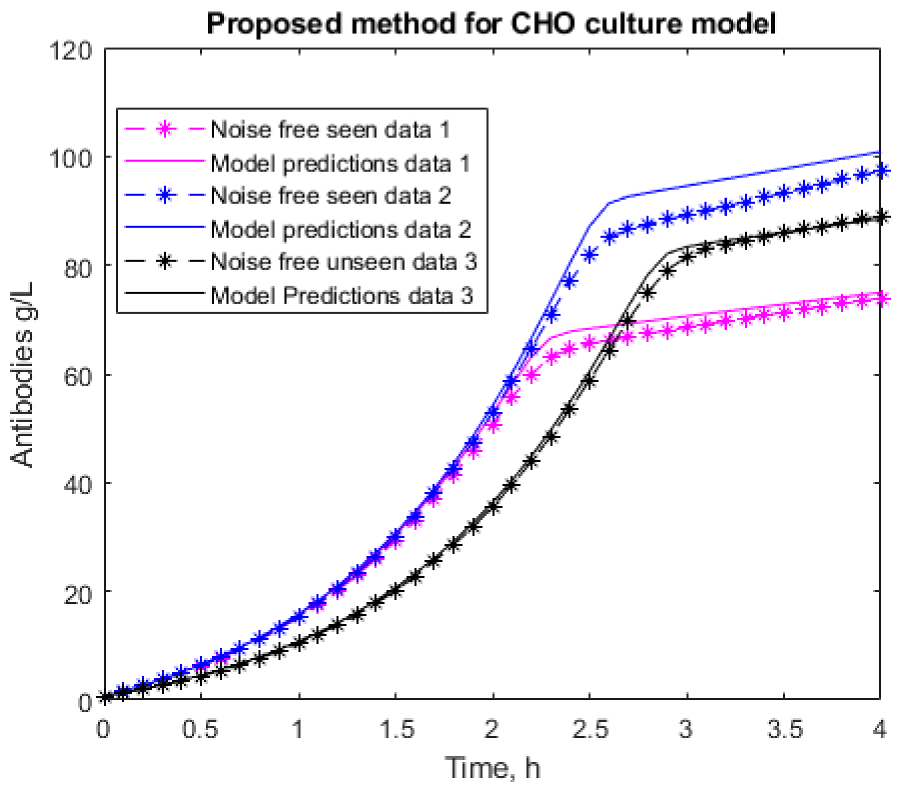


Figure 6.38 Performance results for the proposed state substitution method with 0.1h sampling and 5% random noise for measured data.

6.3.3.2. Experiment 2 0.3h sampling and 5% random noise

Surprisingly, the Derivative estimation method is slowest out of three methods in this experimental setup. It is slower than proposed state substitution method by 98%, and slower than the Latin hyper cube method by 32%. This is most likely cause, by method being at its critical point, where it can still predict general trend of the model, but it is close to its capability limit, if this is the case it should be expected that with increased noise the Derivative estimation method should stop being able to predict the model. On the other hand, proposed state substitution method is faster than the Latin hyper cube method by 72%.

Method	**Computational time, s**
Derivative estimation	4180s
Latin hyper cube multi-start	3021s
Proposed state substitution method	1426s

Table 6-43 Computational time of all three method for experiment 2

When comparing sq. Error for each method (table 6.41) it looks like accuracy is similar, between the proposed and the Latin hyper cube methods, and the Derivative estimation method falls short. It is expected because of two factors: a) increased sampling time, has large negative impact on methods performance b) as discussed before it looks like method is at its critical point. Other two methods seem to deal just fine with this experimental setup, implying they are more effected by noise levels, than sampling time.

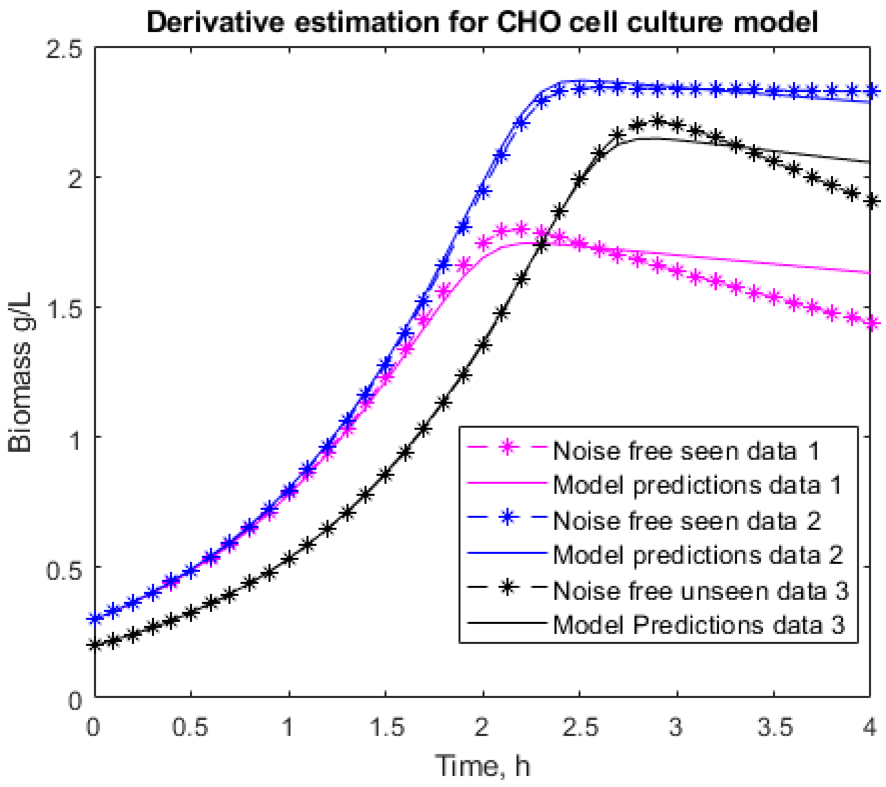
Method	**Sq. Error (Blue)**	**Sq. Error (Pink)**	**Sq. Error (Black)**
Derivative estimation	4.41E-03	4.08E-02	4.03E-02
Latin hyper cube multi-start	3.70E-05	8.01E-04	6.44E-04
Proposed state substitution method	4.20E-05	1.03E-03	1.19E-03

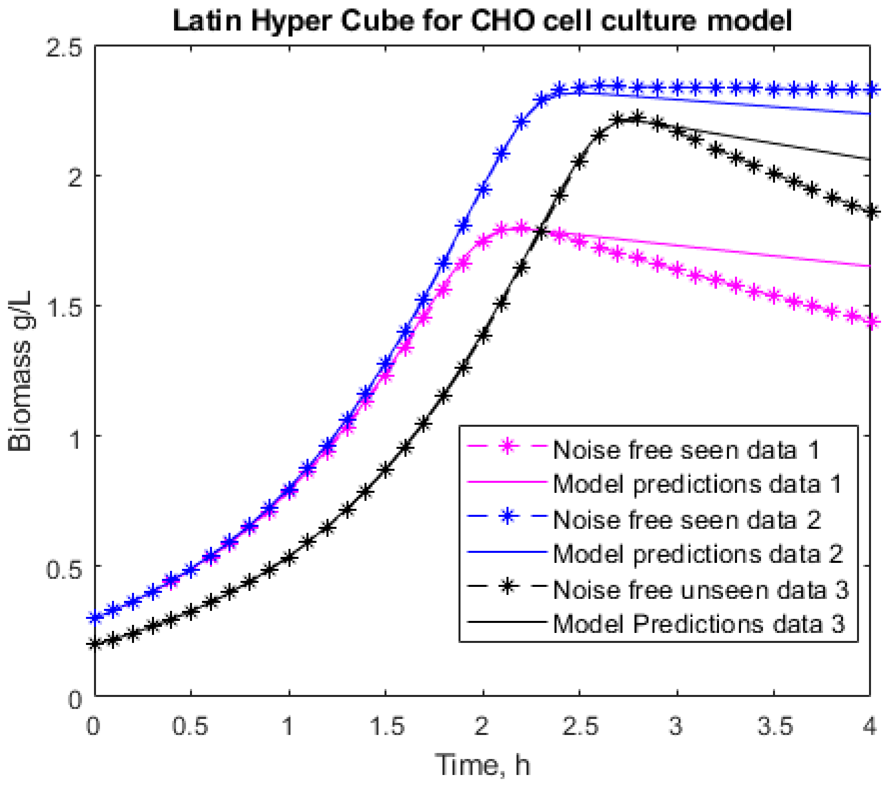
Table 6-44 Squared error values of each method and each data set for experiment 2

Computational time decreased for all methods except the Derivative estimation method. This is due to increased sampling time, which in turn decreases amount of data required to process. When comparing identified parameter values, parameters o1, o5, o8, o9, o12, o13 and o15 are still consistence between all three methods and parameter o10 and o11 vary a lot.

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
μ_{max}	1.09	1.09	1.09	1.15
k_{Glc}	1.00	0.62	1.06	1.37
k_{Gln}	0.30	0.38	0.26	0.39
$\mu_{d,max}$	0.09	0.04	0.03	0.02
$k_{d,Lac}$	0.01	0.00	0.02	0.03
$k_{d,Amm}$	0.06	0.00	0.01	0.00
$k_{d,Gln}$	0.02	0.50	0.50	0.00
$Y_{Xv/Glc}$	0.11	0.11	0.11	0.11
$Y_{Lac/Glc}$	1.80	1.79	1.78	1.82
m_{Glc}	1.70	4.76	2.41	0.78
$k_{m,Glc}$	19.00	50.00	37.16	0.91
$Y_{Xv/Gln}$	0.38	0.37	0.38	0.38
$Y_{Amm/Glc}$	0.85	0.82	0.84	0.85
β	3.50	3.76	3.97	3.40
α	25.70	25.15	24.39	25.00
k_{μ}	0.02	0.02	0.01	0.02

Table 6-45 Summary of identified parameter values for each method for experiment 2


Derivative estimation for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Latin Hyper Cube for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3

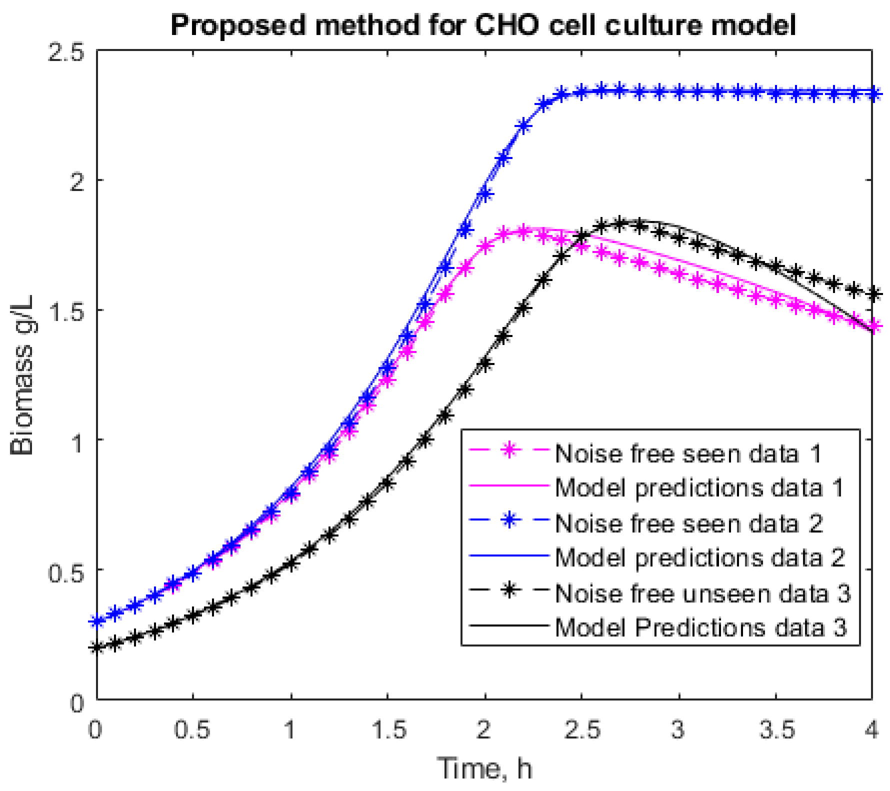


Figure 6.39 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.3h sampling and 5% random noise for measured data.

6.3.3.3. Experiment 3 0.1h sampling and 10% random noise

Similarly, as in experiment 1 when we compare computational times (table 6.43) we can see that although the proposed state substitution method is slower than the derivative estimation method it is faster than the Latin hyper cube method. The proposed state substitution method is faster by 12% compared to the Latin hyper cube method and slower by 86% than the Derivative estimation method.

Method	**Computational time, s**
Derivative estimation	1933s
Latin hyper cube multi-start	5482s
Proposed state substitution method	4848s

Table 6-46 Computational time of all three method for experiment 3

When comparing sq. Error for each method (table 6.44) it looks like accuracy is similar, but when comparing performances (figure 6.37), proposed state substitution method predicts trends of model much more accurately. The proposed state substitution method ability to break down complex problem into smaller sub-sets to solve initially seems to lead much better results accuracy wise when it comes to high complexity models. Although general trend is still predicted by the Derivative estimation and the

Latin hyper cube methods, they accuracy seem to deteriorate heavily with increased noise. Furthermore, most inaccuracy seem to appear in unseen data (Black) for all three methods and in glucose limiting data set (Pink) for the Derivative estimation and Latin hyper cube methods.

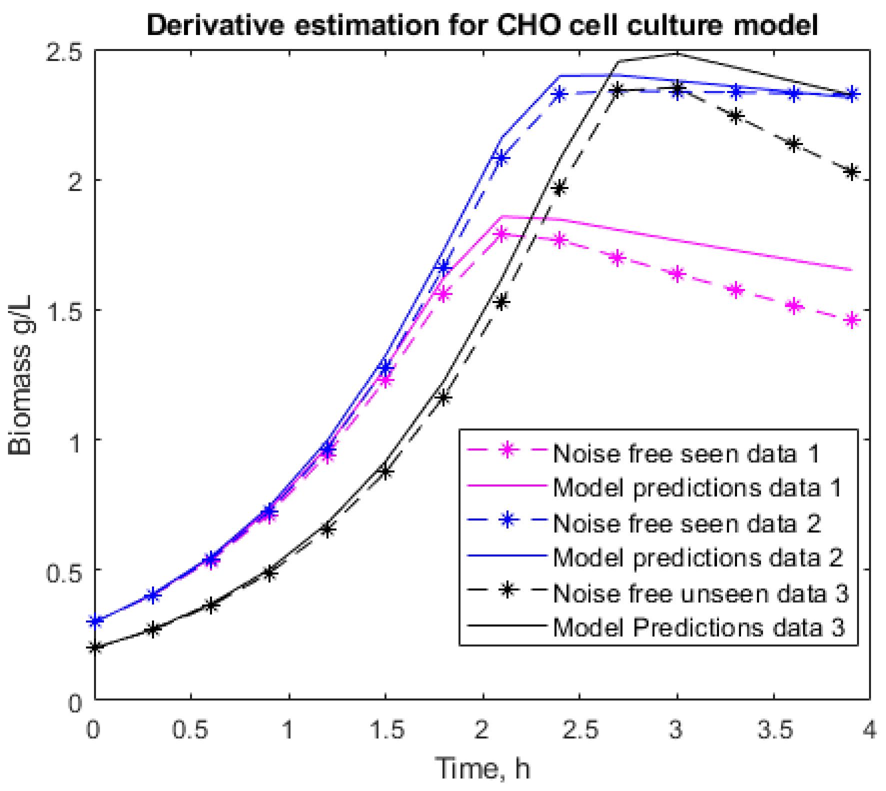
Method	**Sq. Error (Blue)**	**Sq. Error (Pink)**	**Sq. Error (Black)**
Derivative estimation	2.78E-03	6.93E-02	1.66E-02
Latin hyper cube multi-start	1.14E-02	8.92E-02	3.69E-02
Proposed state substitution method	2.65E-03	9.89E-03	1.68E-02

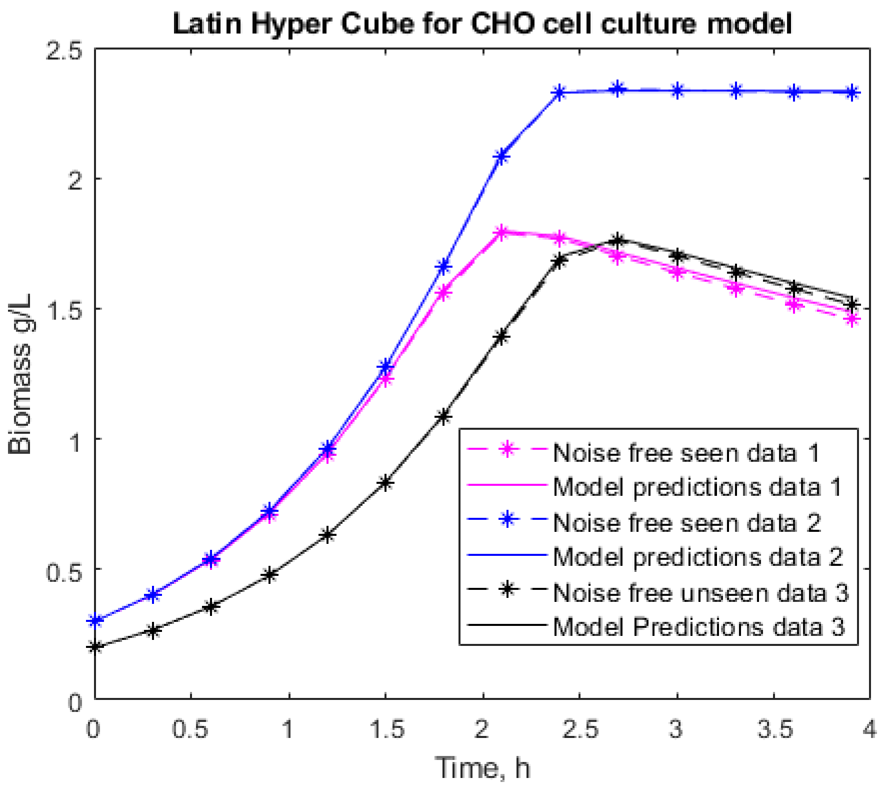
Table 6-47 Squared error values of each method and each data set for experiment 3

Computational time increased for all methods as expected due to increase in noise level, therefore increase in uncertainty optimization algorithm must deal with. When comparing identified parameter values, parameters o1, o5, o8, o9, o12, o13 and o15 are still consistence between all three methods and parameter o10 and o11 vary by large amount.

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
μ_{max}	1.09	1.14	1.07	1.07
k_{Glc}	1.00	1.30	0.89	0.86
k_{Gln}	0.30	0.30	0.26	0.25
$\mu_{d,max}$	0.09	0.10	0.14	0.11
$k_{d,Lac}$	0.01	0.00	0.00	0.00
$k_{d,Amm}$	0.06	0.00	0.00	0.06
$k_{d,Gln}$	0.02	0.50	0.00	0.00
$Y_{Xv/Glc}$	0.11	0.11	0.11	0.11
$Y_{Lac/Glc}$	1.80	1.81	1.81	1.81
m_{Glc}	1.70	1.80	2.82	3.79
$k_{m,Glc}$	19.00	29.67	34.37	50.00
$Y_{Xv/Gln}$	0.38	0.41	0.38	0.38
$Y_{Amm/Glc}$	0.85	0.86	0.85	0.84
β	3.50	4.12	4.04	4.00
α	25.70	25.23	24.54	24.58
k_{μ}	0.02	0.05	0.01	0.01

Table 6-48 Summary of identified parameter values for each method for experiment 3


Derivative estimation for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Latin Hyper Cube for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3

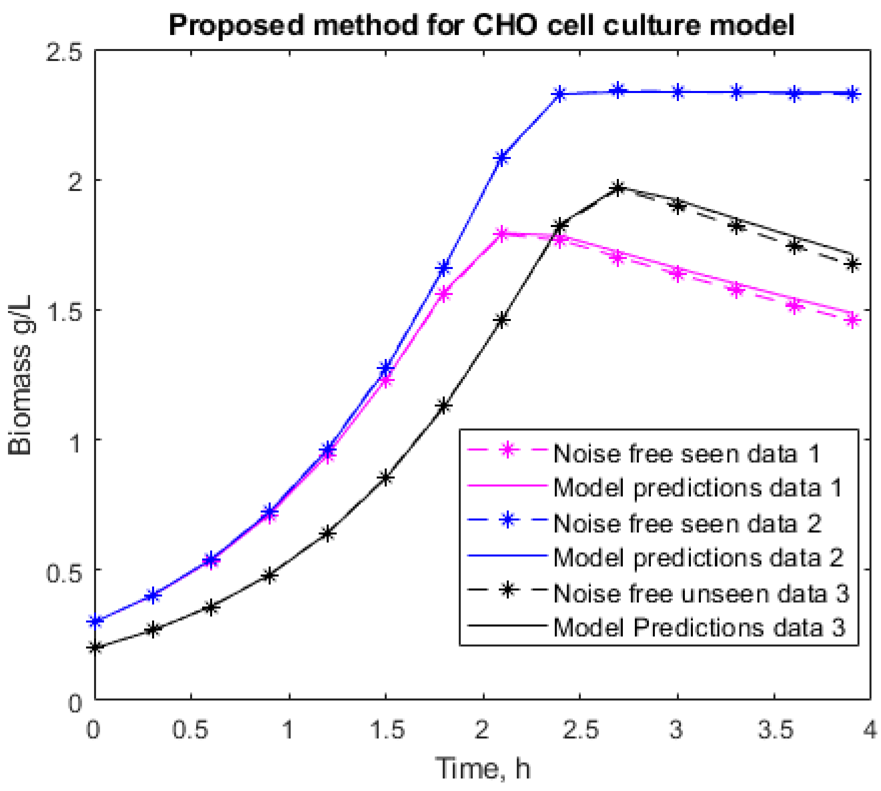


Figure 6.40 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.1h sampling and 10% random noise for measured data.

6.3.3.4. Experiment 4 0.3h sampling and 10% random noise

Computational time (table 6.46) comes back to trend from experiment 1 with the Derivative estimation method being fastest followed by the proposed and Latin hyper cube methods. This type of trend was expected, if the Derivative estimation method is pushed beyond its capability limits. Which mean it should not be able to predict model accurately anymore. This makes the proposed state substitution method slower than the Derivative method by 47% and faster than the Latin hyper cube method by 84%.

Method	**Computational time, s**
Derivative estimation	1466s
Latin hyper cube multi-start	5778s
Proposed state substitution method	2371s

Table 6-49 Computational time of all three method for experiment 4

Just by looking at Sq. Error (table 6.47), we see that the Derivative estimation method indeed falls behind other two methods in terms of accuracy. The difference can be seen clearly in performance graphs (figure 6.38).

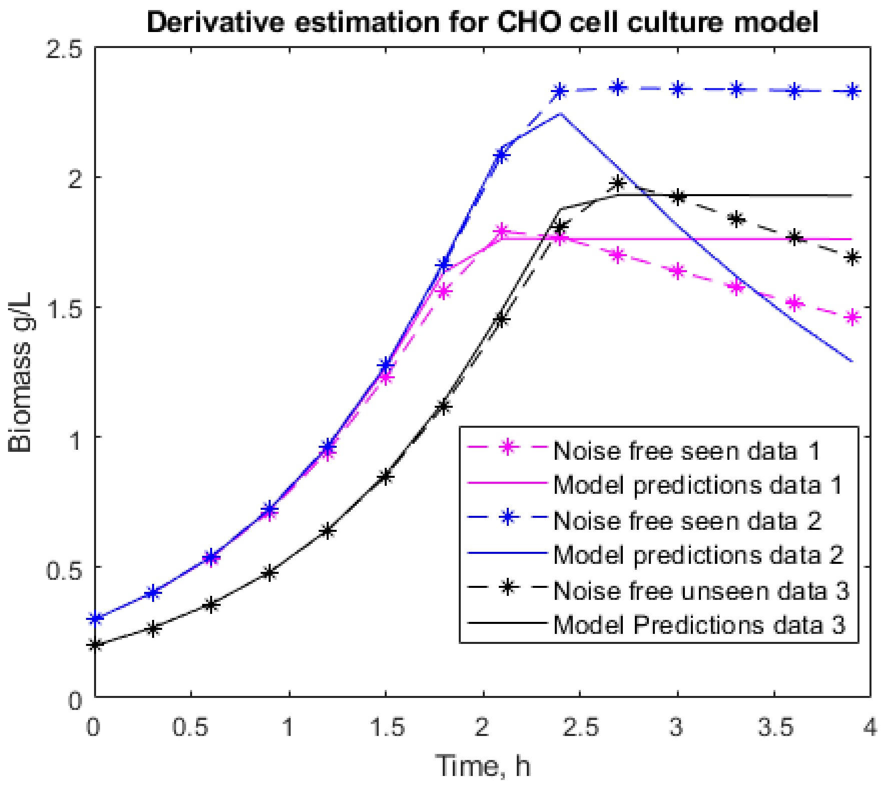
Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	5.22E-01	6.67E-02	2.60E-02
Latin hyper cube multi-start	1.02E-02	3.65E-03	6.84E-03
Proposed state substitution method	4.05E-03	2.35E-03	9.20E-03

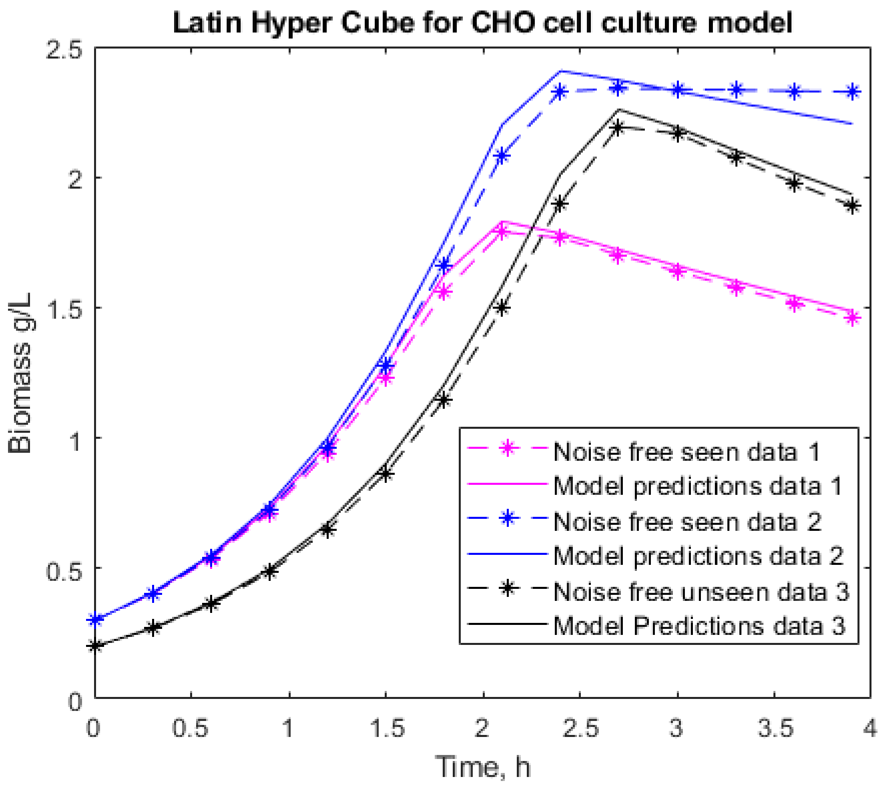
Table 6-50 Squared error values of each method and each data set for experiment 4

As the Derivative estimation method cannot predict trends of the model anymore, parameters identified by it hold no value, but are displayed for comparison. Coupled of the values that seem to be correctly identified allows it to keep accuracy at early parts of the model. When comparing other two methods identified parameter values, parameters o1, o5, o8, o9, o12, o13 and o15 are still consistent between and parameter o10 and o11 vary a lot as before.

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
μ_{max}	1.09	1.15	1.14	1.00
k_{Glc}	1.00	1.07	1.05	0.53
k_{Gln}	0.30	0.00	0.34	0.10
$\mu_{d,max}$	0.09	0.00	0.00	0.01
$k_{d,Lac}$	0.01	0.03	0.03	0.02
$k_{d,Amm}$	0.06	0.05	0.10	0.17
$k_{d,Gln}$	0.02	0.42	0.21	0.07
$Y_{Xv/Glc}$	0.11	0.14	0.12	0.12
$Y_{Lac/Glc}$	1.80	1.85	1.80	1.93
m_{Glc}	1.70	1.32	1.22	0.85
$k_{m,Glc}$	19.00	0.45	10.71	0.13
$Y_{Xv/Gln}$	0.38	0.40	0.40	0.39
$Y_{Amm/Glc}$	0.85	0.76	0.85	0.83
β	3.50	9.89	4.09	1.06
α	25.70	17.03	25.54	25.37
k_{μ}	0.02	0.00	0.05	0.00

Table 6-51 Summary of identified parameter values for each method for experiment 4


Derivative estimation for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Latin Hyper Cube for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3

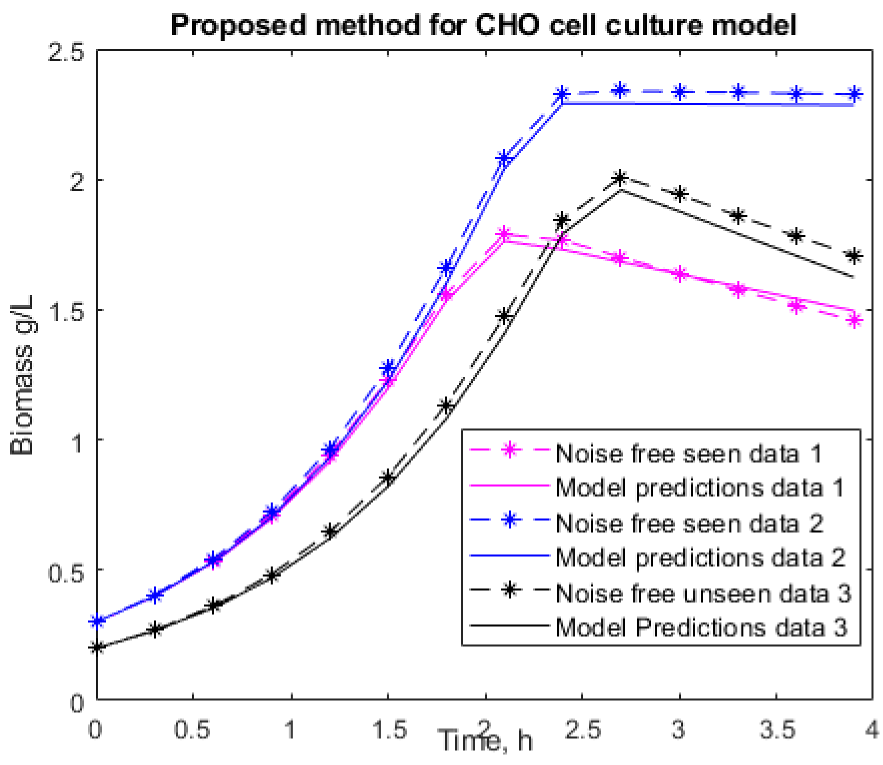


Figure 6.41 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.3h sampling and 10% random noise for measured data.

6.3.3.5. Experiment 5 0.3h sampling and 20% random noise

In worst case scenario experiment 5, the Latin hyper cube and the proposed state substitution method seems to approach their critical points, similar as the Derivative estimation method did in experimental setup 2. The proposed state substitution method is still faster than the Latin hyper cube method by 38%. Decrease in gap of computational time between these two methods shows that both methods approach critical point.

Method	Computational time, s
Derivative estimation	1411s
Latin hyper cube multi-start	11053s
Proposed state substitution method	7492s

Table 6-52 Computational time of all three method for experiment 5

Although the proposed state substitution method can keep the accuracy to certain degree, the Latin hyper cube model struggle to keep accuracy high even more. Both methods suffer in performance due to high noise levels (figure 6.39).

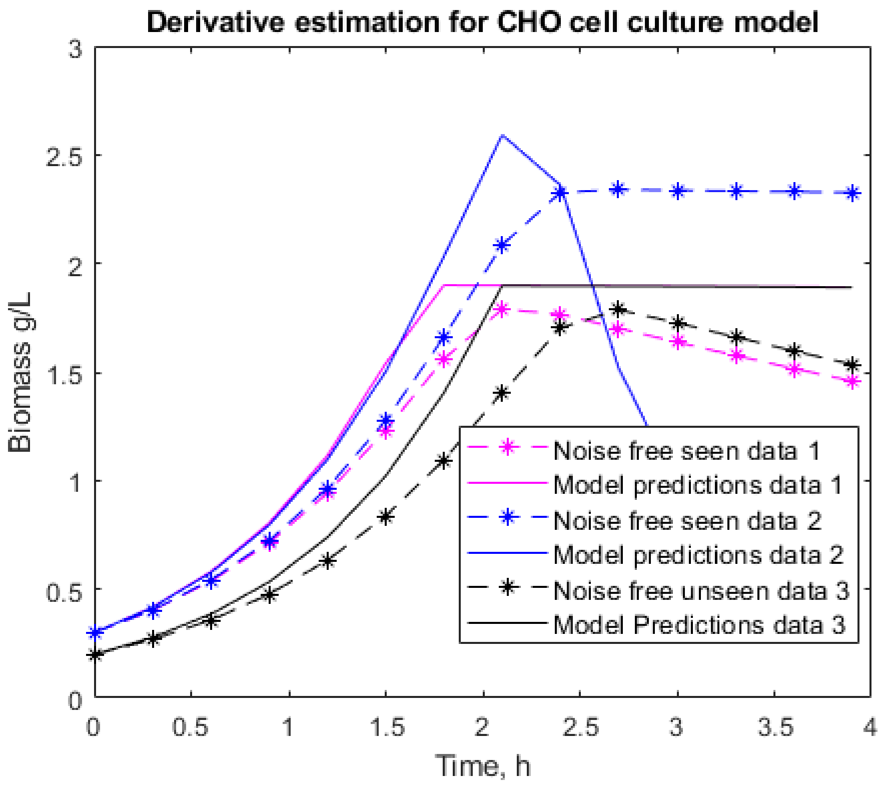
Method	Sq. Error (Blue)	Sq. Error (Pink)	Sq. Error (Black)
Derivative estimation	2.30E+00	2.39E-01	2.15E-01
Latin hyper cube multi-start	9.14E-02	1.82E-01	9.05E-02
Proposed state substitution method	4.46E-02	1.19E-03	4.93E-02

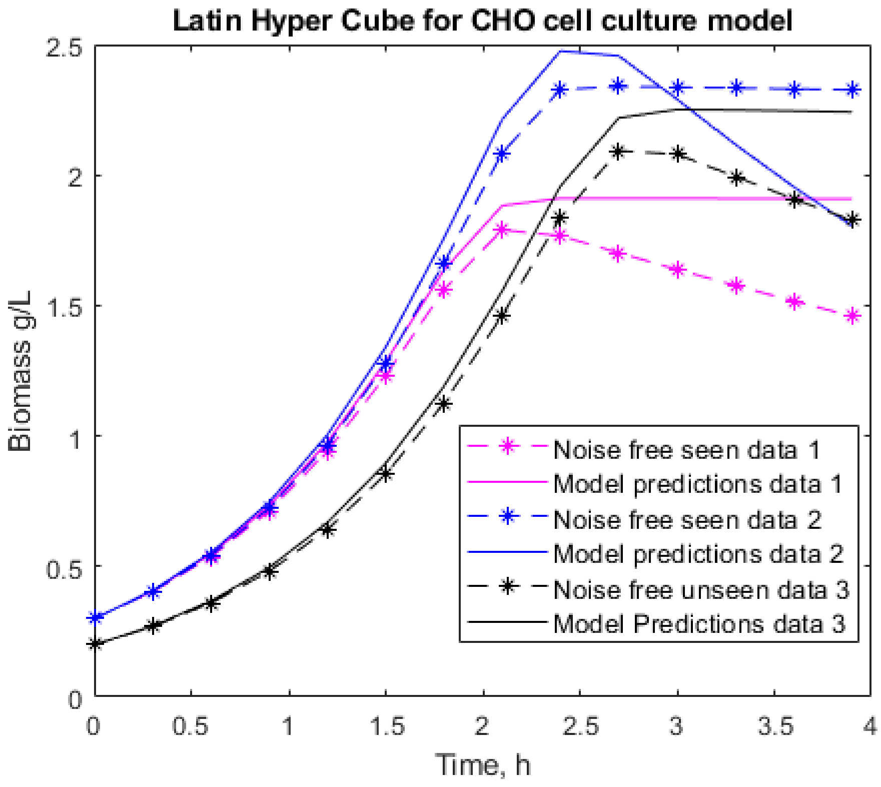
Table 6-53 Squared error values of each method and each data set for experiment 5

Identification of parameters is very difficult at this point and only most dominant variables can be identified. When comparing other two methods identified parameter values, parameters o1, o5, o8, o9, o12, o13 and o15 are still consistent as they were for all five experimental. This suggests that these seven variables are most dominant within this system.

Parameter	Theoretical value	Derivative	Latin hyper cube	Proposed
μ_{max}	1.09	1.08	1.17	1.09
k_{Glc}	1.00	0.79	1.65	1.08
k_{Gln}	0.30	0.27	0.33	0.31
$\mu_{d,max}$	0.09	0.05	0.18	0.09
$k_{d,Lac}$	0.01	0.00	0.00	0.01
$k_{d,Amm}$	0.06	0.15	0.02	0.07
$k_{d,Gln}$	0.02	0.24	0.20	0.02
$Y_{Xv/Glc}$	0.11	0.11	0.11	0.10
$Y_{Lac/Glc}$	1.80	1.80	1.81	1.76
m_{Glc}	1.70	2.25	0.60	0.72
$k_{m,Glc}$	19.00	29.14	0.51	3.23
$Y_{Xv/Gln}$	0.38	0.39	0.40	0.38
$Y_{Amm/Glc}$	0.85	0.84	0.90	0.86
β	3.50	4.30	5.92	3.79
α	25.70	24.81	25.00	25.75
k_{μ}	0.02	0.02	0.11	0.02

Table 6-54 Summary of identified parameter values for each method for experiment 5


Derivative estimation for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3


Latin Hyper Cube for CHO cell culture model
Biomass g/L
Time, h
Noise free seen data 1
Model predictions data 1
Noise free seen data 2
Model predictions data 2
Noise free unseen data 3
Model Predictions data 3

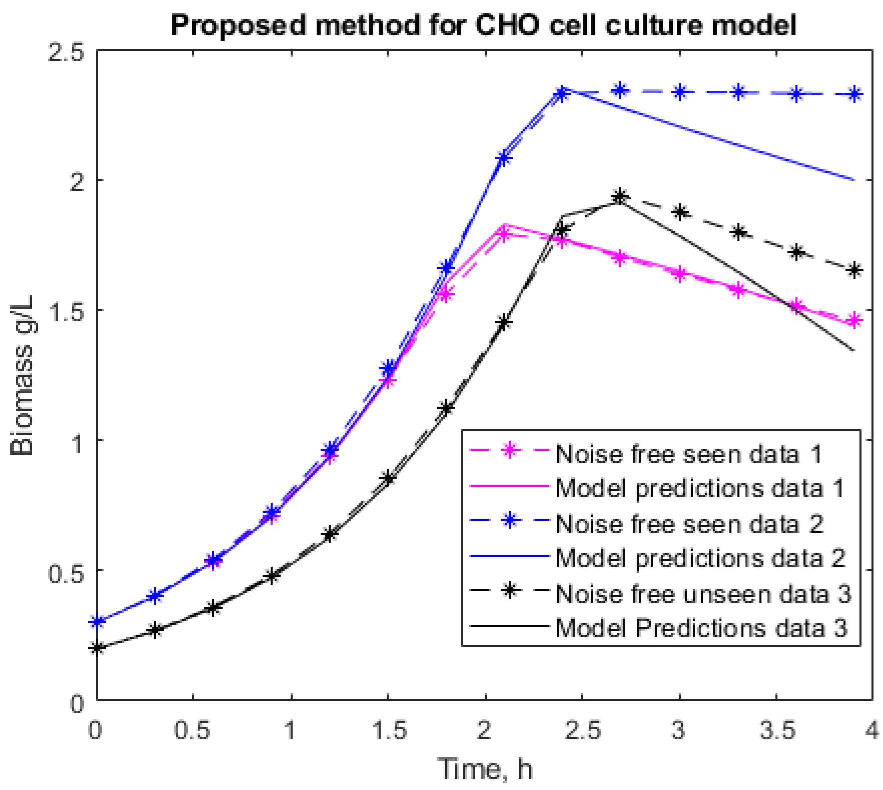


Figure 6.42 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, with 0.3h sampling and 20% random noise for measured data.

6.3.4. The summary of results for the CHO cell culture model

This case study represents the starting complexity of a model the proposed state substitution method is aimed at. It is a system with a moderate number of parameters (sixteen), but complex enough to take significant amount of time to identify parameters using state of art methods. Using complexity analysis tools, complexity of this system was determined to be high for the following reasons. The system describes complex behaviour consisting of four simultaneous reactions, involving two competing substrates and inhibiting by-product. Six ODEs that consist within the system, create a closely coupled network, between first five ODEs. PCA visualization shows multiple local minim that are surrounded by discontinuities. This makes the system especially tricky to optimise with gradient-based algorithms as they can get stuck if they reach boundary of discontinuity. PCA visualization also reveals the in the systems error hyperplane there is a flat boundary level, which separates all the local minima from the rest of the hyperplane. This already indicate high complexity which is also confirmed by the convexity measurement which is 7%. When performing sorted minimisation, it shows that model can achieve good performance and accurate parameter values if algorithm can get pas majority of the local minima. Sorted minimization also reveals that more than half of the samples, are not possible to integrate, due to discontinuity boundary. The SOM analysis are in agreement with PCA visualization showing multiple local minima scattered around the whole plane. The analysis of parameter SOM maps unfortunately is only able to show one dominant variable o1, where it was expected to see at least o1, o2 and o3. Also, the difference between matching factors of dominant variable and the rest are significantly smaller when compared to Monod model SOM analysis. This indicates that increasing number of variables makes this analysis type of analysis less accurate. This most likely is due to fact that large number of parameters make each of them carry less total variance of the system, which in term makes it harder for them to be dominant variable. When comparing results of the different parameter identification algorithms, for this system the derivative estimation method has the lowest computational time, but it also has the lowest accuracy. The derivative estimation method is failing to identify general trend in experiments 4 and 5. The proposed state substitution method is slower than the derivative estimation, but significantly faster than the multi-start method. The proposed state substitution methods accuracy is on par with multi-start accuracy and even slightly better in some cases. I would seem that noise has bigger effect of model performance, than sampling time for all of the methods. Parameter identification reveals that parameter o10 and o11 are extremally not sensitive as their mismatch affect is not seen on the model performance. High complexity of the system allows the proposed state substitution method to utilize its decoupling strategy to keep high accuracy but decrease computational time. All methods have lowest accuracy with unseen data.

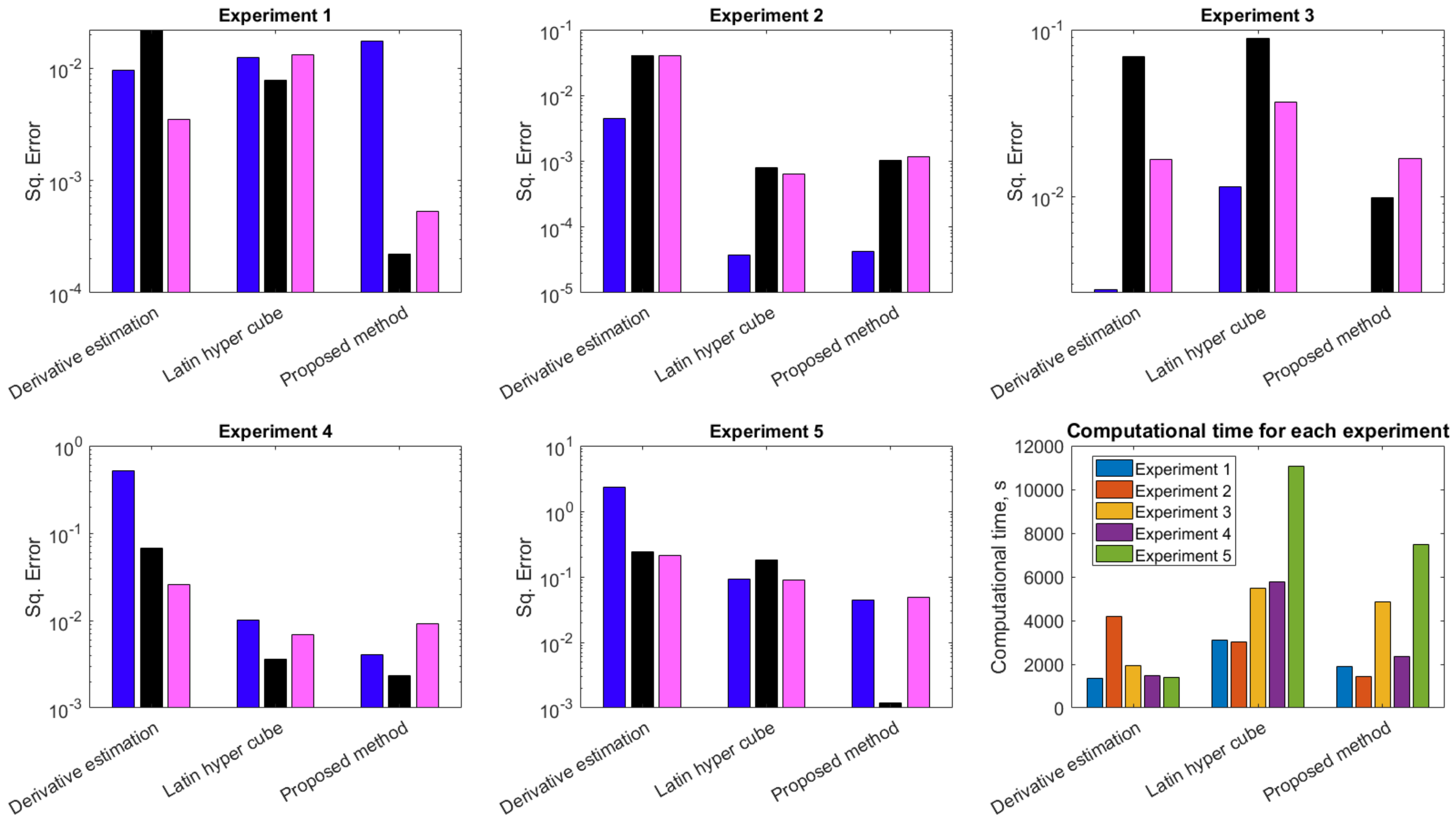


Figure 6.43 Summary of performance results for all three methods for CHO cell culture model

6.4. Case study 4 - Ethanol production with Zymomonas mobilis model

All three cases so far focused on benchmarking developed tools and proposed state substitution method with increasing difficulty and used simulated data which was generated with different noise levels and sampling times to mimic possible scenarios of real data collection. To fully explore benefits and capabilities of this new proposed state substitution method we need to test it against real collected data. This model published by (Hodge and Karim, 2002), and later improved by (Diaz and Willis, 2019). This provides this model a unique environment, where there is plenty of real data based on reactions in this model as it was published long time ago. At the same time, it was shown that model is not perfect and can be improved significantly, meaning there is model – process mismatch making parameter identification more difficult and more realistic. Ten different measured data sets were collected for this case study. All three method will be compared within all ten data sets. Sampling times are determined by data collected and noise levels are unknown. Model has two competing substrates glucose and xylose which produce ethanol and allow biomass to grow. Although it has less ODEs the case study 3, it has increased number of parameters to twenty-six. Equation 6.4 shows full mathematical model.

$$\frac{dX}{dt} = r1 + r2 \quad \frac{dS1}{dt} = -r3 \quad \frac{dS2}{dt} = -r4 \quad \frac{dP}{dt} = r3 * YP\_S1 + r4 * YP\_S2$$

$$r1 = X * \left(\frac{u\_max1 * S1}{K1x + S1 + S2 * \frac{K1x}{K2x}} \right) * \mathrm{f5(S2 + S1) * f6(P)}$$

$$r2 = X * \left(\frac{u\_max2 * S2}{K2x + S2 + S1 * \frac{K2x}{K1x}} \right) * \mathrm{f7(S2 + S1) * f8(P)}$$

$$r3 = X * \left(\frac{q\_pmax1 * S1}{K1 + S1 + S2 * \frac{K1}{K2}} \right) * \mathrm{f1(S2 + S1) * f2(P)} \qquad 6.4$$

$$r4 = X * \left(\frac{q\_pmax2 * S2}{K2 + S2 + S1 * \frac{K2}{K1}} \right) * \mathrm{f3(S2 + S1) * f4(P)}$$

$$f_i(j) = \begin{Bmatrix} a_i * j^2 + b_i * j + 1, \ f_i(j) \geq 0 \\ 0, \ otherwise \end{Bmatrix} \ i = 1,2 \ldots 8; j \in \{P, S1 + S2\}$$

6.4.1. Complexity analysis

Complexity analysis will be performed in three steps as mentioned in cshapter 6.1.1. These steps consist of performing PCA visualization with convexity calculations, sorted minimization and SOM analysis. Aim of this analysis is to compare its' relative complexity with other cases studies within this work, and to find problems gradient-based algorithms might run into trying to optimize this system.

6.4.1.1. PCA visualization

PCA visualization of this system model, shows very similar results to case study 3, but there are some key differences too. It has similar shape - non-convex points scattered everywhere with couple of convex points mixed in, with no obvious global optimum valley. This leads to believe there is multiple local optimums within the system. In contract to case study 3, Zymomonas mobilis model, does not have overwhelming number of failed integrations, which mean error plane is smooth, also it lacks flat plane barrier. Without these two obstacles, optimization algorithm should have easier time reach optimum solutions. That being said, overall convexity of the system is 2%. This complexity is also reflected, by decreases variability explained by PCA visualization. PC1 explains 48.9% of variability and PC2 explain 31.0% variability for a total of 79.9%.

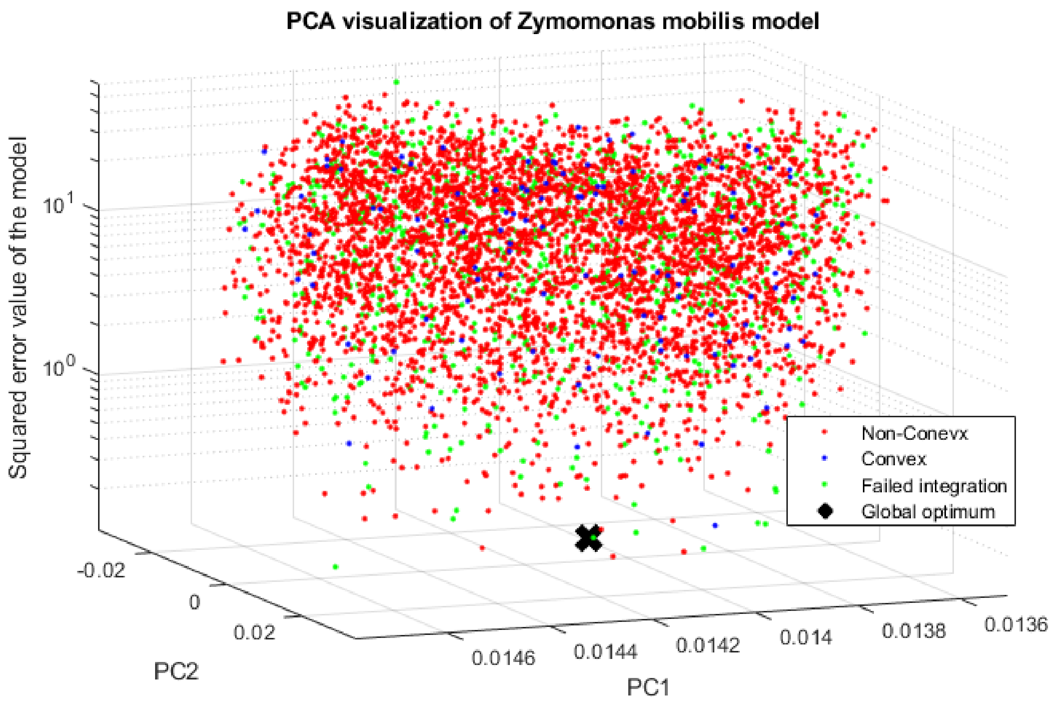


Figure 6.44 PCA error plot for Zymomonas mobilis model (6.4), with colour coded points for convexity, where blue points are convex, red points are non-convex and green points are failed integrations. Black x marks global optimum solution.

6.4.1.2. *Sorted minimization*

Sorted optimization is performed same way as described in section 4.7. Parameter space is sampled for thousand initial parameter values. suspected from PCA visualization, sorted minimization method reveal multiple local minimums, eventually most of them settle for one global solution. This global solution although has lowest error value, it is still high error value. This implies actual parameter values after identification might not be, real parameter value, just optimal for performance. This is mostly cause due to fact we are using real collected data, and there is model – process mismatch. This makes it hard to drive overall error value very low. Nevertheless, this sorted minimization of Zymomonas mobilis model, suggest that it should not be too difficult to achieve optimal performance with the quality of data provided, but it might prove problematic to determinate actual parameter values.

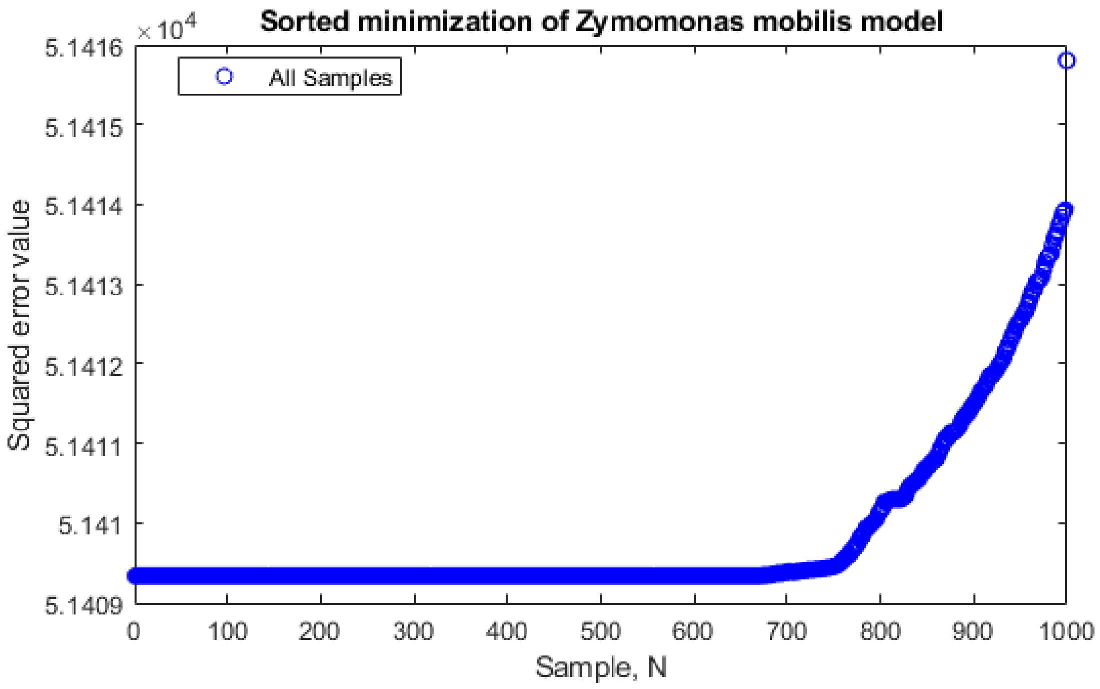


Figure 6.45 Zymomonas mobilis model (6.4) thousand samples sorted optimization, where blue circles are all samples.

6.4.1.3. *SOM analysis*

SOM analysis seem to provide similar picture as PCA visualization, that there are large number of local optimums scattered across the plane. It might not be clear of with colour coded hexes only, but U-matrix makes it clear that there is a lot of local minimums (figure 6.42). If we would compare to SOM analysis of case study 3, there is one major difference. In SOM analysis of the CHO cell culture model (6.3), local minimums where uniformly spread, whereas in SOM map of the Zymomonas mobilis model, local minimums are clustered. We can see four clusters of low error regions separated with high error region in the middle of U-matrix. These clusters represent valleys of local minimum, which we could not observe in PCA visualization. This shows that with large number of variables it is much harder for PCA visualization to capture 2d representation of higher dimensional plane, whereas SOM map, can still do a good job.

To observe if any of the parameters are dominant, we perform cross-correlation of errors map with component maps. Figures 6.44-45 shows how well component maps match up versus total error maps. Positive values mean higher degree of matching, and lower values mean lower. Cross correlation factor of 1, represents perfect match and 0 represent prefect mismatch. Although large number of parameters should hinder SOM parameter analysis to determinate dominant variables, it is able to pick two dominant variables out of twenty-six, u_max1 most dominant variable negatively with matching factor of 0.73 and K1x most dominant variable positively with matching factor of 0.75. Maximum difference between matching factors is 0.18 and 0.20 respectively to positive and negative matching factors.

Parameters	Positive Matching Factor	Negative Matching Factor
u_max1	0.57 ± 0.02	0.73 ± 0.02
u_max2	0.64 ± 0.03	0.61 ± 0.04
K1x	0.75 ± 0.01	0.54 ± 0.03
K2x	0.61 ± 0.02	0.66 ± 0.03
q_pmax1	0.64 ± 0.05	0.64 ± 0.04
q_pmax2	0.67 ± 0.03	0.56 ± 0.03
K1	0.65 ± 0.02	0.64 ± 0.02
K2	0.62 ± 0.02	0.65 ± 0.01
YP_S1	0.66 ± 0.03	0.63 ± 0.04
YP_S2	0.63 ± 0.04	0.65 ± 0.02
a(1)	0.65 ± 0.04	0.60 ± 0.06
a(2)	0.64 ± 0.02	0.64 ± 0.03
a(3)	0.63 ± 0.02	0.64 ± 0.04
a(4)	0.68 ± 0.03	0.63 ± 0.03
a(5)	0.64 ± 0.06	0.62 ± 0.07
a(6)	0.62 ± 0.02	0.65 ± 0.01
a(7)	0.64 ± 0.02	0.62 ± 0.01
a(8)	0.63 ± 0.03	0.61 ± 0.06
b(1)	0.65 ± 0.02	0.65 ± 0.03
b(2)	0.64 ± 0.02	0.62 ± 0.06
b(3)	0.64 ± 0.03	0.63 ± 0.02
b(4)	0.63 ± 0.06	0.60 ± 0.05
b(5)	0.61 ± 0.03	0.67 ± 0.03
b(6)	0.62 ± 0.02	0.65 ± 0.04
b(7)	0.66 ± 0.03	0.62 ± 0.03
b(8)	0.65 ± 0.04	0.65 ± 0.03

Table 6-55 Table of positive and negative mean matching factor of each parameter for system (6.4)

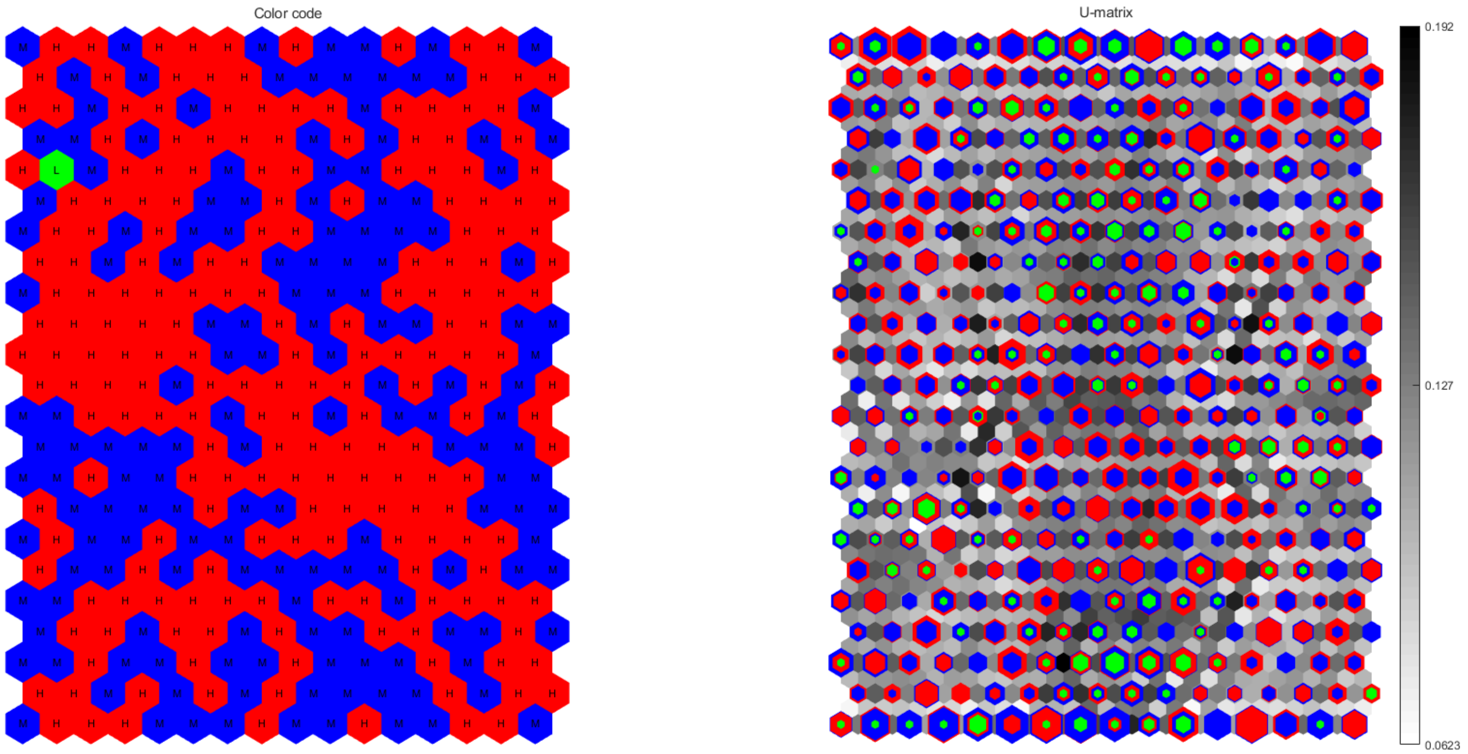


Figure 6.46 Left 2d SOM of Zymomonas mobilis model (6.4), colour coded based of model error, where green (L) is low error, blue (M) is medium error, and red (H) is high error. Right 2d SOM with same colour code as left, but also showing relative distance in n-dimensional plane represented as colour bar and number of individual members of each cluster.

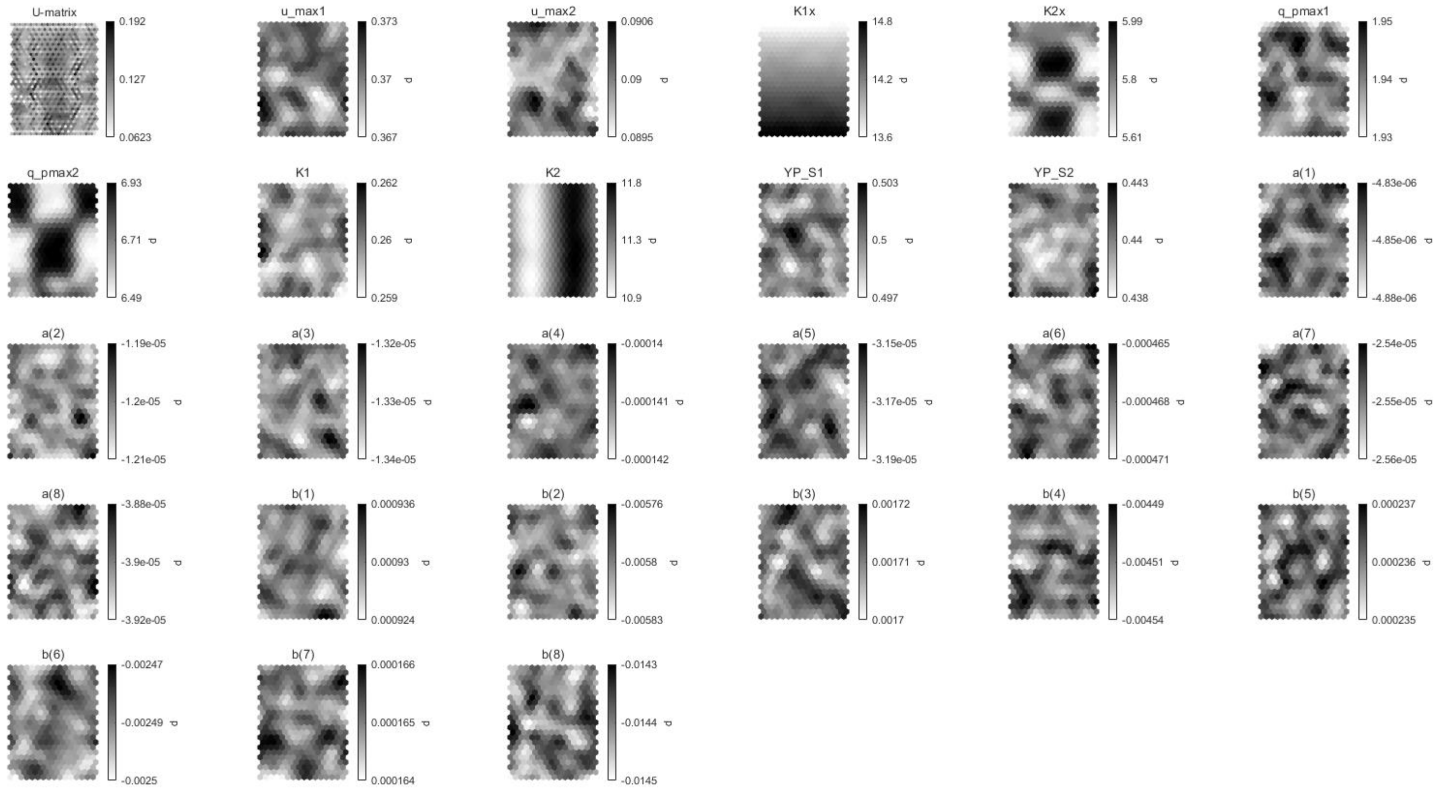
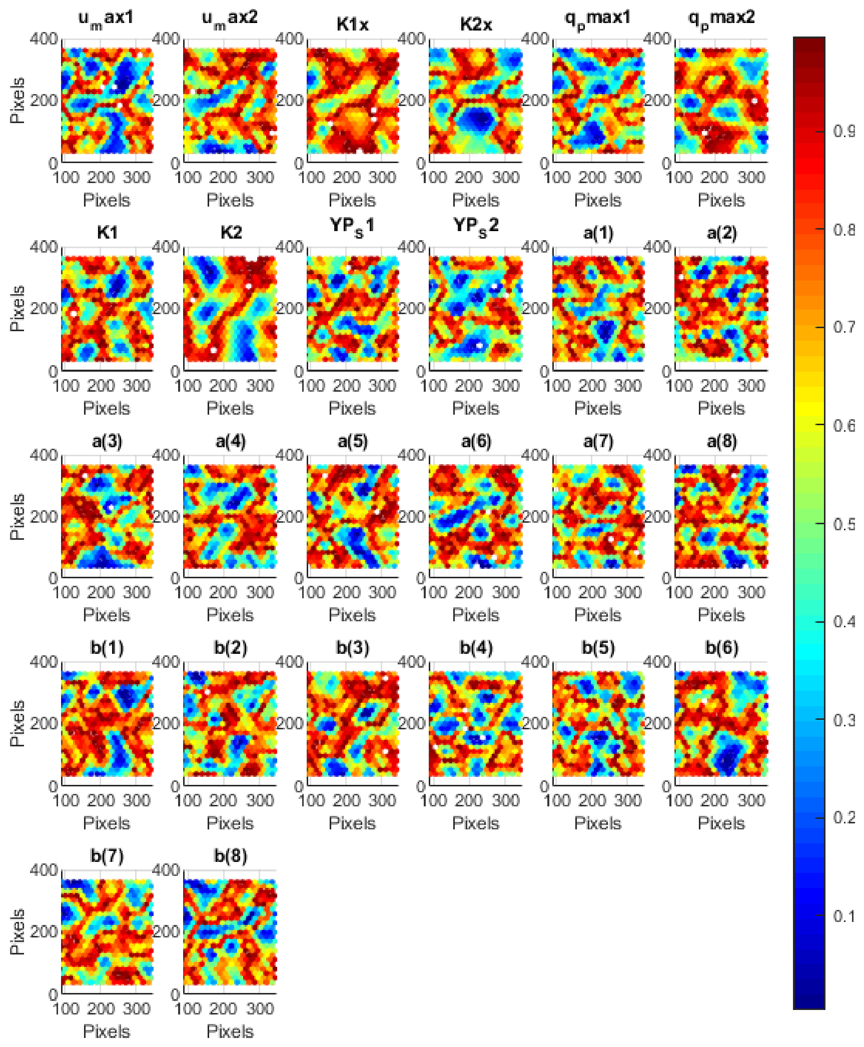


Figure 6.47 U-matrix represents N-dimensional plane as 2d SOM with relative distances as colour bar, followed by each component n-dimensional plane of its value distribution represented as colour bar.

a)



b)

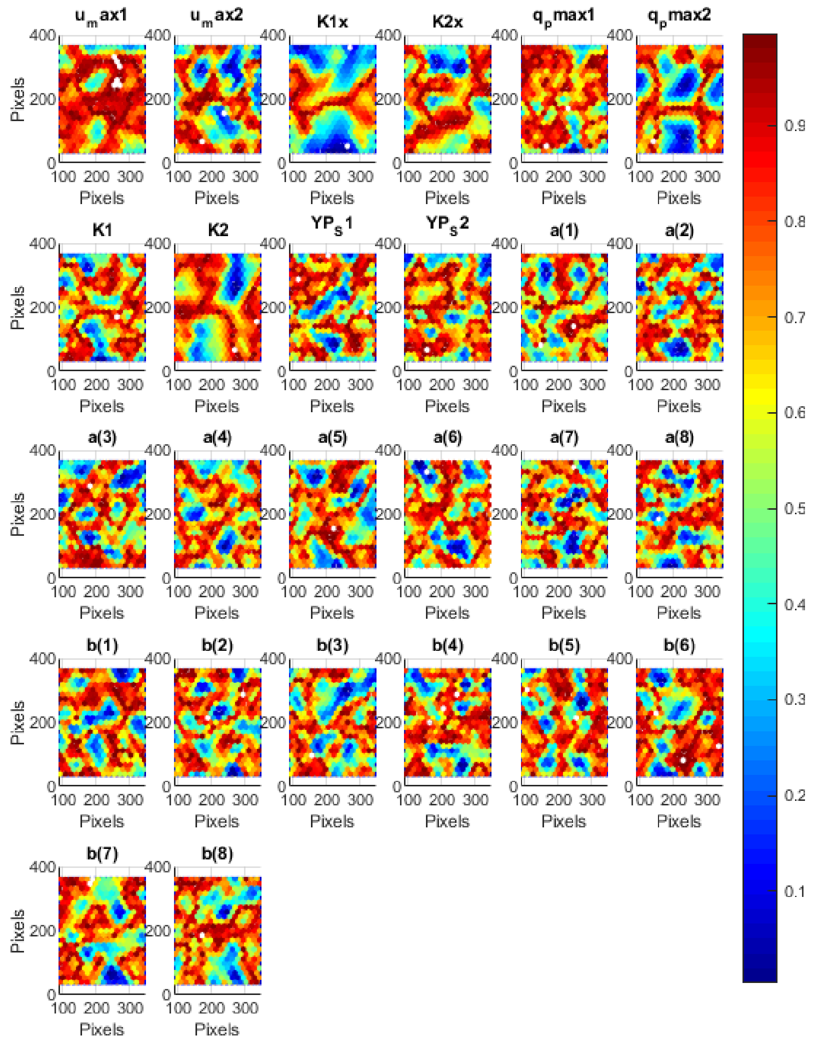


Figure 6.48 a) Positive cross-correlation between each parameter map and overall error map. b) Negative cross-correlation between each parameter map and overall error map.

6.4.2. *Model hierarchy*

The Zymomonas mobilis model consist of four coupled ODE's, which when decoupled with proposed state substitution method leave four independent sub-sets that can be solved in any order. Specific order of solution will yield best results, therefore model hierarchy needs to be established, based on importance on each sub-set and how many dominant variables it has. To figure out best hierarchy of the model we need to use additional tools like bi-partite chart (figure 6.47) and SOM analysis of parameter importance (figures 6.44-6.45, table 6.52). When combined these tools reveal which order of solving individual subsets will lead best results. Bi-partite chart reveals that although all four states of model have heavy coupling, parameters can be separated into groups that do not mix. Only exception to that is ethanol, as it has same parameters as both substrates, as it directly dependant on them. This makes model hierarchy simple two-level hierarchy with ethanol state being only state in second level. Unfortunately, there are two ratios in this model, but because they only exist in ratio form and never separate, it is not possible to solve for their exact values.

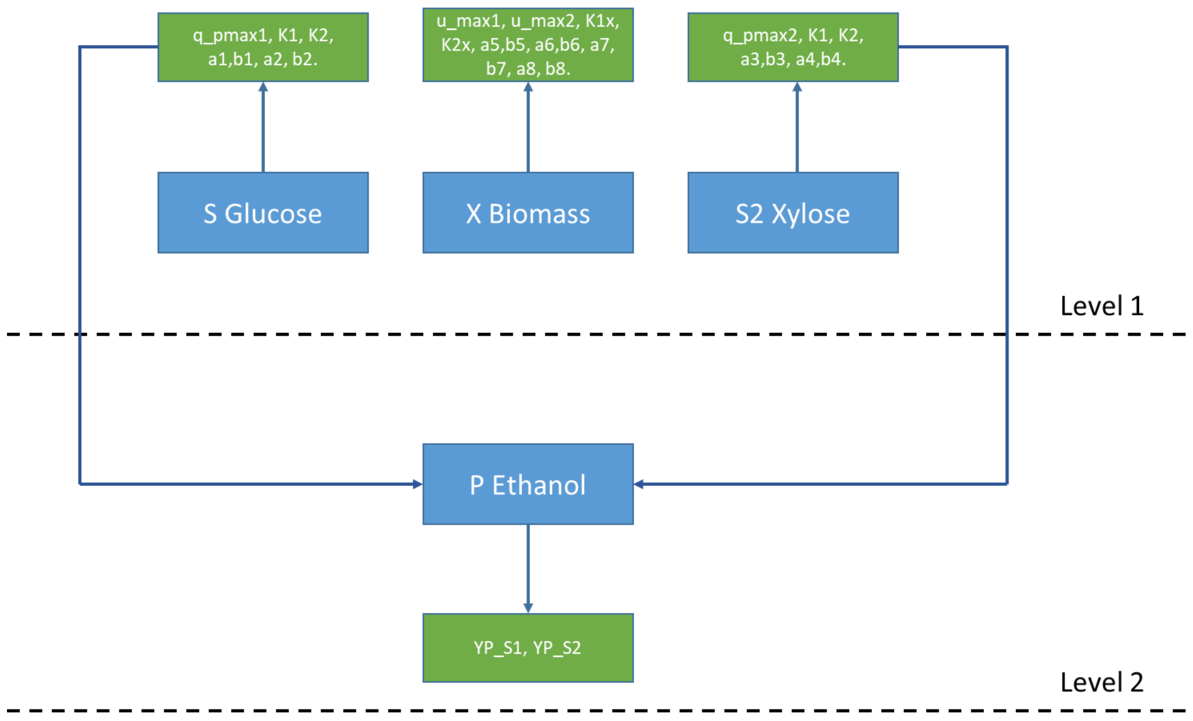

q_pmax1, K1, K2, a1,b1, a2, b2.
u_max1, u_max2, K1x, K2x, a5,b5, a6,b6, a7, b7, a8, b8.
q_pmax2, K1, K2, a3,b3, a4,b4.
S Glucose
X Biomass
S2 Xylose
Level 1
P Ethanol
YP_S1, YP_S2
Level 2

Figure 6.49 Hierarchy of the Zymomonas mobilis model (6.4)

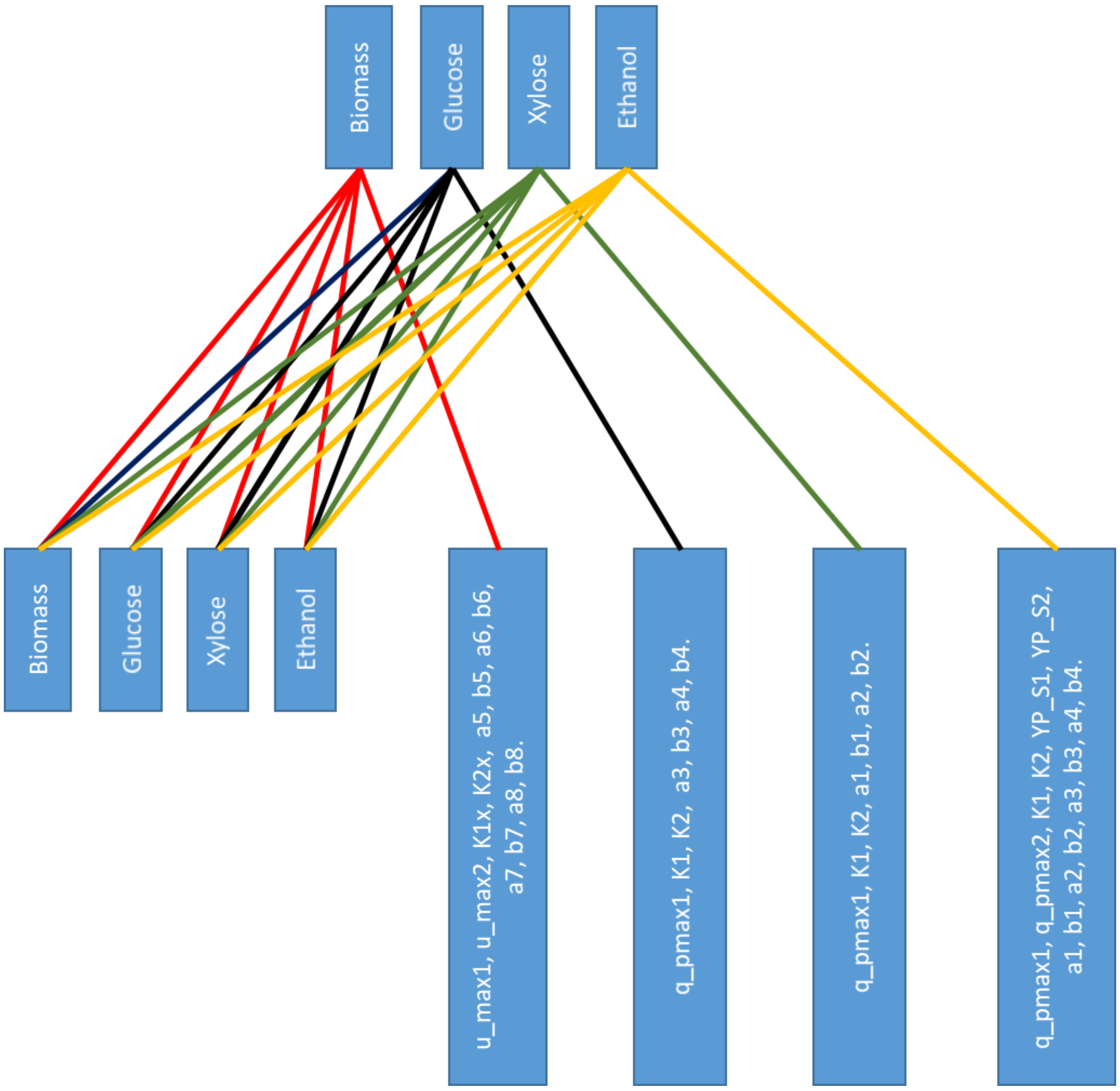

Biomass
Glucose
Xylose
Ethanol
Biomass
Glucose
Xylose
Ethanol
u_max1, u_max2, K1x, K2x, a5, b5, a6, b6, a7, b7, a8, b8.
q_pmax1, K1, K2, a3, b3, a4, b4.
q_pmax1, K1, K2, a1, b1, a2, b2.
q_pmax1, q_pmax2, K1, K2, YP_S1, YP_S2, a1, b1, a2, b2, a3, b3, a4, b4.

Figure 6.50 Bipartite chart of the Zymomonas mobilis model (6.4)

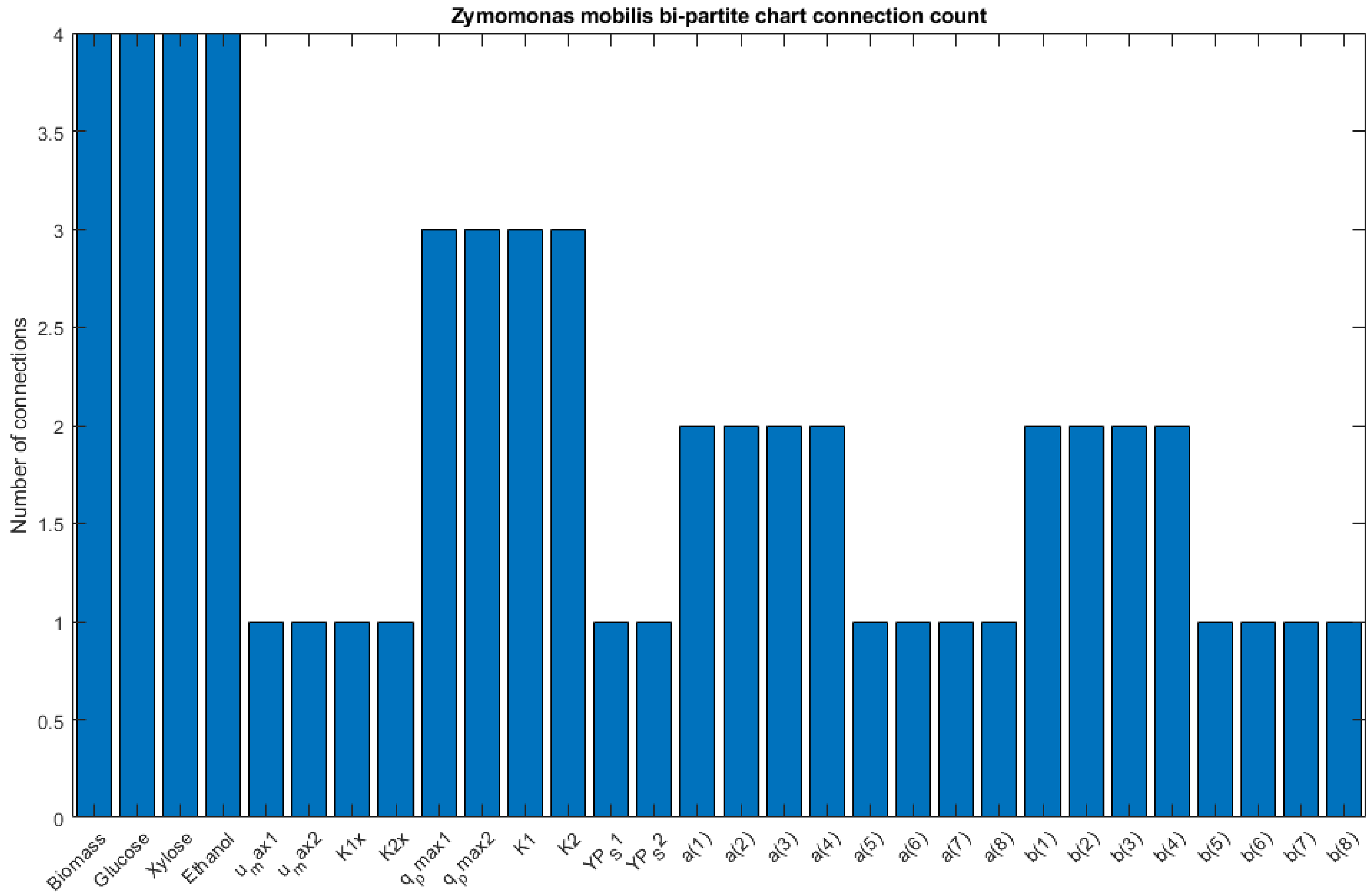


Figure 6.51 Bar graph of connections of Zymomonas mobilis bipartite chart

6.4.3. Method comparison

To evaluate the proposed state substitution method, it will be compared to the two existing methods discussed in chapter 3. The derivative estimation and Latin-hyper cube sampled multi start method. Two main criteria will be compared, model performance accuracy and the computational time between each of the methods. Although the proposed state substitution method aims to reduce computational time, model performance accuracy is also very important and cannot be completely neglected. In contrast to the first three case studies, we no longer have simulated data, so we cannot vary noise or sampling time to see its effects on methods. Instead after parameter identification each of their model prediction will be compared with measured data, for each of the ten different data sets. Although measured data can be unreliable and might have larger noise or error involved methods performance will be compared as in how close its prediction is to measured data. As before only biomass data will be presented for model performance as biomass dictates accuracy for the rest of the states. This will allow to avoid unnecessary graphs while still presenting enough evidence about model performance.

When comparing computational time (table 6.53) we see that the proposed state substitution method is fastest among all three methods. It is faster by 56% compared to the Latin hyper cube method, and 44% faster than the Derivative estimation method.

Method	**Computational time, s**
Derivative estimation	7019s
Latin hyper cube multi-start	7948s
Proposed state substitution method	4478

Table 6-56 Computational time of all three methods

As discussed before to compare performance of model prediction of each of the methods, squared error value was calculated for each method versus experimental data. The Derivative estimation method and Latin hyper cube method, has same error values because their identified parameters were identical, meaning they both reached same local minimum. This is most likely caused, because SOM analysis revealed clusters of local minimums, and one of the clusters was larger than the other. This mean that starting locations of the Latin hyper cube method, are much more likely to start in this large cluster and if this cluster does not hold global optimum, it becomes nearly impossible to reach. On other hand the Derivative estimation method uses global search global solver for optimization. Starting position for this global optimizer is determined by optimizing derivative estimates at each time point and taking average of all solutions. Although global search after reaching solution look for other solutions, it starts its search around primary solution. If this cluster is local minimum is large enough, global search will never look outside of its boundaries. Both methods in comparison to the proposed state substitution method did worse, managing to get three out of ten data sets to lower squared error value than the proposed state substitution method (table 6.54). Furthermore, the proposed state substitution method in data sets 7 and 9, managed to capture correct trend of experimental data, where in comparison other two methods failed to even predict general trend.

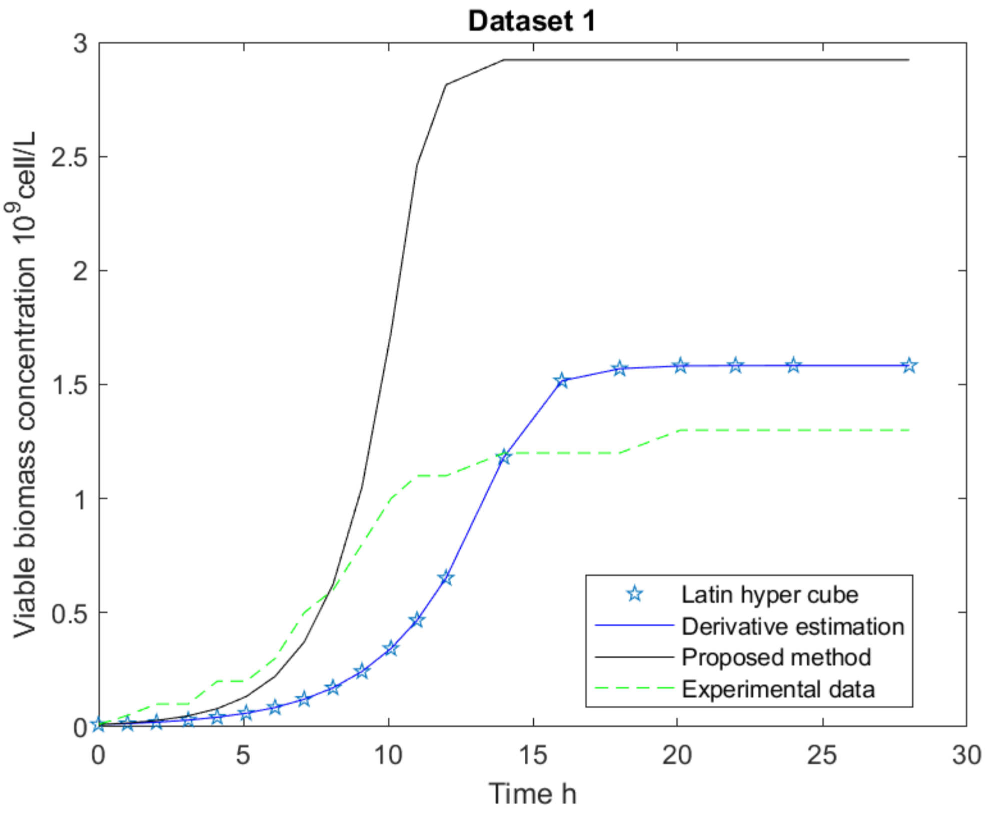
Comparing identified parameter values (table 6.55), shows large gap within the proposed state substitution method and other two methods. This makes it difficult to say with confidence that identified parameter values are correct parameter values, as sorted minimisation suggests, even once global optimum is reached total error value is still high.

Data set number	Derivative estimation	Latin hyper cube	Proposed state substitution method
1	2.33E+00	2.33E+00	2.49E+01
2	2.22E+00	2.22E+00	7.75E-01
3	1.05E+00	1.05E+00	7.33E-01
4	5.43E-01	5.43E-01	1.06E+00
5	8.25E+00	8.25E+00	1.92E+00
6	3.35E+00	3.35E+00	2.72E-01
7	5.01E+00	5.01E+00	1.68E+00
8	1.37E+00	1.37E+00	1.76E+01
9	9.66E+00	9.66E+00	9.75E-01
10	3.39E+00	3.39E+00	4.53E-01

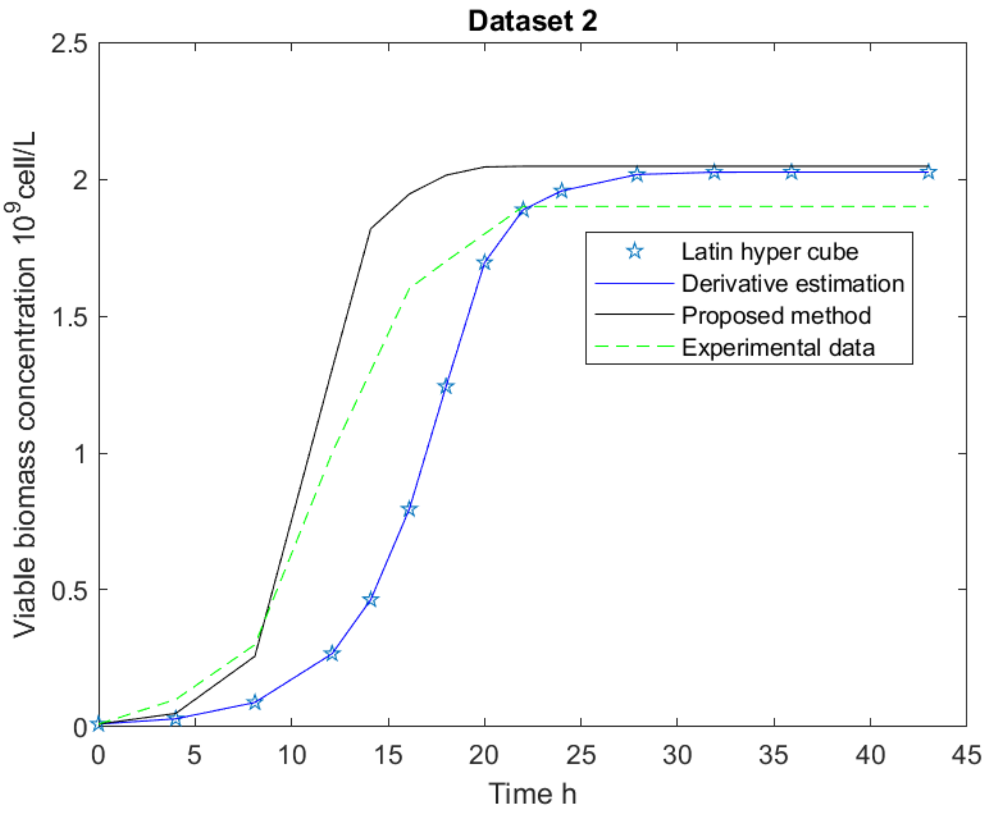
Table 6-57 Squared error values for each method and each data set

Parameters	Derivative estimation	Latin hyper cube	Proposed state substitution method
u_max1	0.389	0.389	0.047
u_max2	0.095	0.095	0.45
K1x	13.772	13.772	0.409
K2x	6.08	6.08	0.121
q_pmax1	2.037	2.037	2.02
q_pmax2	6.375	6.375	12.53
K1	0.247	0.247	0.294
K2	11.886	11.886	1.697
YP_S1	0.502	0.502	0.806
YP_S2	0.462	0.462	0.349
a(1)	0	0	0
a(2)	0	0	-0.001
a(3)	0	0	0
a(4)	0	0	-0.002
a(5)	0	0	0
a(6)	0	0	0
a(7)	0	0	0
a(8)	0	0	-0.005
b(1)	0.001	0.001	-0.004
b(2)	-0.006	-0.006	0.025
b(3)	0.002	0.002	-0.007
b(4)	-0.004	-0.004	0.1
b(5)	0	0	0.023
b(6)	-0.002	-0.002	0.007
b(7)	0	0	0.013
b(8)	-0.014	-0.014	0.05

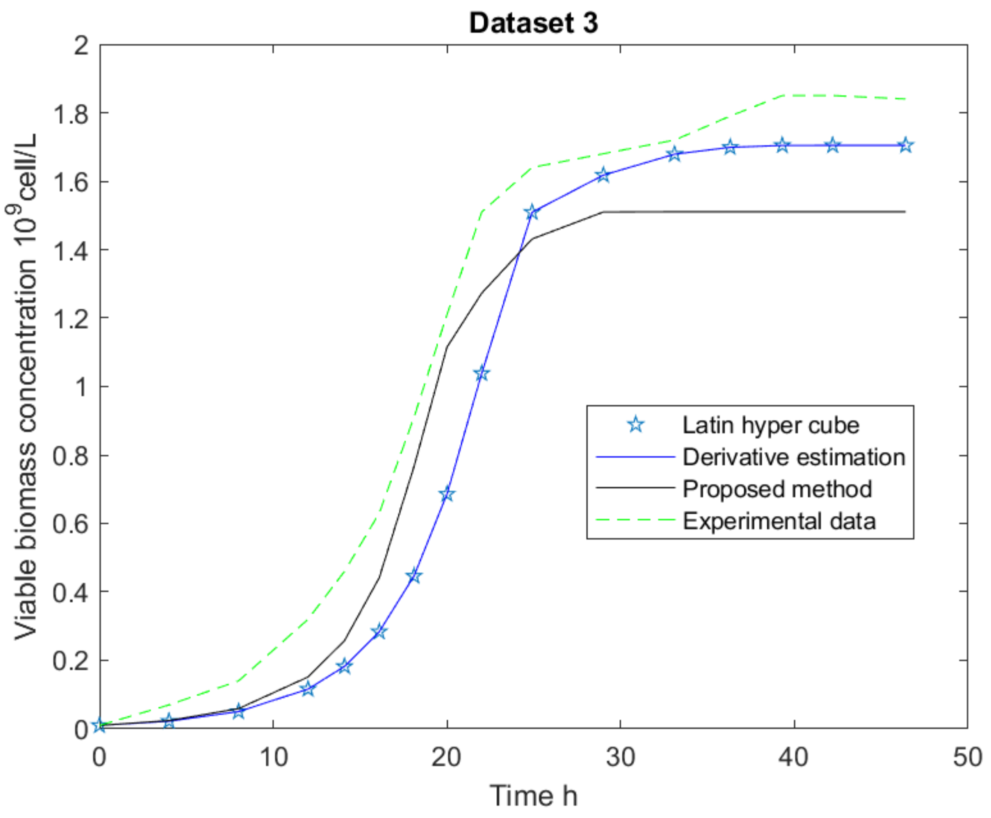
Table 6-58 Summary of identified parameter values for each method

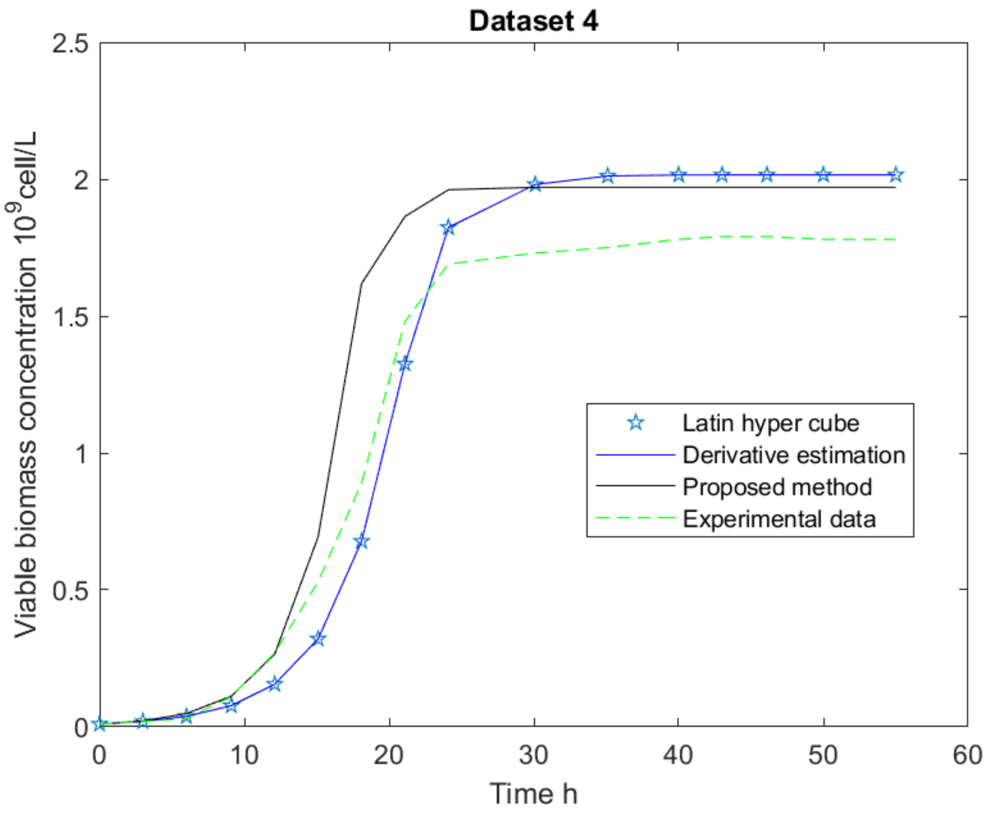


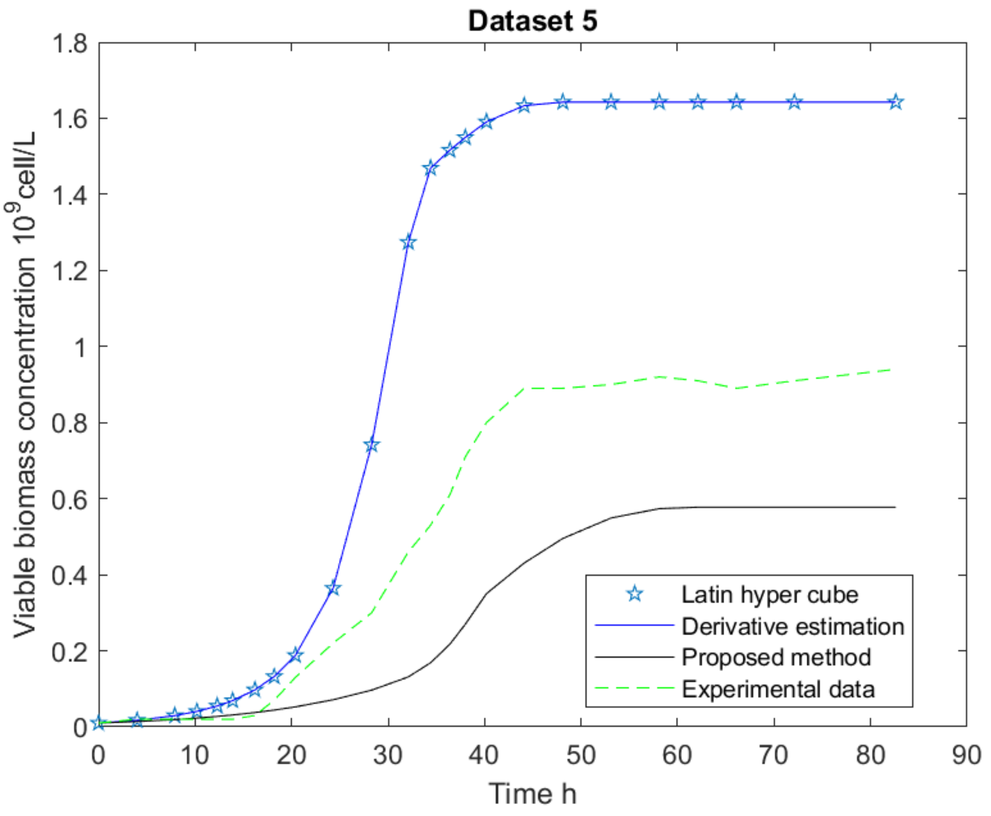
Dataset 1
Viable biomass concentration 10^9cell/L
Time h
Latin hyper cube
Derivative estimation
Proposed method
Experimental data



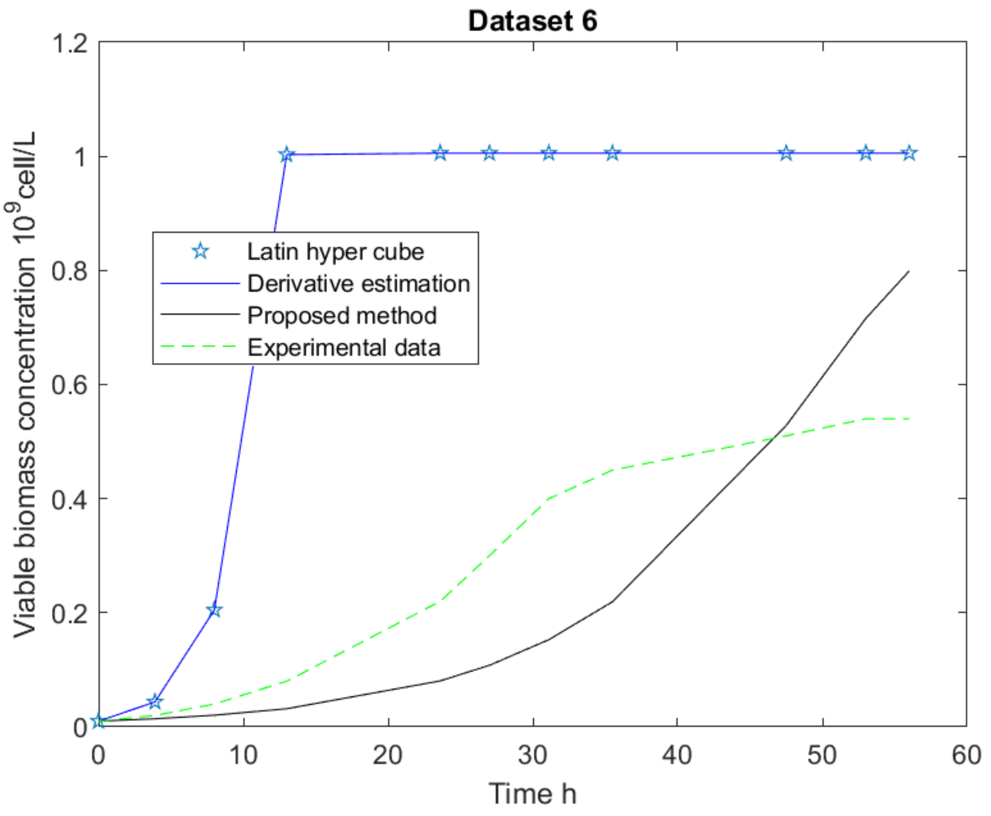
Dataset 2
Viable biomass concentration 10^9cell/L
Time h
Latin hyper cube
Derivative estimation
Proposed method
Experimental data


Dataset 3
Viable biomass concentration 10^9cell/L
Time h
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
0
10
20
30
40
50
Latin hyper cube
Derivative estimation
Proposed method
Experimental data

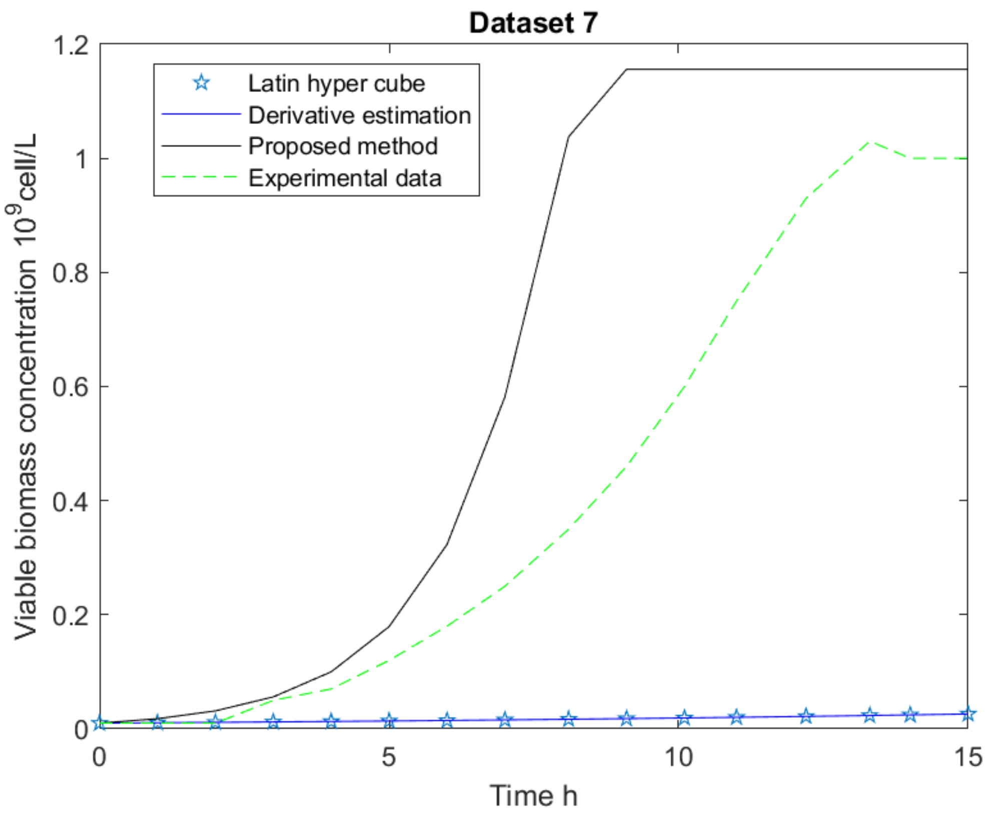

Dataset 4
Viable biomass concentration 10^9cell/L
Time h
0
0.5
1
1.5
2
2.5
0
10
20
30
40
50
60
Latin hyper cube
Derivative estimation
Proposed method
Experimental data

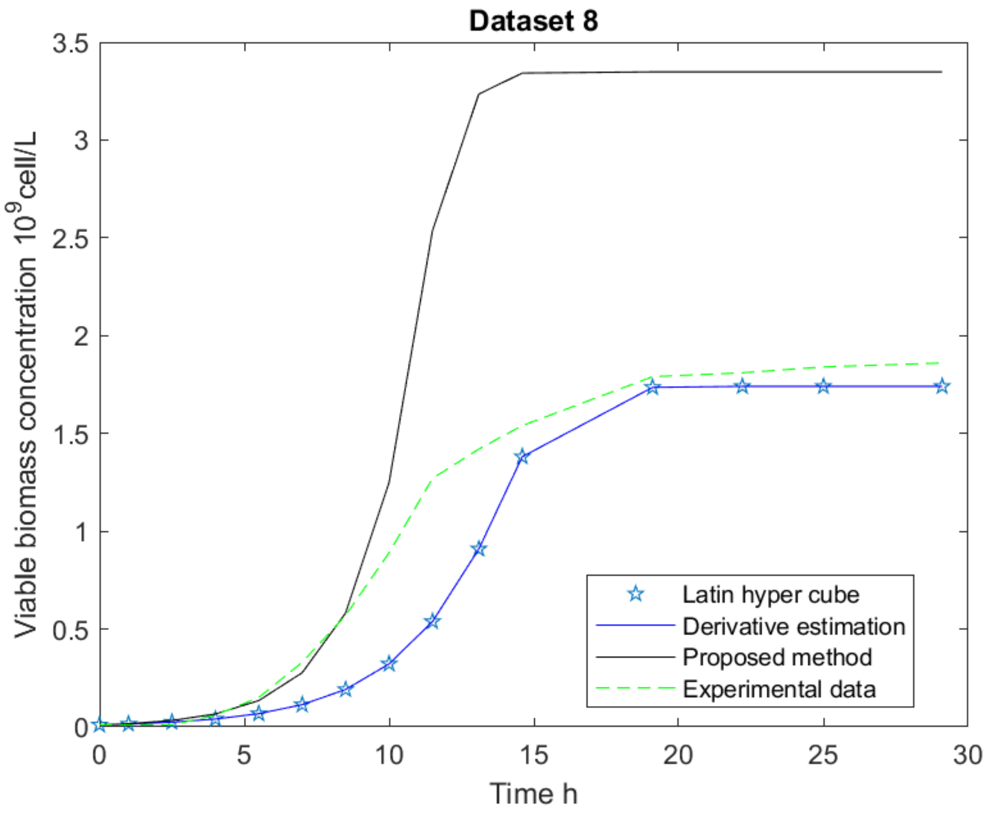


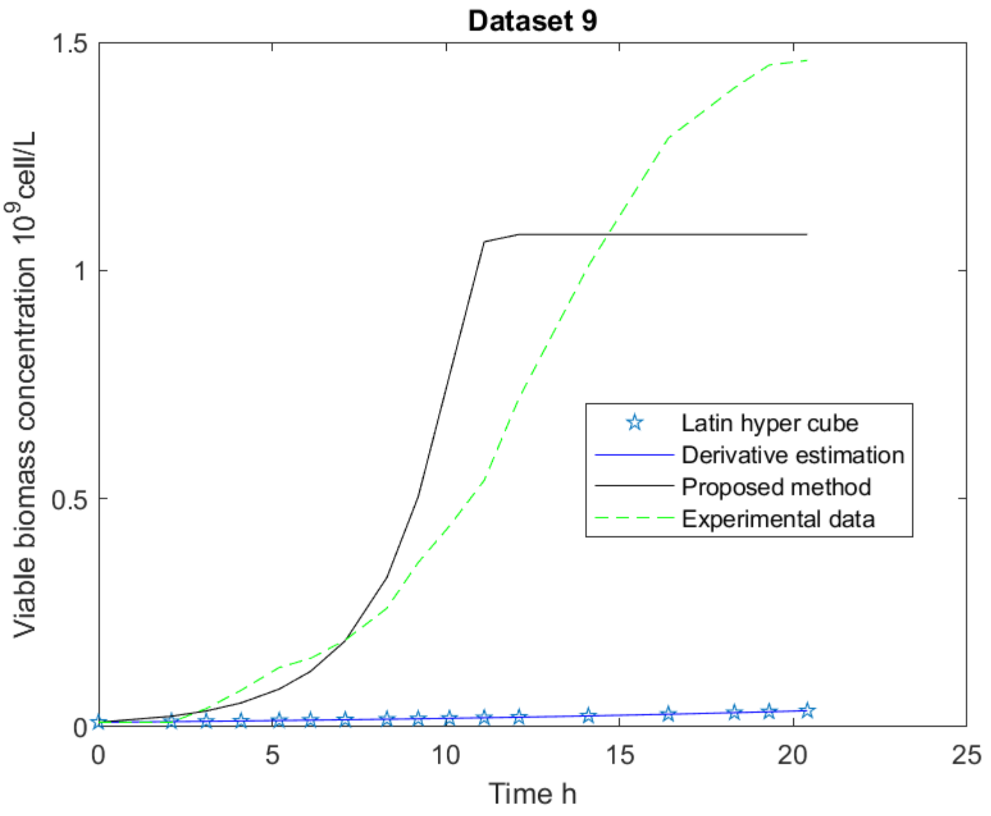
Dataset 5
Viable biomass concentration 10^9cell/L
Time h
Latin hyper cube
Derivative estimation
Proposed method
Experimental data



Dataset 6
Viable biomass concentration 10^9cell/L
Time h
Latin hyper cube
Derivative estimation
Proposed method
Experimental data


Dataset 7
Latin hyper cube
Derivative estimation
Proposed method
Experimental data
Viable biomass concentration 10^9cell/L
Time h


Dataset 8
Latin hyper cube
Derivative estimation
Proposed method
Experimental data
Viable biomass concentration 10^9cell/L
Time h



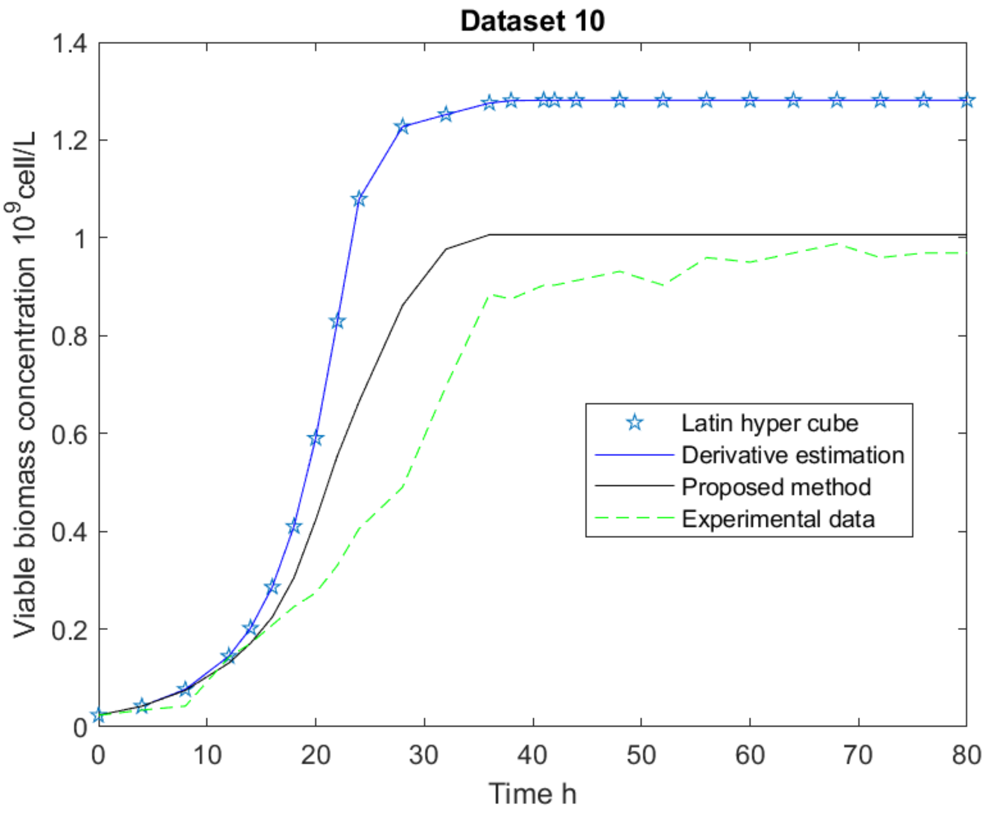


Figure 6.52 Performance results for Derivate estimation, Latin hyper cube, and proposed state substitution methods, for 10 different experimentally collected data sets.

6.4.4. The summary of results for the Zymomonas mobilis model

This case study is chosen to illustrate the practical application of the proposed state substitution method. It has twenty-six parameters making it not a light optimisation problem. It has model-process mismatch, as (Grisales Díaz and Willis 2019) were able to improve the model that it is used in this study. This is important as in practical applications model-process mismatch happens all the time and it is important that optimization algorithm can accommodate this mismatch. Using complexity analysis methods, complexity of this system was determined to be medium/high for the following reasons. System consist of four ODEs, with coupled interactions between states, but there is very little interaction between variables. Only ethanol state parameters are coupled with other states. PCA visualization shows multiple local minima, similarly to case study 3. This system does not have so many discontinuities as case study 3, but its overall convexity of the system is only 2%. Sorted minimization shows that model has one dominant global optimum, but its overall error value is high. This leads to suspect that identified parameter values are not 'true' values of the system, however they are statically most optimal values for highest model accuracy. Reason for such high error value at global optimum, lies with the fact that this model is not truly correct representation of the bio-system in question. This process-model mismatch is the cause of high error, but statically accurate parameter values. SOM analysis reveals multiple minima that are clustered into four clusters. Thus, making local minima have large separation. It should be noted that if gradient based algorithm gets into one of these clusters it will be stuck there. SOM parameter analysis also reveal two dominant variables u_max1 and K1x. It seems reasonable that these variables are dominant as from bio-system perspective, specific growth rate and substrate consumption rate are very highly linked with biomass growth. When comparing results of different parameter identification algorithms, we can see that multi-start method and Derivative estimation method both identified same solution. This is most likely due to fact they both ended up in the same cluster of local minima, whereas the proposed state substitution method ended up in different one. The proposed state substitution method almost twice as fast as other two methods, also the proposed state substitution method had better accuracy in seven out of ten data sets, when compared too other methods. In addition, for data sets 7 and 9, state of art methods failed to identify general trend, whereas the proposed state substitution method succeeded.

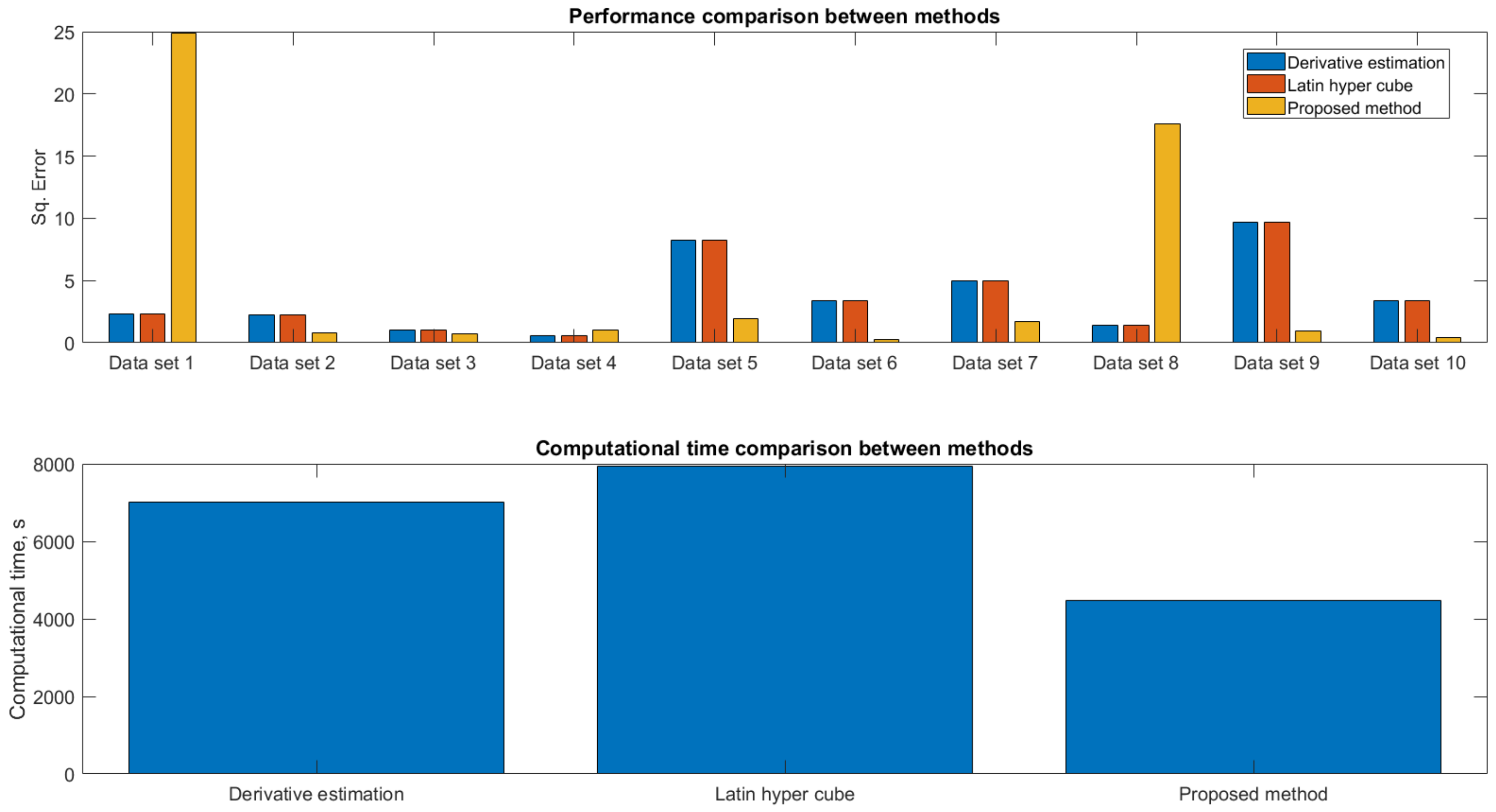


Figure 6.53 Summary of performance results for all three methods for Zymomonas mobilis model

7. Conclusions

During this research two algorithms were developed. Firstly, the proposed new method for parameter identification, and secondly complexity analysis algorithms. This chapter will cover, what was learned and achieved by developing and applying complexity analysis and new proposed state substitution method for the parameter identification of the four different case studies. We will look at each of the algorithm separately stating their strengths and weaknesses.

7.1. Complexity analysis

Complexity analysis consists of three different approaches, that should provide overall picture of the system complexity and some quantifiable measure of the systems complexity. This complexity analysis allows to compare the systems complexity relatively to other systems complexity but does not provide an absolute benchmark value of the complexity. Analysis algorithms are a) PCA visualization b) Sorted minimization c) SOM analysis.

a) PCA visualization is very effective for systems with smaller number of parameters, providing good insight on location of all local minimums. Combining it with convexity calculation, makes observations easier to interpret, due to distinction of convex and non-convex regions. This method can allow users to avoid local minimums, by selecting starting position, which when locally optimized tends towards global optimum. With increasing number of parameters PCA visualization loses its benefits, as PCA visualization uses only two first principal components to visualise error plane of the system. With increasing complexity and number of parameters, first two principal components no longer contain enough variability of overall system to be able to depict error plane accurately. Overall convexity percentage provides quantifiable measurement of complexity and is an accurate tool to compare complexity of different systems. It should be noted that very low convexity percentage systems (0-5%) are harder to compare between.

b) Sorted minimization is useful for identifying expected number of local minimums in the system, within constrained boundaries. In addition, sorted minimization determinates if the systems parameters are identifiable or just systems performance can be optimized. This helps to make objective decision about the optimization provided optimal parameter. Are these parameters real values or just statically optimal arbitrary values that makes model perform at its best. Although sorted minimization provides valuable information, it is most computationally demanding algorithm used for complexity analysis in this work. For systems, that has extremely large search spaces, this analysis can take even up to several days. For this reason, sorted minimization should only be performed if analysis is not time constrained.

c) SOM analysis provides similar visualization of error plane of the system as PCA visualization, but it is more robust when dealing with large number of parameters within the system. In terms of visualization, it is harder to interpret results of SOM than PCA visualization. Key observation that can be derived from SOM analysis, is parameter dominance. Positive and negative matching factor of each parameter provides insight into how dominant each parameter is in comparison with others. While algorithm resolution decrease with systems containing large number of parameters it is still able to pinpoint most dominant variable. This is important for use of proposed state substitution method as this information will help to set up best model hierarchy which will lead to best results.

All of three analysis methods should be used with caution as interpretation of results might lead to different conclusions. Nevertheless, each of these methods provide valuable information that can be used to understand level of complexity within the system and compared this complexity between different systems. These methods might also, allow to establish what makes the system complex and how to mitigate complication introduced by these complexities.

7.2. Proposed state substitution method

Proposed state substitution method was compared with two state of art methods - derivate and integral methods. Comparison was done using four case studies, with increasing complexity. In terms of computational time, proposed state substitution method performed worse than both state of art methods when optimizing simple system with single ODE (Case study 1). However, its advantages become increasingly more apparent as complexity of the systems rose. Where state of art performance was linearly decreasing with increasing complexity of the systems as expected, the proposed state substitution method not only performed better, but also gap between the state of art methods and the proposed state substitution method was increasing. In terms of robustness to noise, integral method and the proposed state substitution method were able to deal with higher level of noise, where the derivate method would fall short. In terms of accuracy the derivative method had worse average accuracy, throughout all case studies. The proposed state substitution method and the integral method, where able to keep similar accuracy levels in all four case studies. This would suggest that best course of action, when dealing with unknown system, is to perform complexity analysis first, then if system has high complexity use the proposed state substitution method approach instead of the state of art methods. This should lead to lower computational times, and same levels of accuracy. It should be noted that this computational time save, would be especially impactful, during model development where multiple different variations of model need to be tested to check which one produces best results.

8. Future work

This chapter will focus on limitations of this work and areas which can be expanded or improved. Chapter will cover both complexity analysis and proposed new method as independent methods.

Complexity analysis, main use is to assess the systems complexity. While discussed methods allows to do it, only overall convexity value and number of local minima can be used as numerical comparison, leaving other features to be interpreted by the user. This makes it not universal when used by different users as their assessment of topology of the SOMs and PCA visualization will vary. To make complexity analysis techniques useful in broader spectrum, there is a need for uniform framework of these type of analysis to allow uniform benchmarking process. Furthermore, methods used to evaluate systems complexity, where only tested on four case studies mentioned in this work, which is too small of a sample size to make general conclusions about methods usage to benchmark any problems. However, this work highlight possible uses in industry to identify systems that do require additional attention and might require to deviate from state of art methods to produce fast and accurate model capable of prediction. Excluding sorted minimisation, other techniques are fast and easy to implement, making them good for initial exploration of system complexity and helps to choose appropriate methodology for further analysis.

The proposed state substitution method seems to achieve significant reduction in computational time, when applied to complex systems, but case studies provided where full parametric models. While this shows that method is able to identify all parameters that are identifiable, it says nothing about its ability to work on hybrid models. While in theory there should be no change in methodology as long as non-parametric part can be evaluated by objective function this was not tested and cannot be stated to work. In addition, similarly the proposed state substitution method was extensively tested with four case studies described in this work. Even though these case studies cover large variability of the problems, it does not cover all types of problems. Furthermore, this work did not investigate optimizing algorithms that are part of the proposed state substitution method, such as ODE integrators. Nevertheless, this new approach allows to make exploration of model structure faster and more efficient. By reducing computational time of parameter identification, it allows researchers and industry to explore check more model structures within save time frame or introduce more complex model without sacrificing additional time required to identify them.

9. Appendix

Example code for case study 3. It should be noted not all functions are given and provided code will not work, if directly copied to the MATLAB environment. This is to give high-level overview of methodologies used in this work:

Derivative estimation main script example:

```
clear all
close all
profile on % turning profiler on to measure computational time

global par
global p
global w

w = [200,50,1,50,1,1]; % weight values for states manually inputted
sample = 0.1; % Sampling interval of generated data
noise = 0.1; % Noise level of the generated data
timeS = 0.001; % Sampling interval of the spline

Name = 'case study 3';
[fnc,x0,par0,a,b,t,ode,OPTS] = Idata(Name); % Pulling initial conditions for
selected case study

p = par0;
x02(:,1)    =    ((x0(:,1)-x0(:,2)).*rand(length(x0(:,1)),1)+x0(:,2));    %
Generating random staring point of unseen data

[y0n,y02n,y0,y02,t1] = DataGen(x0,sample,noise,t,fnc,OPTS,ode); % Generating
data, with (y0n) noise and without (y0) noise
[y03n,y03,~] = DataGen(x02,sample,noise,t,fnc,OPTS,ode);

% Cubic spline opproximatation
for i = 1:length(y0n(1,:))
    yy1(i,:) = csaps(t1,y0n(:,i));
    yy2(i,:) = csaps(t1,y02n(:,i));
end

% Sampling of the spline with smaller interval
for ii = 1:length(y0n(1,:))
    fx1(ii,:) = fnval(yy1(ii,:),0:timeS:t1(end));
    fx2(ii,:) = fnval(yy2(ii,:),0:timeS:t1(end));
end

% Derivative calculation of time series
for iii = 1:length(y0n(1,:))
    dx1(iii,:) = diff(fx1(iii,:))/timeS;
    dx2(iii,:) = diff(fx2(iii,:))/timeS;
end

opts                                                                        =
optimoptions(@fmincon,'Algorithm','sqp','Display','iter','MaxFunEvals',6400
); % Solver settings

% Optimisation of all estimated derivatives
for j = 1:(t1(end)/timeS)
```

```
[x(j,:),fval(j)]                                                       =
fmincon(@(p)DerivMonod(p,yy1,yy2,dx1(:,j),dx2(:,j),timeS*j),zeros(1,length(
p))+0.5,[],[],[],[],a,b,[],opts);
end

par = mean(x); % taking mean of the estimated parameter vector

Problem  =  @(par)MinError(par,fnc,x0,t1,y0n,ode,OPTS,y02n,t1);  %Setting  up
objective function for the global search
opts                                                                   =
optimoptions(@fmincon,'Algorithm','sqp','Display','iter','MaxFunEvals',3200
); % setting parameters for global search

problem = createOptimProblem('fmincon','objective',Problem,'x0',par,...
    'lb',a,'ub',b,'options',opts);
gs = GlobalSearch('Display','iter');
ms = MultiStart('Display','iter','MaxTime',60); % initializing global search
[xf,f,flag,table,residGS] = run(gs,problem); % Running global search

PPP = profile('info');

RunTime                                                                =
PPP.FunctionTable(structfind(PPP.FunctionTable,'FunctionName','DerivativeEs
t')).TotalTime; % Checking computational time
profile off % Stopping profiler
```

Multi-start script example:

```
clear all
close all
profile on % Turning profiler on to measure computational time

global w
global p

w = [200,50,1,50,1,1]; % weight values for states manually inputed
sample = 0.1; % Sampling interval of generated data
noise = 0.1; % Noise level of the generated data

Name = 'case study 3';
[fnc,x0,par0,a,b,t,ode,OPTS] = Idata(Name); % Pulling initial conditions for
selected case study

p = par0;
x02(:,1)     =     ((x0(:,1)-x0(:,2)).*rand(length(x0(:,1)),1)+x0(:,2));     %
Generating random staring point of unseen data

[y0n,y02n,y0,y02,t1] = DataGen(x0,sample,noise,t,fnc,OPTS,ode); % Generating
data, with (y0n) and without (y0) noise
[y03n,y03,~] = DataGen(x02,sample,noise,t,fnc,OPTS,ode);

pop = 20; % Population of the Latin hyper cube samples
N = length(par0); % Number of Variables
Best = pop*1; % Number of best initial cases after screening

lb = a; % Setting lower parameter boundries
ub = b; % Setting upper parameter boundries
X = lhsdesign(pop,N,'criterion','correlation');
```

```
D = bsxfun(@plus,lb,bsxfun(@times,X,(ub-lb))); % Initial conditions for latin
hyper cube

% Performing intial screening
for j = 1:pop
    E = MinError(D(j,1:N),fnc,x0,t1,y0n,ode,OPTS,y02n,t1);
    D(j,N+1) = E;
end

% Selecting fraction of best case initial conditions
D = sortrows(D,N+1);
D = D(1:round(Best/pop*size(D,1)),1:N+1);

fLow = 1e+10;

% initialing multi-start
for i = 1:size(D,1)
    disp({'Run Number:',i});
opts = optimoptions(@fmincon,'Algorithm','sqp','Display','iter');
problem                                                                     =
createOptimProblem('fmincon','objective',@(par1)MinErrorW(par1,fnc,x0,t1,y0
n,ode,OPTS,y02n,w,t1),...
'x0',D(i,1:N),'lb',lb,'ub',ub,'options',opts);

ms = MultiStart('Display','iter','MaxTime',60);
[xf,f] = run(ms,problem,1);

if f < fLow % updating parameter values, if error is lowest compared to other
runs
     Par = xf;
     fLow = f;
end
end

PPP = profile('info');

RunTime                                                                     =
PPP.FunctionTable(structfind(PPP.FunctionTable,'FunctionName','Latin')).Tot
alTime; % Checking computational time
profile off % Stopping profiler
```

State substation Method main script example:

```
clear all
close all
profile on % Turning profiler on to measure computational time

global w
global p

w = [200,50,1,50,1,1]; % weight values for states manually inputed
sample = 0.1; % Sampling interval of generated data
noise = 0.1; % Noise level of the generated data

Name = 'case study 3';
[fnc,x0,par0,a,b,t,ode,OPTS] = Idata(Name); % Pulling initial conditions for
selected case study

p = par0;
```

```
x02(:,1)     =     ((x0(:,1)-x0(:,2)).*rand(length(x0(:,1)),1)+x0(:,2));     %
Generating random staring point of unseen data

[y0n,y02n,y0,y02,t1] = DataGen(x0,sample,noise,t,fnc,OPTS,ode); % Generating
data, with (y0n) and without (y0) noise
[y03n,y03,~] = DataGen(x02,sample,noise,t,fnc,OPTS,ode);


options = optimset('Display', 'iter', 'MaxIter',200,...
                'LargeScale','on','Jacobian','on','DiffMaxChange',...
                0.1,'DerivativeCheck','off','MaxfunEvals',200,...
                'TolFun',1e-10, 'TolX',1e-10);
% local solver options

% Each state is solved individually based on hierarchy model
p0 = [0.5,0.5,0.5,0.001,0.001,0.001,0.001];
[p1,resnorm1,resid1,exitflag1,output1,lambda1,jacobian1]                    =
lsqnonlin(@Parameters1,p0,...
[0,0,0,0,0,0,0],[2,2,1,0.5,0.5,0.5,0.5],options);
par = p1;

p0 = [0.5,0.5,0.3,0.5,0.5];
[p2,resnorm2,resid2,exitflag2,output2,lambda2,jacobian2]                    =
lsqnonlin(@Parameters2,p0,...
[0,0,0,0,0],[2,1,0.5,5,50],options);
par(8) = p2(3);

p0 = [0.5,0.5,0.4];
[p4,resnorm4,resid4,exitflag4,output4,lambda4,jacobian4]                    =
lsqnonlin(@Parameters4,p0,...
[0,0,0],[2,1,1],options);
par(2) = mean([p1(2),p2(1),p4(1)]);
par(3) = mean([p1(3),p2(2),p4(2)]);
par(12) = p4(3);

p0 = [0.5,0.5,0.5];
[p3,resnorm3,resid3,exitflag3,output3,lambda3,jacobian3]                    =
lsqnonlin(@Parameters3,p0,...
[0,0,0],[5,5,50],options);
par(9) = p3(1);
par(10) = mean([p2(4),p3(2)]);
par(11) = mean([p2(5),p3(3)]);

p0 = [0.5];
[p5,resnorm5,resid5,exitflag5,output5,lambda5,jacobian5]                    =
lsqnonlin(@Parameters5,p0,...
[0],[1],options);
par(13) = p5(1);

p0 = [0.5,0.5,0.5];
[p6,resnorm6,resid6,exitflag6,output6,lambda6,jacobian6]                    =
lsqnonlin(@Parameters6,p0,...
[0,0,0],[10,50,0.5],options);
par([14,15,16]) = p6([1,2,3]);

w = [200,50,1,50,1,1]; % weigths for state
fnc = @(b)problem(b); % Setting up objective function for the global solver

lb = a; % Setting lower parameter boundries
ub = b; % Setting upper parameter boundries
```

```
opts                                                              =
optimoptions(@fmincon,'Algorithm','sqp','Display','iter','MaxFunEvals',3200
); % Global solver options
problem = createOptimProblem('fmincon','objective',fnc,'x0',par,...
    'lb',lb,'ub',ub'options',opts);
gs = GlobalSearch('Display','iter');
ms = MultiStart('Display','iter','MaxTime',60);
[xf,f,flag,table,residGS] = run(gs,problem); % Running global search

PPP = profile('info');
RunTime                                                           =
PPP.FunctionTable(structfind(PPP.FunctionTable,'FunctionName','Latin')).Tot
alTime; % Checking computational time
profile off % Stopping profiler
```

PCA visualization algorithm main script example:

```
clear all
close all

global w
global p

w = [200,50,1,50,1,1]; % weight values for states manually inputted
sample = 0.1; % Sampling interval of generated data
noise = 0.1; % Noise level of the generated data
pop = 5000; % Population of the samples

Name = 'case study 3';
[fnc,x0,par0,a,b,t,ode,OPTS] = Idata(Name); % Pulling initial conditions for
selected case study

p = par0;
x02(:,1)    =    ((x0(:,1)-x0(:,2)).*rand(length(x0(:,1)),1)+x0(:,2));    %
Generating random staring point of unseen data

[y0n,y02n,y0,y02,t1] = DataGen(x0,sample,noise,t,fnc,OPTS,ode); % Generating
data, with (y0n) and without (y0) noise
[y03n,y03,~] = DataGen(x02,sample,noise,t,fnc,OPTS,ode);


[Answer,ACP,y0,Sample,CC,Error,Cluster,NoC,y02]                   =
Convex(x0,t1,par,a,b,fnc,pop,noise,ode,OPTS,t2);% function that calculates
convexivity of #pop triple samples in percentage


[coff,score] = pca(Sample',2); % Calculating scores of the first two principal
components.
```

SOM main script example:

```
clear all
close all

global w
global p

w = [200,50,1,50,1,1]; % weight values for states manually inputed
```

```
sample = 0.1; % Sampling interval of generated data
noise = 0.1; % Noise level of the generated data

Name = 'case study 3';
[fnc,x0,par0,a,b,t,ode,OPTS] = Idata(Name); % Pulling initial conditions for
selected case study

p = par0;
x02(:,1)    =    ((x0(:,1)-x0(:,2)).*rand(length(x0(:,1)),1)+x0(:,2));    %
Generating random staring point of unseen data

[y0n,y02n,y0,y02,t1] = DataGen(x0,sample,noise,t,fnc,OPTS,ode); % Generating
data, with (y0n) and without (y0) noise
[y03n,y03,~] = DataGen(x02,sample,noise,t,fnc,OPTS,ode);

pop = 5000; % population size of the latin hyper cube sampling
N = length(par); % Variables
lb = a;
ub = b;
XX = lhsdesign(pop,N,'criterion','correlation');
D = bsxfun(@plus,lb,bsxfun(@times,XX,(ub-lb)));

w = [200,50,1,50,1,1]; % weigths for state

for i = 1:length(D)
    Error(i) = MinErrorW(D(i,:),fnc,x0,t,y0,ode,OPTS,y02,w,t); % optimizing
initial samples points
end

% Removing all failed integration samples and sorting from best to worst
E = Error(Error<2e+10);
Esort = sort(E);

% Labelling based samples based on error value
EE(1:length(E))= {'H'};
EE(E<Esort(round(0.5*length(E))))= {'M'};
EE(E<Esort(round(0.05*length(E))))= {'L'};

SomData                                                                     =
som_data_struct(D(Error<2e+10,:),'name','SOMdata','labels',EE','comp_names'
,Names);
Map = som_make(SomData, 'algorithm', 'batch','shape','cyl'); % defaults batch
traininf used (faster than sequential training)
Map = som_autolabel(Map,SomData,'vote'); %vote = most hits dictates class!

x1=find(ismember(SomData.labels,'H')==1);
x2=find(ismember(SomData.labels,'M')==1);
x3=find(ismember(SomData.labels,'L')==1);

h1 = som_hits(Map,SomData.data(x1,:));
h2 = som_hits(Map,SomData.data(x2,:));
h3 = som_hits(Map,SomData.data(x3,:));

    c1=ismember(Map.labels,'H')*1;         %Class  1  (Application  for  iris
Accepted)
    c2=ismember(Map.labels,'M')*2;         %Class  2  (Application  for  iris
Rejected)
    c3=ismember(Map.labels,'L')*3;        %Class 3 Empty
    c4=ismember(Map.labels,'')*4;

    classes=c1+c2+c3+c4;
    col_mat=zeros([length(classes),3]); % Initialise Matrix of colors to be
associated with each node

    for count=1:length(classes)
        if ismember(classes(count),1)==1
            col_mat(count,:)= [1,0,0];
        elseif ismember(classes(count),2)==1        %This section finds and
replaces with the correct colors
            col_mat(count,:)=[0,0,1];
        elseif ismember(classes(count),3)==1        %This section finds and
replaces with the correct colors
            col_mat(count,:)=[0,1,0];
        elseif ismember(classes(count),4)==1
            col_mat(count,:)=[0,0,0];
        end
    end

    C=som_clustercolor(Map,classes,col_mat);  %  coloring  based  most  voted
clusters

    for i = 1:length(Map.codebook)
        Error1(i)                                                           =
MinErrorW(Map.codebook(i,:),fnc,x0,t,y0,ode,OPTS,y02,w,t);   %   Calculating
error of the mean cluster parameter values
    end

    Error1(2,:) = 1:length(Error1);
    Error1 = sortrows(Error1',1);
    CG = gray(length(Error1)); % Creating grayscale vector based on the amount
of samples
    RCG(Error1(:,2),:) = CG; % assigning correct grayscale values to each
error value

[Like,LikeI,Diff]  =  Pmatch(Map,RCG);  %  Comparing  SOM  error  plane  with
parameter error plane in grayscale
```

10. References

Abadlia, Issam, et al. (2020), 'Adaptive fuzzy control with an optimization by using genetic algorithms for grid connected a hybrid photovoltaic–hydrogen generation system', *International Journal of Hydrogen Energy,* 45 (43), 22589-99.

Abdel Moamen, O. A., *et al*. (2015), 'Factorial design analysis for optimizing the removal of cesium and strontium ions on synthetic nano-sized zeolite', *Journal of the Taiwan Institute of Chemical Engineers,* 55, 133-44.

Aguirre, L.A and Billings, S.A (1995), 'Retrieving dynamical invariants from chatotic data using NARMAX models', *Int. J. Bifurcation and Chaos,* 5, 449 - 74.

Almeida, Jonas S (2002), 'Predictive non-linear modeling of complex data by artificial neural networks', *Current Opinion in Biotechnology,* 13.

Anonymous 'ODE45 solver options', <https://uk.mathworks.com/help/matlab/ref/odeset.html#input_argument_d0e708771>, accessed 2017/05/11.

Bardow, Andrew and Marquardt, Wolfgang (2004), 'Incremental and simultaneous identification of reaction kinetics: methods and comparison', *Chemical Engineering Science,* 59, 2373-684.

Biró, E., *et al*. (2009), 'Three-step experimental design to determine the effect of process parameters on the size of chitosan microspheres', *Chemical Engineering and Processing: Process Intensification,* 48 (3), 771-79.

Bomhoff, Matthijs, Kern, Walter, and Still, Georg (2010), 'On bounded block decomposition problems for under-specified systems of equations', *Memorandum 1930*.

Breeden, J.L and Hubler, A. (1990), 'Reconstructing equations of motion from experimental data with unobserved variables', *Phys. Rev.,* A42, 5817 - 26.

Charles Audet; J. E. Dennis, Jr. (2002), 'Analysis of generalized pattern searches', *Society for Industrial and Applied Mathematics,* 13, 889–903.

Chou, I. C. and Voit, E. O. (2009), 'Recent developments in parameter estimation and structure identification of biochemical and genomic systems', *Math Biosci,* 219 (2), 57-83.

Cremers, J. and Hubler, A. (1987), 'construction of differental equations from experimental data', *Z. Naturforsch,* 42a, 797 - 802.

Crutchfield, J. P. and McNamara, B.S. (1987), 'Equations of motion from data series', *Compl. Syst.,* 1, 417 - 51.

D.I. Kamenski and Dimitrov, S.D. (1993), 'Parameter estimation in differential equations by applications of rational functions', *Computers & Chemical Engineering,* 17, 643-51.

Degasperi, A., Fey, D., and Kholodenko, B. N. (2017), 'Performance of objective functions and optimisation procedures for parameter estimation in system biology models', *NPJ Syst Biol Appl,* 3, 20.

documentation;, MATLAB (2020), 'Matlab global solver comparison', <https://www.mathworks.com/help/gads/example-comparing-several-solvers.html>, accessed 30/05/2020.

Eilers, P.H.C. (2003), 'A perfect smoother', *Analytical Chemistry,* 75, 3631-36.

Fausett, L. (1994), *Fundamental of Neural Networks: Architectures, Algorithms, and Applications*.

Froment, G.F. and Bischoff, K.B. (1990), *Chemical reaction analysis and design.* (New York: Wiley).

Glover, F (1998), 'A template for scatter search and path relinking', *Artificial Evolution* 13-54.

Goldberg, David E (1989), *Genetic Algorithms in Search, Optimization & Machine Learning*.

Gotshall, Stanley and Rylander, Bart (2002), 'Optimal population size and the genetic algorithm', *2nd WSEAS International Conference of Soft Computing, Optimization, Simulation and Manufacturing Systems*.

Gouesbet, G (1991), 'Reconstruction of the vestor fields of continuous dynamical systems from scalar time series', *Phys. Rev.,* A43, 5321 - 31.

Grisales Díaz, Víctor Hugo and Willis, Mark J. (2019), 'Ethanol production using Zymomonas mobilis: Development of a kinetic model describing glucose and xylose co-fermentation', *Biomass and Bioenergy,* 123, 41-50.

Gutmann, H.-M. (2001), 'A Radial Basis Function Method for Global Optimization', *Journal of Global Optimization,* 19, 201–27.

Hass, Helge, *et al*. (2018), 'Benchmark Problems for Dynamic Modeling of Intracellular Processes', *Bioinformatics*.

Hegger, R. *et al*. (1998), 'Dynamical properties of a ferroelectric capacitor observed through nonlinear time series analysis', *Chaos,* 8, 727 - 36.

Holland, D.H. and Rayford, G.A. (1989), *Fundamentals of chemical reaction enginerring* (New Jersey: Prentice-Hall).

J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas. 'SOM Toolbox for Matlab 5. Laboratory of Computer and Information Science (CIS)', <http://www.cis.hut.fi/somtoolbox/package/papers/techrep.pdf>, accessed.

J.S. Almeida, E.O. Voit (2003), 'Neural-network-based parameter estimation in S-system models of biological networks', *Genome Informatics,* 14.

James Kenedy, Russell Ebehart (1995), 'Particle swarm optimization', *IEEE*, 1942-48.

Kadtke, J., Brush, J., and Holzfuss, J. (1993), 'Global dynamical equations and Lyapunov exponents from noisy chaotic time series', *Int. J. Bifurcation and Chaos,* 3 (607 - 616).

Kikuchi, S., *et al*. (2003), 'Dynamic modeling of genetic networks using genetic algorithm and S-system', *Bioinformatics,* 19 (5), 643-50.

Kutalik, Z., Tucker, W., and Moulton, V. (2007), 'S-system parameter estimation for noisy metabolic profiles using newton-flow analysis', *IET Syst Biol,* 1 (3), 174-80.

Leonori, Stefano, *et al*. (2020), 'Optimization strategies for Microgrid energy management systems by Genetic Algorithms', *Applied Soft Computing,* 86.

Levenberg, Kenneth (1944), 'A Method for the Solution of Certain Non-Linear Problems in Least Squares', *Quarterly of Applied Mathematics,* 2.

Lu, Jiawei, *et al*. (2020), 'Surrogate modeling-based multi-objective optimization for the integrated distillation processes', *Chemical Engineering and Processing - Process Intensification*.

M. Vilela, C.C. Borges, S. Vinga, A.T. Vasconcelos, H. Santos, E.O. Voit, J.S. Almeida (2007), 'Automated smoother for the numerical decoupling of dynamics models', *BMC Bioinform.,* 8.

Marin-Sanguino, A., *et al*. (2007), 'Optimization of biotechnological systems through geometric programming', *Theor Biol Med Model,* 4, 38.

Mark K, Transtruma; James P, Sethnaa (2012), 'Improvements to the Levenberg-Marquardt algorithm for nonlinear least-squares minimization', *Journal of Computational Physics*.

Mascarenhas (2013), 'The divergence of the BFGS and Gauss Newton Methods', *Mathematical Programming,* 147, 253-76.

McKay, M. D., Beckman, R. J., and Conover, W. J. (1979), 'Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code', *Technometrics,* 21 (2), 239-45.

Michael Wetter, and Jonathan Wright (2003), 'Comparison of a generalized pattern search', *Eighth International IBPSA Conference*, 1401-08.

Mittelhammer, Ron C.; Miller, Douglas J.; Judge, George G. (2000), *Econometric Foundations.* 197–98.

Niemann, Henrik and Miklos, Robert (2014), 'A Simple Method for Estimation of Parameters in First order Systems', *Journal of Physics: Conference Series,* 570 (1).

Nigel, Meade; Towhidul, Islam (1995), 'Prediction Intervals for Growth Curve Forecasts', *Journal of Forecasting,* 14, 413-30.

P.J. Green and Silverman, B.W. (1994), *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*.

Pearson, Karl. (1901), 'On lines and planes of closest fit to systems of points in space', *Philosophical Magazine,* 2, 559-72.

Poli, Riccardo, Kennedy, James, and Blackwell, Tim (2007), 'Particle swarm optimization', *Swarm Intelligence,* 1 (1), 33-57.

Powell, M. J. D (1992), 'The Theory of Radial Basis Function Approximation ', *Advances in Numerical Analysis,* 2, 105–210.

R.L. Burden and Faires, J.D. (1993), *Numerical Analysis*.

Rasmus Bro, Age K, Smilde (2014), 'Principal component analysis', *Analytical Methods,* 6, 2812-31.

Raue, A., *et al*. (2013), 'Lessons learned from quantitative dynamical modeling in systems biology', *PLoS One,* 8 (9), e74335.

Riley, B. S. and Li, X. (2011), 'Quality by design and process analytical technology for sterile products--where are we now?', *AAPS PharmSciTech,* 12 (1), 114-8.

Salata, Ferdinando, *et al*. (2020), 'Effects of local conditions on the multi-variable and multi-objective energy optimization of residential buildings using genetic algorithms', *Applied Energy,* 260.

Saraiva, I., Vande Wouwer, A., and Hantson, A. L. (2015), 'Parameter identification of a dynamic model of CHO cell cultures: an experimental case study', *Bioprocess Biosyst Eng,* 38 (11), 2231-48.

Simeone, Marino; Eberhard O. Voit (2006), 'An automated procedure for the extraction of metabolic network information from time series data', *Journal of Bioinformatics and Computational Biology,* 4.

Spyridon Dallas, Konstantinos Machairas, Evangelos Papadopoulos (2017), 'A Comparison of ODE Solvers for Dynamical Systems with Impacts', *J. Comput. Nonlinear,* 12 (6), 8.

Steijns, A.N.R Bos; L. Lefferts; G.B. Marin; M.H.G.H (1997), 'Kinetic research on heterogeneously catalysed processes: A questionnaire on the state-of-the-art in industry', *Applied catalysis,* 160, 185-90.

T.Kohonen (1982), 'Self-organized formation of topologically correct feature maps', *Biological Cybernetics,* 43, 59-63.

--- (1990), 'The self-organizing map', *Proceedings of the IEEE,* 78, 1464-80.

Timmer J, *et al*. (2000), 'Parameter estiamtion in nonlinear stochastic differential equations', *Chaos solid. Fract.,* 11, 2571 -78.

Ugray, Zsolt, Leon Lasdon, John C. Plummer, Fred Glover, James Kelly, and Rafael Martí. (2007), 'Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization', *INFORMS Journal on Computing,* 19, 328–40.

Voit, E. O. (1982), 'Power-law approach to modeling biological-systems', *Journal of fermentation technology,* 60, 233-41.

Voit, E. O. and Almeida, J. (2004), 'Decoupling dynamical systems for pathway identification from metabolic profiles', *Bioinformatics,* 20 (11), 1670-81.

Voss, Henning U. and Timmers, Jens (2004), 'Nonlinear dynamical system identification from uncertain and indirect measurements', *International Journal of Bifurcation and Chaos,* 14 (6), 1905-33.

W. Natita, W. Wiboonsak, and S. Dusadee (2016), 'Appropriate Learning Rate and Neighborhood Function of Self-organizing Map (SOM) for Specific Humidity Pattern Classification over Southern Thailand', *International Journal of Modeling and Optimization,* 6 (1), 61-65.

Wellstead, P., *et al*. (2008), 'The rôle of control and system theory in systems biology', *Annual Reviews in Control,* 32, 33-37.

Whittaker, E. T. (1923), 'On a New Method of Graduation', *Proc. Edinburgh Math. Soc*.

Willis, Mark J. and Stosch, Moritz von (2016), 'Inference of chemical reaction networks using mixed integer linear programming', *Computers & Chemical Engineering,* 90, 31-43.

Wolkenhauer, O., *et al*. (2014), 'Enabling multiscale modeling in systems medicine', *Genome Medicine,* 6.

Y. Maki and Tominaga, D. (2001), 'Development of a system for the inference of large scale genetic networks', *Pacific Symposium on Biocomputing,* 6, 446-58.

Yeow, Y. Leong, *et al*. (2003), 'A new method of processing the time-concentration data of reaction kinetics', *Chemical Engineering Science,* 58 (16), 3601-10.

Yu, L. X. and Woodcock, J. (2015), 'FDA pharmaceutical quality oversight', *Int J Pharm,* 491 (1-2), 2-7.

Zhao, Y., C. Jiang, and A. Yang (2012), 'Towards computer-aided multiscale modelling: An overarching methodology and support of conceptual modelling', *Computers & Chemical Engineering,* 36, 10-21.