NTNU
Norwegian University of
Science and Technology

Searching Historic Data For Model Identification

Eirik Viksmo-Slettan

Master of Science in Engineering Cybernetics
Submission date: Januar 2012
Supervisor: Lars Imsland, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract

The focus of this thesis is obtaining useful segments of historic data from normal operation due to, for instance, economical and safety concerns of performing experiments on a plant. This will be done by development, and implementation of a user friendly concept. The concept will give the user a collection of search criteria to choose from, and the ability to combine them, to locate possible informative data in historic data. The angle of attack will be more practical than theoretical.

The concept SHDMI is proposed as such a tool. As a prototype of the concept it shows promise dealing with both SISO- and MIMO-systems in its simplicity and structure, but it is concluded that improvements and/or expansions of SHDMI is needed if it is to become a reliable and adequate for its purposes.

Preface

This master's thesis is the culmination of my M.Sc. degree in Engineering Cybernetics at the Norwegian University of Science and Technology.

I would like to thank my supervisor Prof. Lars Imsland for his good advise and guidance throughout the semester.

I would also like to thank my friends at NTNU for making the years here so enjoyable. A special thanks goes out to Jonas Magnussen for helping with the correction of the thesis.

Last, but not least I would to thank my family for all their support throughout my time at NTNU.

Eirik Viksmo-Slettan
Trondheim, 29 jan. 2012

Contents

List of Figures

Chapter 1

Introduction

In modern monitoring and control of processes and plants (e.g. a chemical production plant), a mathematical representation, a model of the its dynamic behaviour is required to apply (for instance) a Model Predictive Controller(MPC) to the process. Such a dynamic model can be derived from the system identification procedure, based on observed data of the physical process at hand. A data set, containing the observed data, connects input and output variables of the system. The data set is preferably obtained by performing experiments on the process, specially designed to excite its every mode of operation, and in turn describe its dynamic behaviour accurately.

If a process is physically altered, e.g. replacement or deterioration of integral parts, its dynamic model must be updated. To perform a new experiment on the process, and update its model, the process has to be taken out of operational order, which in many cases is not preferable or even possible due to reasons of regularity, economy, safety, etc.

The alternative to obtaining data by performing such experiments is to use historic data observed at normal operation, in the system identification procedure. Using such data can be problematic, primarily due to segments of missing data and long periods of stationary data, which poorly describes the dynamic behaviour of the process, rendering the data as useless.

Scientific books on the subject, such as Ljung (1999), Goodwin & Payne (1977) and Pintelon & Schoukens (2001), does not suggest any methods for locating informative data segments in historic data sets. Ljung (1999) states

that: *"The procedure of how to select such segments will basically be subjective and will have to rely mostly upon intuition and process insights"*.

1.1 Dynamic Identifiability Analysis - DYNIA

In Fordal (2010) it is proposed to use the Dynamic Identifiability Analysis (DYNIA) method to extract information about dynamic systems from historic/operational data.

DYNIA was developed by Dr. Wagener, associate professor of civil engineering at Penn State University. The method was first introduced in Wagener et al. (2002), and was described in more detail in Wagener, McIntyre, Lees, Wheater & Gupta (2003). DYNIA was originally developed for use in the field of hydrology, but no assumptions made implicate that the methods performance should degrade regarding general dynamic mathematical models. An extensive review of the DYNIA-method is not in the scope of this text, so an overview based on Wagener et al. (2003) and Fordal (2010) will be presented in this section. For a more comprehensive study of the method, reviewing Fordal (2008), Fordal (2010) and other refereed articles is advised.

Wagener et al. (2003) describes DYNIA as an approach for locating periods of high identifiability for individual parameters and to detect failures of model structures in an objective manner. The method draws on elements from Regional Sensitivity Analysis (RSA) (Spear & Hornberger 1980, Hornberger & Spear 1981), the Generalized Likelihood Uncertainty Estimation Framework (GLUE) (Beven 2004, Stedinger, Vogel, Lee & Batchelder 2008), wavelet analysis (Goswami & Chan 1999) and applications of the Extended Kalman Filter (Beck 1985, Beck 1987).

The original RSA approach investigates whether the parameter distribution change when conditioned on a measure of performance (e.g. an objective function). The sensitivity of the model response due to changes in the parameter is indicated by deviations from an initially uniform distribution, and the differences between parts of the distribution performing well and poorly (behavioural and non-behavioural) (Spear & Hornberger 1980). In DYNIA

this approach is extended to assess the identifiability of parameters, not just their sensitivity.

Figure 1.1 represents the basic steps of the DYNIA-procedure. The feasible parameter space is examined by Monte Carlo sampling based on a uniform prior distribution. Each parameter has an associated objective function which is transformed into a support measure[1], and higher values indicate better performing parameter values. These parameter values are shown in the form of a dotty plot (part a) in Figure 1.1. The top 10 % (for instance) performing parameter values are selected and their cumulative support are calculated (part b) in Figure 1.1. A straight line indicates a poorly identified parameter, i.e. the highest support values are widely distributed over the feasible range. When a parameter is conditioned by the objective function used it is indicated by deviations from this straight line. An indicator of strength of the conditioning, and the identifiability of the parameter is the marginal probability distribution of the parameter, which is the gradient of the cumulative support. By segmenting the range of each parameter and calculating the gradient in each segment one get the schematic distribution shown in part d) of Figure 1.1. The highest value indicates the location of greatest identifiability of the parameter. Different model structure in terms of parameter uncertainty [2].

Using a moving window approach when calculating the parameter identifiability one can investigate the identifiability as a function of time (part e of Figure 1.1), and the gradient distribution aggregates the residuals within the window of time. Results for each parameter per time, plotted in colour coding indicates areas of higher (darker colour) identifiability.

1.1.1 Fordals Implementation - A Users Point of View

An implementation of the DYNIA-method by Dr. Wagener is available in the Matlab toolbox MCAT (Wagener & Weather 2004), but the source code of this implementation is protected and unavailable. To apply DYNIA as a

[1] All support measures have the characteristic that they sum to unity.

[2] assumed the opposite of identifiability can be compared by this measure of identifiability.

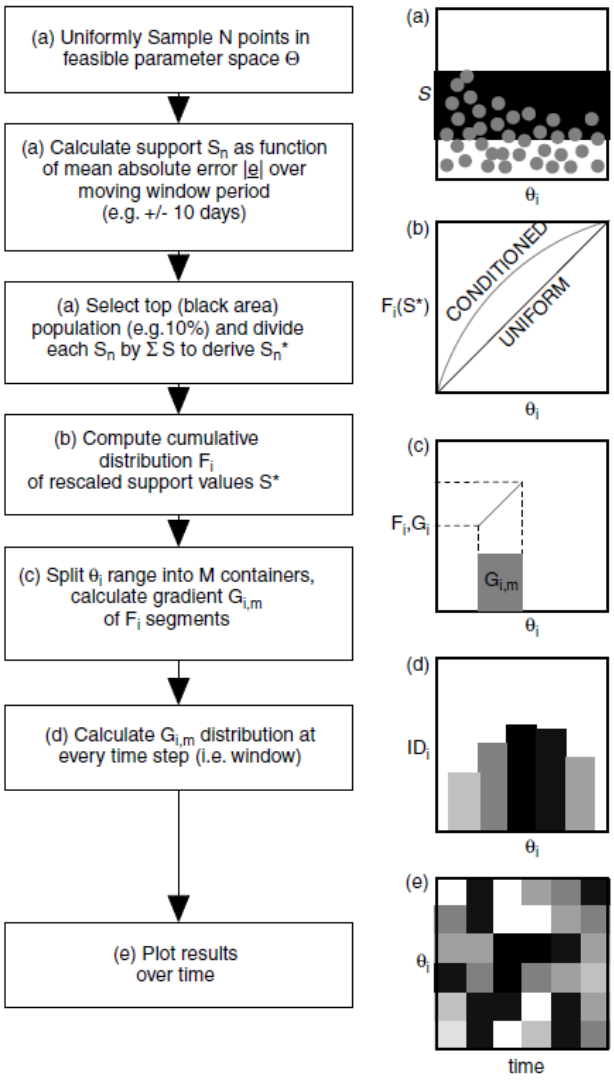

Figure 1.1: The DYNIA procedure, courtesy of (Wagener et al. 2002).

tool for extracting information about dynamic systems from historic data, and gain a more thorough understanding of, and to be able to suggest and test modifications to the method, Fordal (2010) implemented the method in Matlab. A scheme of the implementation is presented in Figure 1.2. Reviewing Fordal (2010) is advised if one wants a comprehensive study of the implementation. This will not be elaborated on in this thesis.

Looking at this implementation from a users viewpoint, it is clear that that the method demands a lot to execute with satisfying results. This is also mentioned as a concern in Fordal (2010), as no guidelines for setting feasible parameter ranges, a window size, number of Monte Carlo simulations, and tolerance limits for the behavioural parameter sets, are available. Another concern is the methods exponential time complexity. For large quantities of data, the running time of the method is considerable.

Replicating an example from Fordal (2010), with the before mentioned parameters of concern set by Fordal, considering the first order LTI SISO-system given in Equation 1.1 and Figure 1.3 [3], yields the results given in Figure 1.4 and 1.5.

$$\frac{y}{u}(s) = \frac{k}{T_1 s + 1} \qquad k = 2,\ T_1 = 3 \tag{1.1}$$

Figure 1.4 implies that the identifiability of the gain parameter is high when the system is at steady state[4], and the it is estimated to be in the vicinity of 2. Further, Figure 1.5 implies that the identifiability of the time constant parameter is high after the last step response of the system[5], and is estimated to be in the vicinity 3. The method appears to make satisfactory estimations of the system parameters, but Fordal proposes that the method has to be researched and developed further to function as a tool.

Another approach is to start from scratch, and develop a new concept for searching historic data for model identification purposes, with focus on the user.

[3] Some noise added to output signal to mimic measurement noise.

[4] Which seems peculiar as an possible bias between input and output would only be apparent at a step response.

[5] Again, the responses of the steps earlier in the data set does not seem to contain information about the system, which is peculiar.

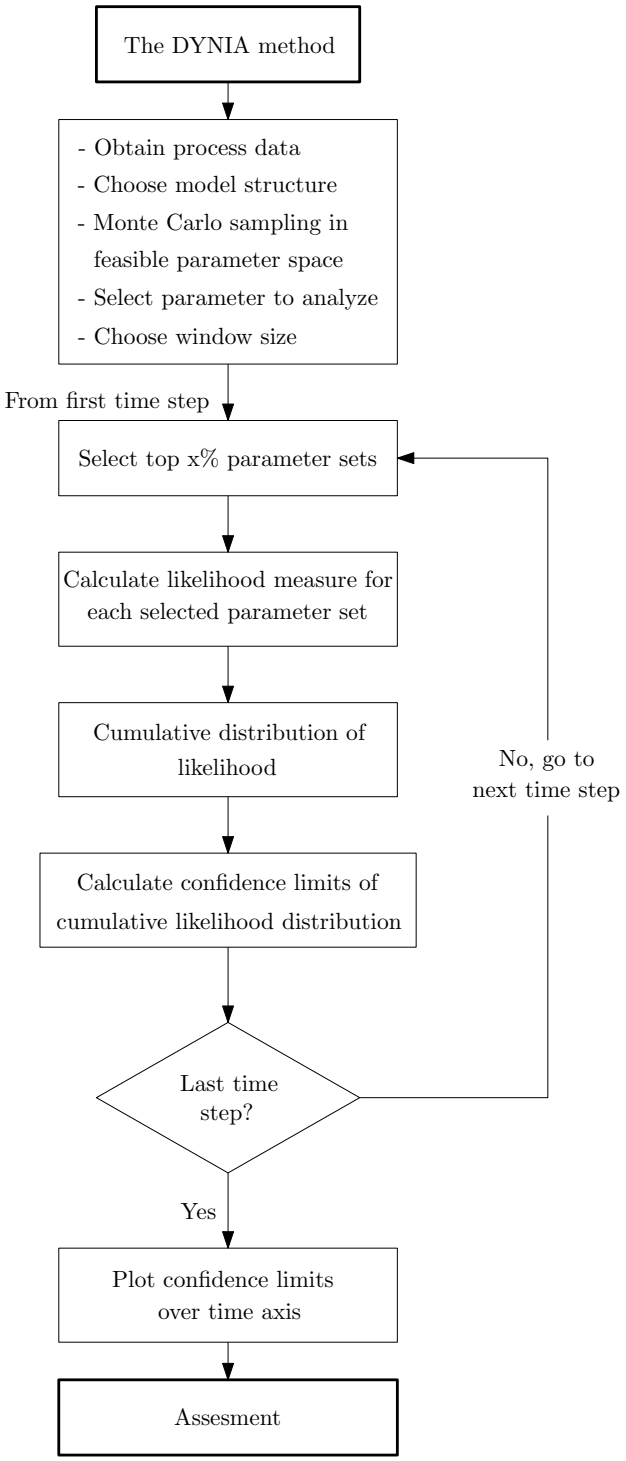

Figure 1.2: A scheme of the DYNIA implementation by Fordal (2010).

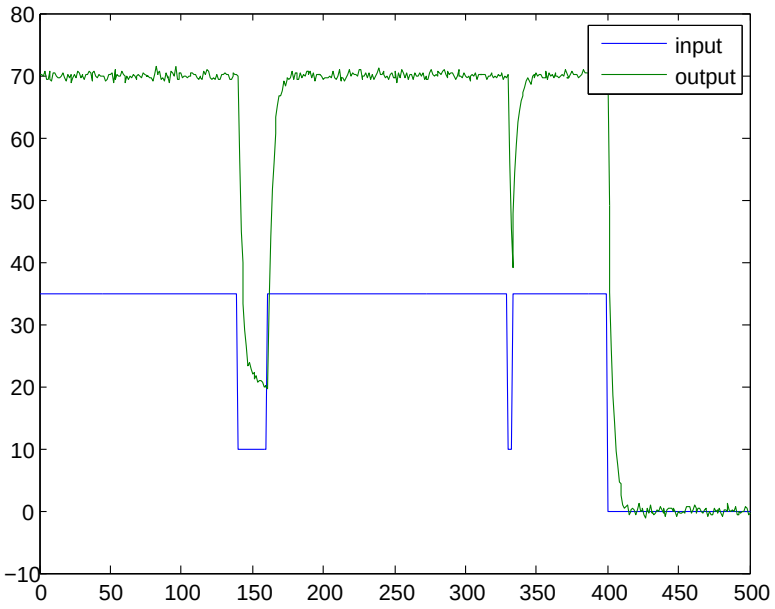

Figure 1.3: First order LTI SISO-system.

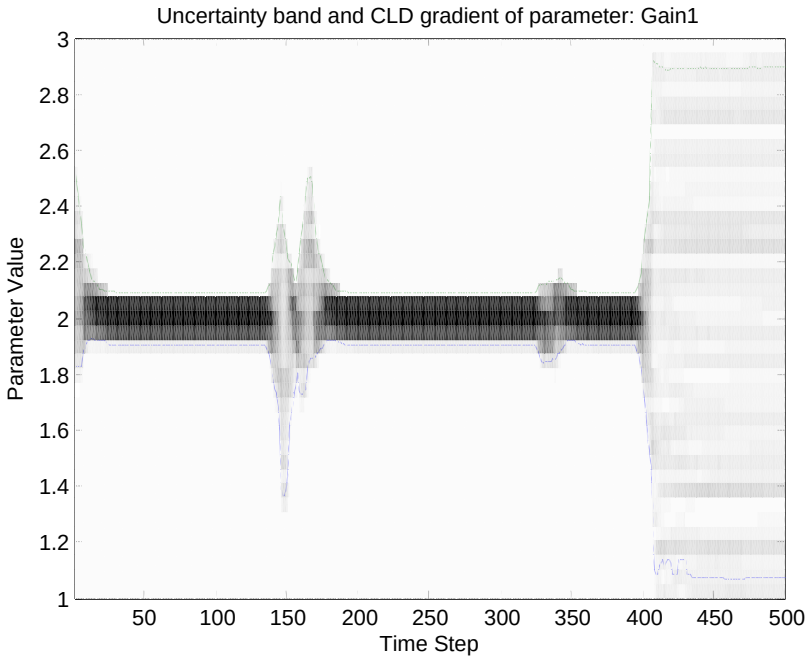

Figure 1.4: Parameter identifiability plot of the gain parameter k.

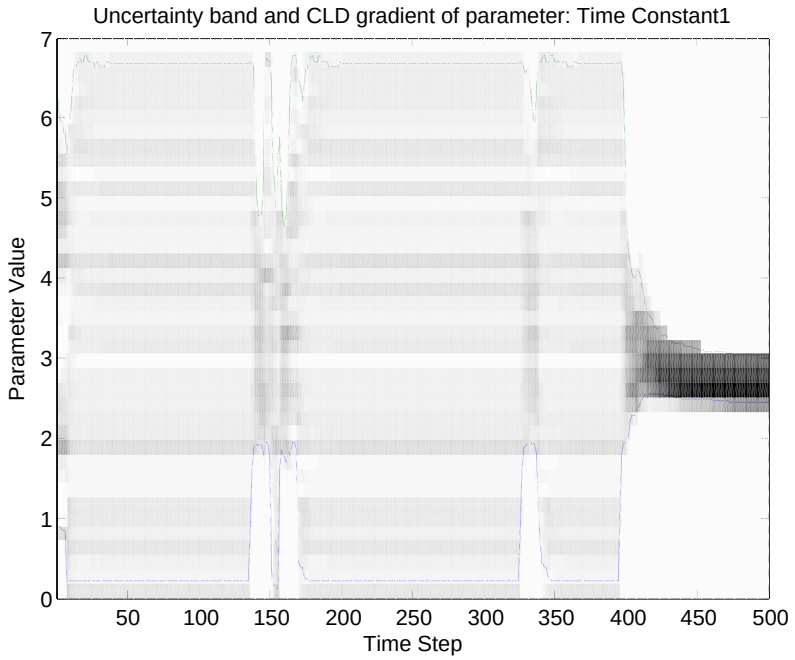

Figure 1.5: Parameter identifiability plot of the time constant parameter T_1.

1.2 Concept

Motivated by the complexity of DYNIA from a users point of view, and the statement of Ljung (1999), regarding selection of data segments in historic data, the main objective of this thesis will be to develop and implement a concept for obtaining appropriate data segments from historic data for model identification, that is easy to use(i.e. the user supplies the concept/method with basic knowledge of dynamic system behaviour, so that it can decide which data segments(if any present in the data set at all) to use in model identification). The concept will primarily be implemented as a method to deal with Single-Input Single-Output(SISO)-systems affected by noise and time delays. The possibility of expanding the concept to deal with Multiple-Input Multiple-Output(MIMO)-systems will also be explored.

1.3 Outline of the Thesis

- **Chapter 2:** Background theory.
- **Chapter 3:** Design and implementation of the proposed concept.
- **Chapter 4:** Expansion of the concept for use with MIMO-systems.
- **Chapter 5:** Discussion of the proposed concept.
- **Chapter 6:** Conclusions and recommendation for further work.

Chapter 2

Theory

One of the main attributes of the concept developed in this thesis, is that it should be simple and easy to use. A practical (not 'theory heavy') approach is therefore taken, using well known and relatively simple criteria to distinguish interesting from not interesting data segments of the historic data set. The term interesting is used instead of informative, as no measure for informative data is used, and only basic knowledge of system dynamics is exercised as mentioned before. As background material a brief introduction to system identification, based on Ljung (1999), is presented.

2.1 System Identification

As mentioned, the system identification procedure is the process of developing a mathematical model to describe a systems behaviour. This is achieved by selection of a model structure with a corresponding set of parameters which give a high degree of similarity between simulated (model) and observed (actual system) output data. Ljung (1999) states that the system identification procedure consists of three basic entities:

1. A data set Z^N.
2. A set of candidate models.
3. A rule by which the candidates can be assessed using the data.

2.1.1 The Data Set

The data set, Z^N, contains input and output data recorded from a process. This recording often takes place during a specially designed experiment, where the user may choose the input signals as well as which measurements to record, and when they are to be recorded. The objective of this experiment is to make these choices so one can ensure that the data set recorded is maximally informative. If one does not have the opportunity to perform such an experiment, due to economy, safety, etc., one can use data from normal operation of the plant[1]. As mentioned earlier, the main task in this thesis is to obtain interesting segments of such historic data sets, and its focus rarely ventures beyond this objective.

2.1.2 Candidate Models

The choice of candidate models is considered the most important and difficult in the system identification procedure. The candidate model set must be specified within a collection of models is to be assessed, e.g. linear models. There are different ways to build a model for such a candidate set, such as *blackbox* models. These are standard linear models, whose parameters are adjusted to fit the data, without reference to physical attributes of the system. *Graybox* model sets, accordingly, have adjustable parameters with physical interpretation, and model sets obtained by careful modelling of physical laws may also be employed.

2.1.3 Assessment Rules

A model is selected from a candidate model set based on its ability to 'reproduce' the observed output data. This is done by choosing a criterion to which the model should fit. Determining which is the best model in the candidate set is done by estimating its parameter vector Θ that minimizes a objective function $V_N(\Theta, Z^N)$. The objective function is usually one of prediction errors over a time series $t = 1 : N$ of a data setZ^N:

[1]Which can contain large segments of missing or stationary data.

$$V_N(\Theta, Z^N) = \frac{1}{N}\Sigma_{t=1}^{N} l(\varepsilon_F(t, \Theta)) \tag{2.1}$$

Figure 2.1 represents the system identification loop, in which this thesis will be focused on the entity regarding historic data selection.

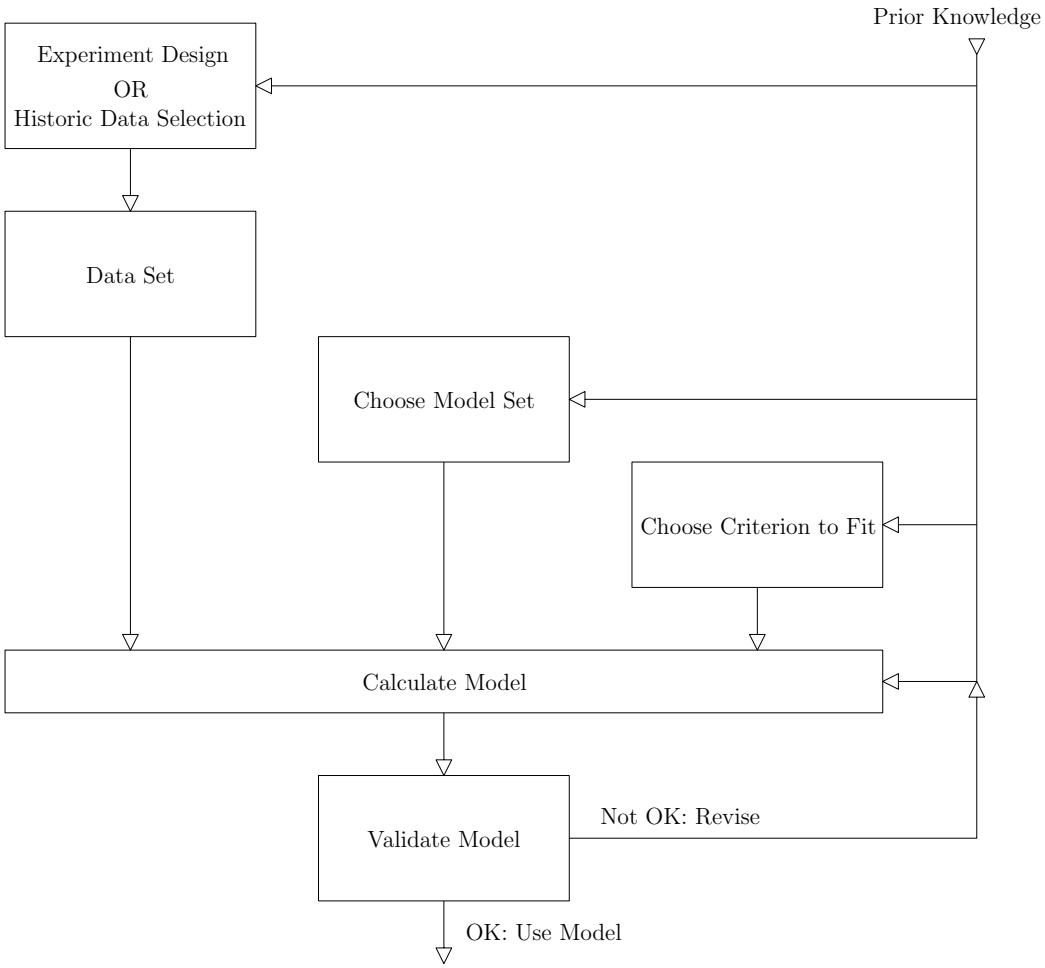

Figure 2.1: The system identification loop. Based on Ljung (1999).

Chapter 3

Concept - Design and Implementation

In this chapter the design and implementation of the concept is presented. The implementation is done in `Matlab`, and is structured as a main/framework function using a collection of other functions, which serve different purposes:

1. **Search criteria**: Functions to search the data set for interesting data segments, using different search criteria.
2. **Combinatorics**: A function to combine different search criteria.
3. **Segment selection**: A function to select which data segments to use for system identification and model validation.

This collection of functions will is presented, followed by a presentation of the framework. The `Matlab` code for functions and framework can be found in Appendix A.

3.1 Searching the Data Set

If a system is in a steady state[1], one can not determine the system's attributes. A system's, e.g. a chemical production plant, historic/operational

[1] I.e. the input and output of the system is stationary.

data can contain large segments where nothing happens, as it is at a steady state. To identify a model of the system, one must locate changes in the input and output variables, where information about the systems dynamics is revealed. In a SISO-system, changes in the input variable (u) result in a response in the output variable (y). Implementing a simple conceptual function to locate changes in u is an non-complex task. When traversing the data set, the function returns a segment of the data when it encounters an input value $u(t)$ which is unequal to $u(t-1)$.

Note:

The systems used in the examples of this text mimic open-loop experiments, as it is only the response of the system which is of interest. With no feedback to a controller, the input data is not dependent on the output data, and a set point variable is not included. Input data is chosen by the author and the output data is generated by Matlab. Changes in the input data occurs as steps, but as this data is discrete and the plots are continuous, steps appear as ramps between samples in these plots.

Example 1

In this example a first order Linear Time-Invariant (LTI) SISO-system will be considered. A gain and a time constant determine the response from the system output to changes in the input. No time delay or noise will be considered. The input variable, u is constant until a step occurs at time step 50. Output variable, y, is generated by Equation 3.1. The data set is shown in Figure 3.1.

$$\frac{y}{u}(s) = \frac{k}{T_1 s + 1} \qquad k = 4,\ T_1 = 3 \tag{3.1}$$

When a change in u is perceived, the function returns a segment containing the change, as seen in Figure 3.2. This functionality is not of any practical use, as any change of u, e.g. measurement noise(not considered in this example), alerts the user of informative data, so other criteria must be explored.

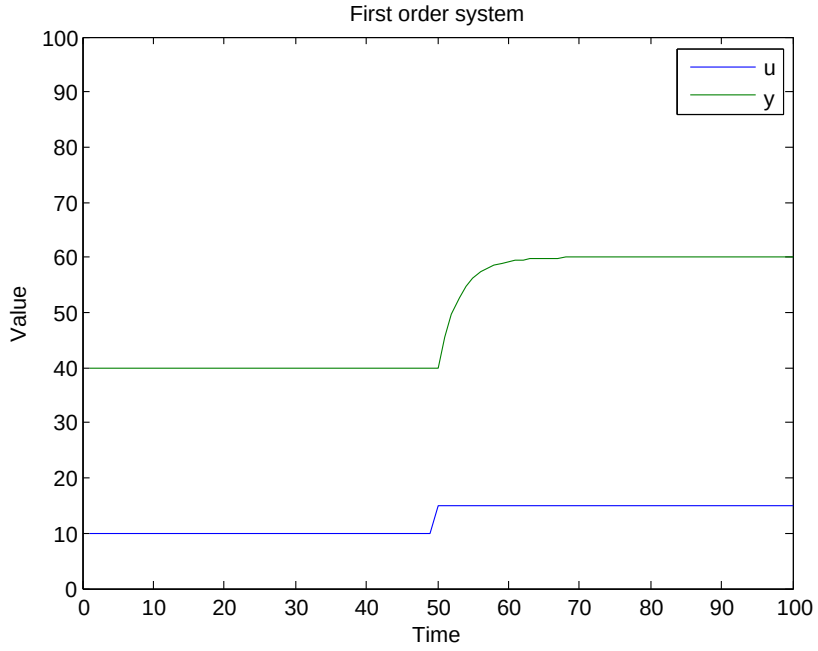

Figure 3.1: Example 1: First order LTI SISO-system.

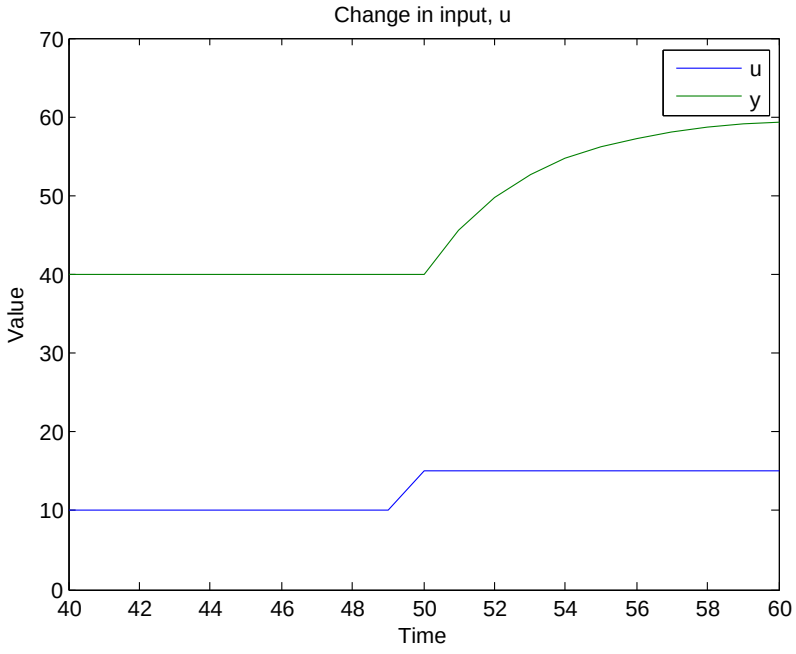

Figure 3.2: Data segment.

3.1.1 Search Criterion Functions

Using different criteria, it is desirable to search a data set for informative/interesting data. In the concept this will be done by 'criterion functions'. These have a criterion parameter as an input argument, which decide which segments of the data set is returned as interesting from the functions. The returned data comes in the form of a vector, which is the same size as the original data set. This vector is initially filled with zeros, and when the criterion of the function is met, a segment of a desired size is filled with ones. This indicates an 'interesting' segment. By filling the vector in this manner, overlapping segments of interesting data become one larger segment.

3.1.2 Finding Steps in the Input Variable

According to Ljung (1999), there are three basic facts that govern the choices one makes designing an open loop experiment for the identification of linear systems:

1. The asymptotic properties of the estimate (bias and variance) depends only on the *input spectrum* - not the actual waveform of the input.
2. The input must have limited amplitude: $u_{lower} \leq u(t) \leq u_{upper}$.
3. Periodic inputs may have certain advantages.

As common input signals for such experiments, filtered gaussian white noise and random binary signals are mentioned. These are series of steps in the input signal, so searching for steps in the input data of historic data sets seems like the obvious place to start. Instances where u changes in steps may be seldom or non existing in real life scenarios. Although the set point of a controller may behave in this manner, the examples of this text, as mentioned, treat the systems as open loop experiments, and u is preferred to the set point as a system variable.

inputStep.m

The `Matlab` function *inputStep.m*, returns segments of a given data set in the manner described in Section 3.1.1. As input arguments the function needs a window size which determines the size of the returned data segments to be considered interesting, the input data of the set and a 'minimum step size' parameter[2]. The last argument gives the user a choice to filter out small and insignificant changes between samples in the input data. It seems to demand little from the user to set these arguments (window size and step size), which is in line with the concepts attempt to be easy to use, but some knowledge of the given system/plant is of course preferable. The criterion of the function is shown in pseudo code in equation 3.2, and considers both positive and negative steps in the data.

$$\mathsf{if}(u(i) \geq (u(i-1) + stepSize) \;\; \mathsf{OR} \;\; u(i) \leq (u(i-1) - stepSize)) \quad (3.2)$$

Example 2

Consider the same LTI system as in Section 3.1 described in Equation 3.1. A small step is set to occur at $u(5)$ and $u(10)$. A greater step occurs at $u(30)$ and $u(71)$. Figure 3.3 shows the simulated system.

Running the function *inputStep.m* on this data set, with a window size of 20 samples and a minimum step size of 5, returns a vector with the segments shown in Figure 3.4. The small steps in the beginning of the data set is not considered, but two independent segments meet the criterion. Note that these segments are one sample larger than the chosen window size, as *inputStep.m* considers one sample at the time and creates a segment around this sample, i.e. $windowSize/2 + activeSample + windowSize/2$. This complies to all the 'criterion functions'.

The segments suggested by *inputStep.m* capture steps in u well enough, but Figure 3.4 illustrates that these segments may be too small as the response of y does not seem to reach steady state. Sufficient information re-

[2]Difference quotient between subsequent samples.

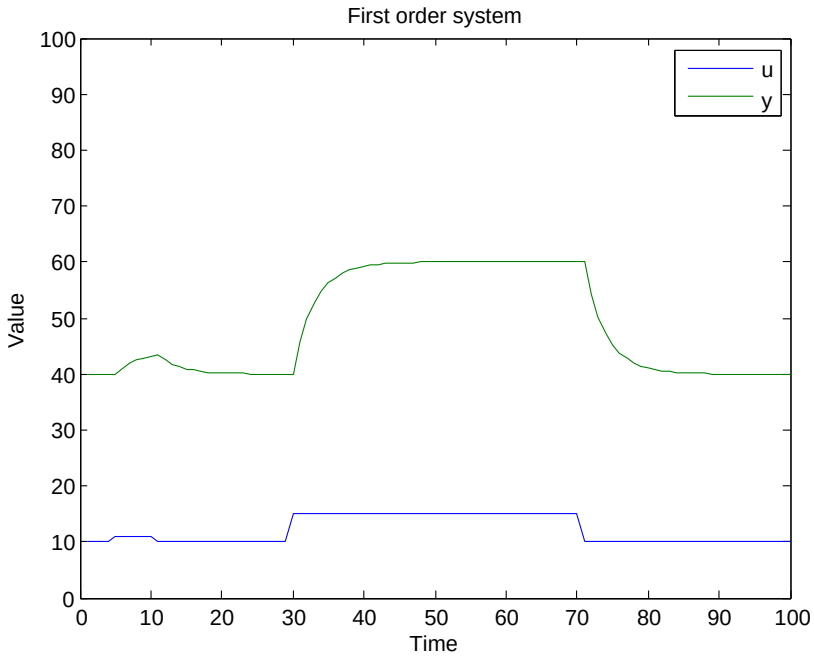

Figure 3.3: Example 2: First order LTI SISO-system.

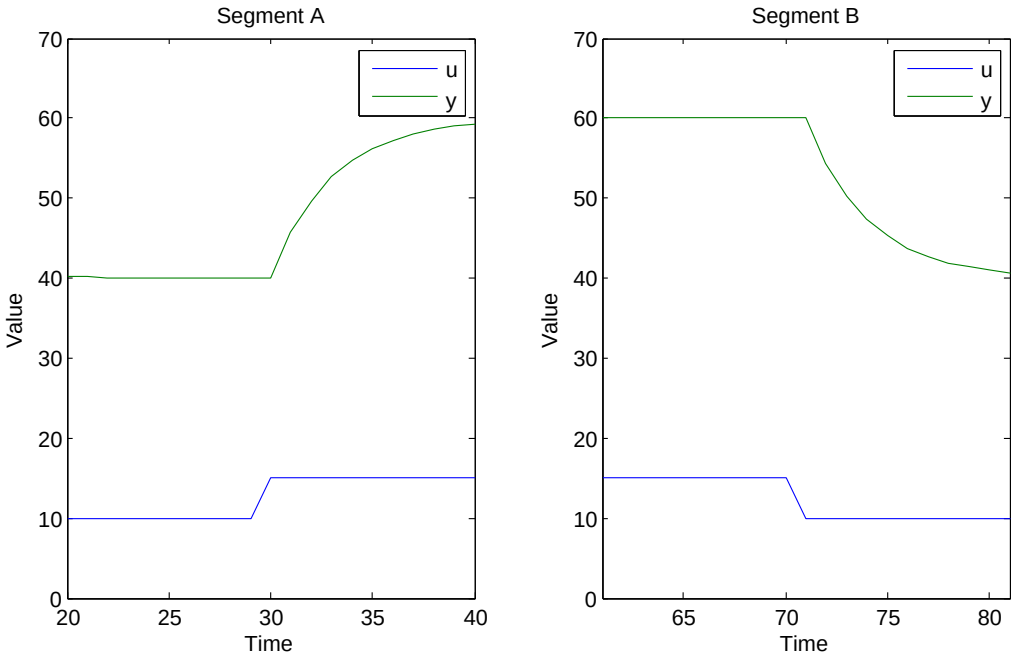

Figure 3.4: Data segments considered interesting by *inputStep.m*.

garding the gain and time constant of the system might not be present. A solution to such a problem can be to increase the window size.

In this simplified example, no noise/disturbances were considered, and it is obvious that this would create problems for *inputStep*. This will be discussed in Chapter 5.

3.1.3 Finding Changes in the Mean Value of a Data Set

Brown & Hwang (1997) defines the mean value of a variable X as:

$$\mathsf{Mean}(X) = \frac{X_1 + X_2 + \cdots + X_N}{N} \tag{3.3}$$

This implies that if a variable, X, is constant(steady state) over a period of time, N, its mean value will be constant as well[3]. Traversing a data sequence with a sliding window, in which one compute the mean value of that data sequence for each window, changes of the mean value of these windows indicate significant change in the sequence, as small and/or random disturbances, such as noise, are 'cancelled' out. This is shown in Figure 3.5, where a variable y is affected by a random noise.

This line of thought is inspired by Blanke, Kinnaert, Lunze & Staroswiecki (2006), where the Cumulative Sum Algorithm(CUSUM) is proposed to detect change in the mean of normally distributed random sequences for fault detection, isolation and/or estimation in various systems.

meanChange.m

The function *meanChange.m* uses a sliding window to calculate the mean values of a data sequence as described above. Input arguments of the function is a sliding-window size, a data sequence, and a criterion parameter 'minimum mean change'. The criterion is the same as the one used in *inputstep.m*, but

[3]And the same value as X.

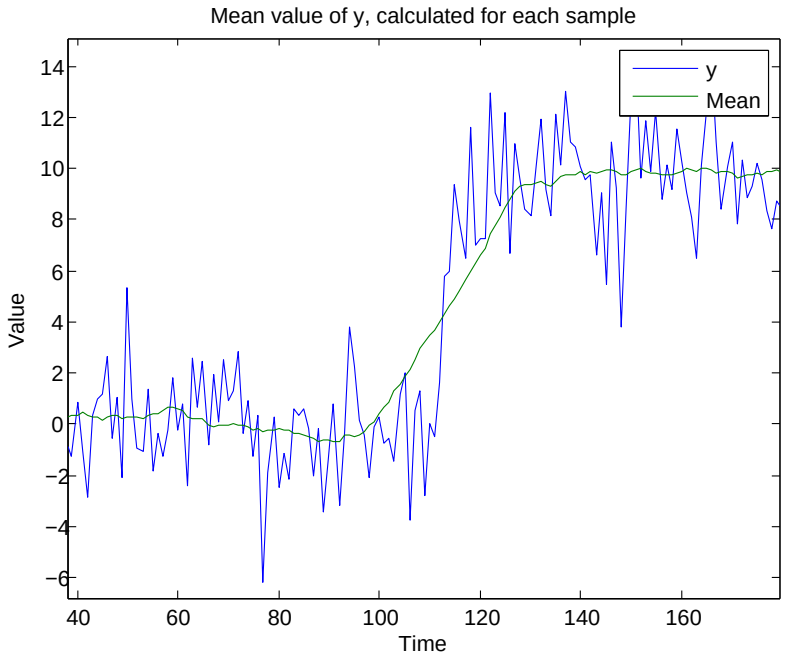

Figure 3.5: Mean value of variable y, calculated at each sample.

regards changes in the calculated means, and not the data sequence itself (Equation 3.4).

$$\mathsf{if}(meanChange \geq minMeanChange \ \mathsf{OR} \ meanChange \leq minMeanChange) \tag{3.4}$$

When the criterion is met, a segment the size of the sliding window is considered an interesting segment in the functions returned vector.

Example 3

Consider the second order LTI SISO-system given in Equation 3.5, where Θ is the time delay of the system. The output data, y, is affected by a relatively small random noise variable with an amplitude of 0.2. The simulated system is shown in Figure 3.6.

$$\frac{y}{u}(s) = \frac{k}{(T_1 s + 1)(T_2 s + 1)} e^{-\Theta s} \qquad k = 2,\ T_1 = 2,\ T_2 = 4, \Theta = 10 \quad (3.5)$$

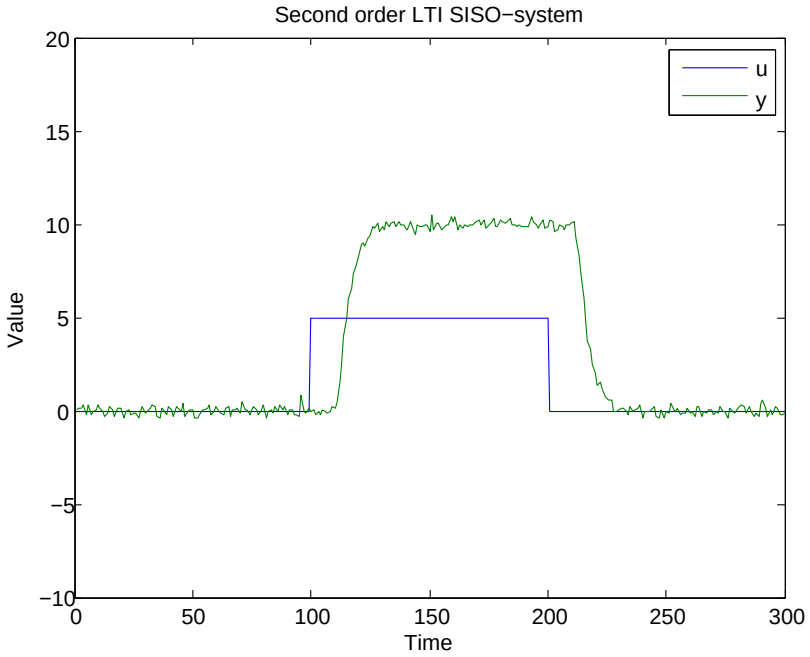

Figure 3.6: Example 3: Second order LTI SISO-system

Running *meanChange.m* with y as data sequence, a window size of 30 samples and 0.2 as a minimum difference quotient between subsequent mean values(criterion argument), returns the segments in Figure 3.7 as interesting. Both the steps in u and the response of y are captured in the segments. A long enough time delay, Θ, would ultimately result in returned segments which do not contain the steps in u(without increasing the window size). More on this later.

Figure 3.8 shows the calculated means of y per window. Note that the vector containing these means is one window size smaller that the data set, as calculations begin and end half a window size into the set. The remaining sample points are set to zero due plotting reasons. Setting the window size affects the calculation of the mean values. A large window results in low sensitivity to disturbances/noise and a low difference quotient at step responses,

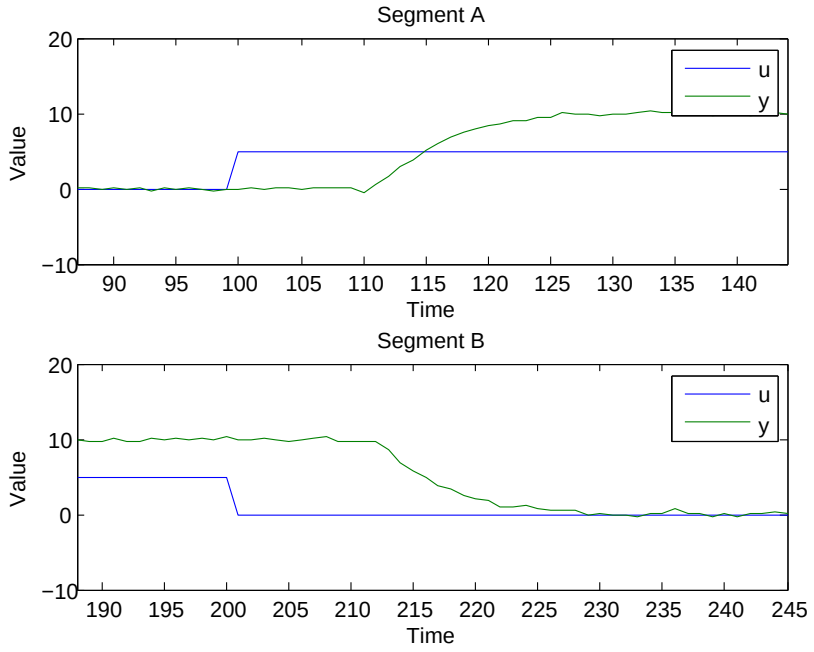

Figure 3.7: Data segments considered interesting by *meanChange.m*.

while a small window have a higher sensitivity to noise but also a higher and more accurate difference quotient at step responses. More on this in Chapter 5.

3.1.4 Finding Variance in the Data Set

The function *meanChange.m* compares changes in the mean value of subsequent windows in a sliding window scheme. If one wants to find diversification of a data sequence within each of those windows, *variance* of the sequence can be utilized.

Brown & Hwang (1997) defines the variance of a variable X as:

$$\mathsf{Var}\ X = \mathsf{Mean}(X^2) - (\mathsf{Mean}(X))^2 \tag{3.6}$$

Where:

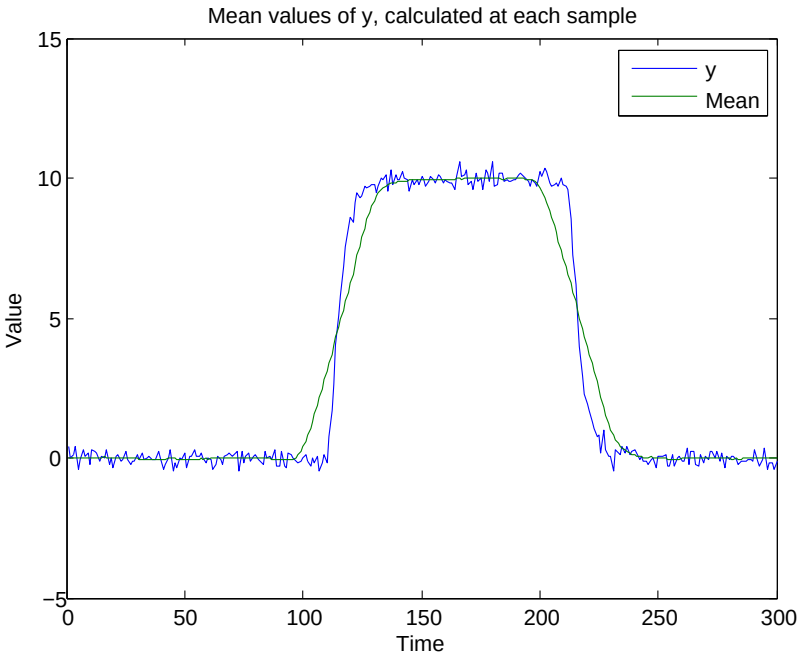

Figure 3.8: Mean values of y calculated by *meanChange.m*.

$$\mathsf{Mean}(X) = \frac{X_1 + X_2 + \cdots + X_N}{N} \tag{3.7}$$

$$\mathsf{Mean}(X^2) = \frac{X_1^2 + X_2^2 + \cdots + X_N^2}{N} \tag{3.8}$$

In segments of the data set where the system is at a steady state (and without any noise), the variance of a variable will be zero. Implementing a variance criterion should make a user able to locate data segments where the system is not at a steady state, as the variance of the variable will be non zero and positive.

variance.m

The `Matlab` function *variance.m* has the following input arguments: a data sequence, a sliding window size, and a 'minimum variance' parameter. Traversing the data sequence is done by using a sliding window. At each time sample the variance of the sequence is calculated over this window using Equation

3.6. If this window/segment triggers the functions criterion, which is represented as pseudo code in Equation 3.9, it is considered interesting in the returned vector (filled with ones). Most likely, the variance of the next window also triggers the criterion, and does so until the data sequence is at a steady state. This results in a segment returned containing the transient of the response, i.e. what one wants. From Equation 3.6 one can see that the unit of variance is the original variables unit squared. With this in mind, setting the minimum variance criterion may not be straight forward. More on this in Chapter 5.

$$\mathsf{if}(variance \geq minimumVariance) \tag{3.9}$$

Example 4

Consider the second order LTI-system in Example 3 in Section 3.1.3, presented in Equation 3.5 and Figure 3.6. Running *variance.m* on y with a window size of 30 samples and 10 as a minimum variance criterion, results in the calculated variances presented in Figure 3.9. At the low noise level of this example the variance gives a clear indication of where the step responses are in the system(discussed in Chapter 5), and with the minimum variance criterion set to 10, the segments in Figure 3.10 are returned as interesting.

3.1.5 Variance of the Mean Value

Figure 3.11 illustrates that variance, calculated by *variance.m*, of a variable y affected by considerable noise, become less reliable as a criterion than indicated in Example 4. Although the variance 'peaks' at the step response of y, it has a considerable bias off zero when the variable is constant.

Recall the properties of the mean values calculated by *meanChange.m* regarding noise. Calculating the variance of these in the same manner as *variance.m*, results in the variance of mean values shown in Figure 3.12.

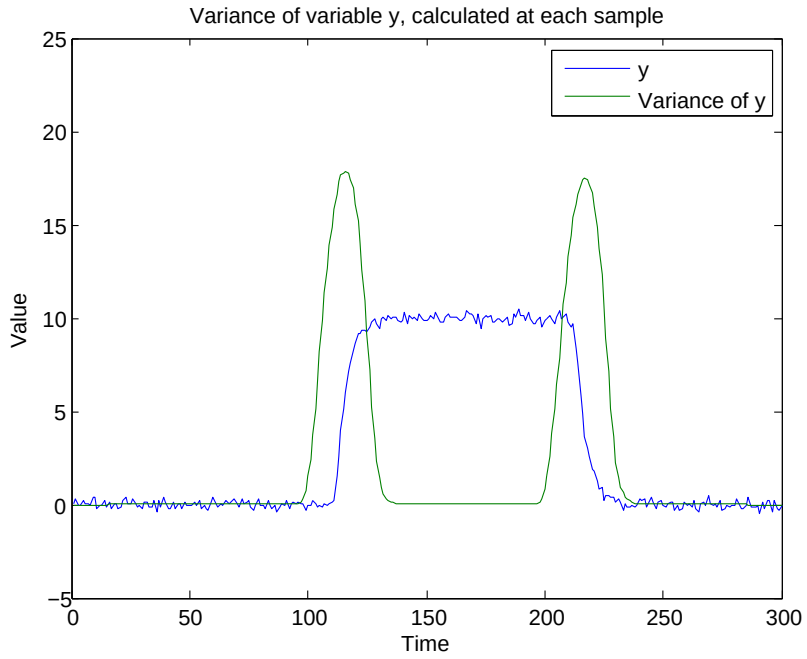

Figure 3.9: Variance of y calculated by *variance.m*.

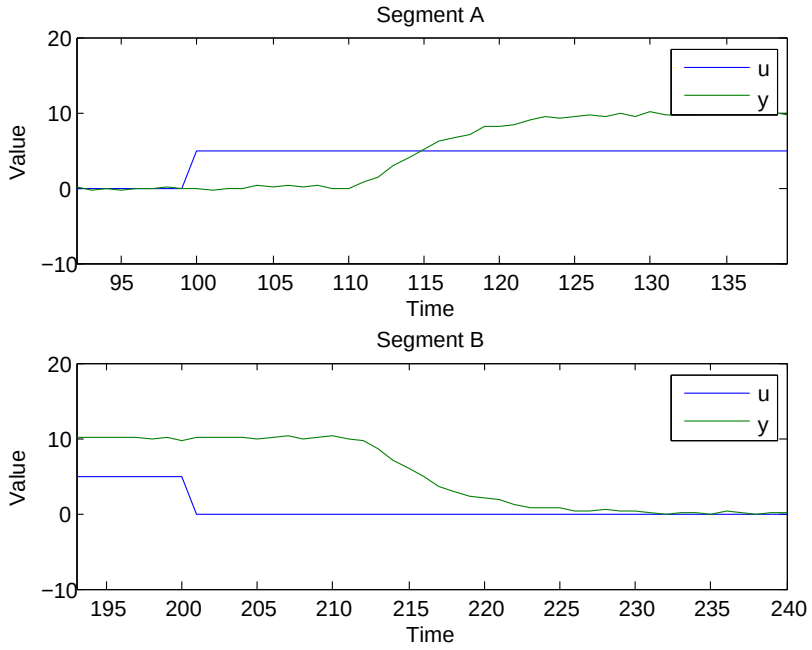

Figure 3.10: Data segments considered interesting by *variance.m*.

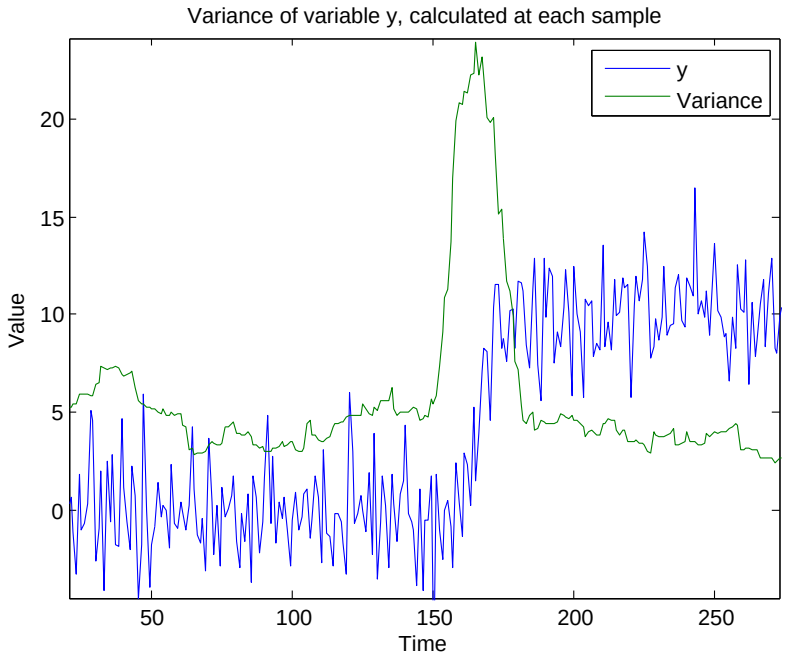

Figure 3.11: Variance of y calculated by *variance.m*.

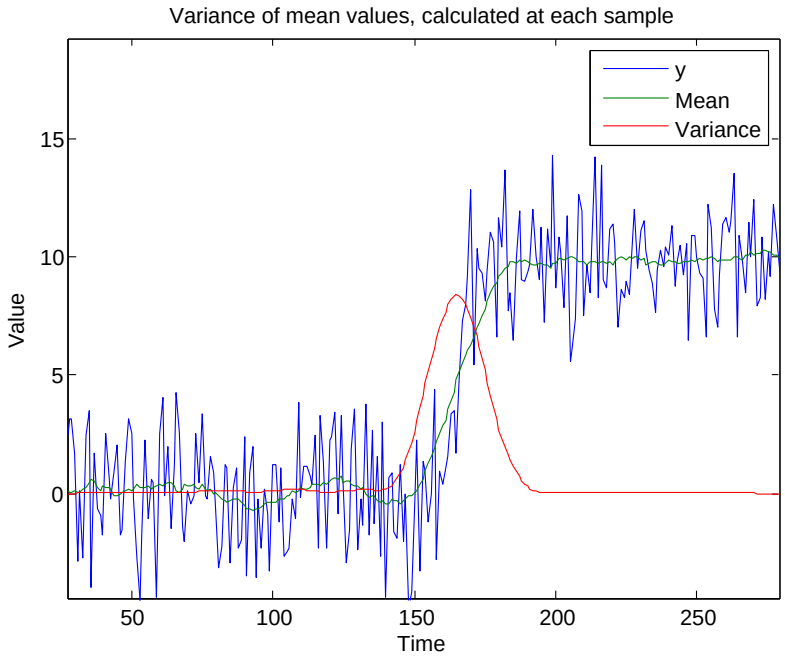

Figure 3.12: Variance of mean values.

meanVariance.m

meanVariance.m calculate the mean values of a data sequence in the manner suggested in Section 3.1.3. It then calculates the variance of these in the manner of *variance.m*, and the function uses the same sliding window size for these operations. This is done for convenience, although use of different window sizes could well be implemented. The criterion and its parameter is the same as in Equation 3.9, but regards the variance of mean values of a variable, and not the variable itself. When the criterion is met, the active window is considered interesting.

Example 5

Consider the second order LTI SISO-system given by Equation 3.10, with a noise affecting y, presented in Figure 3.13.

$$\frac{y}{u}(s) = \frac{k}{(T_1 s + 1)(T_2 s + 1)} e^{-\Theta s} \qquad k = 2,\ T_1 = 1,\ T_2 = 2, \Theta = 5 \quad (3.10)$$

The mean values of y and their variance calculated by *meanVariance.m*, with a sliding window size set to 30 and a minimum variance (criterion parameter) set to 1, are presented in Figure 3.14. Variance of the variable seems to give clear and unambiguous indications to where the step responses of y occur. In turn the criterion parameter can be set low, even with considerable noise compared to the size of some of the step responses. This will be discussed further in Chapter 5. The segments returned by *meanVariance.m* are shown in Figure 3.15, where Segment B is one segment made up by two overlapping segments considered interesting.

3.1.6 Combining Criteria

For a data segment suggested by the criterion functions to be of any use in system/model identification, it is imperative that it contains both the

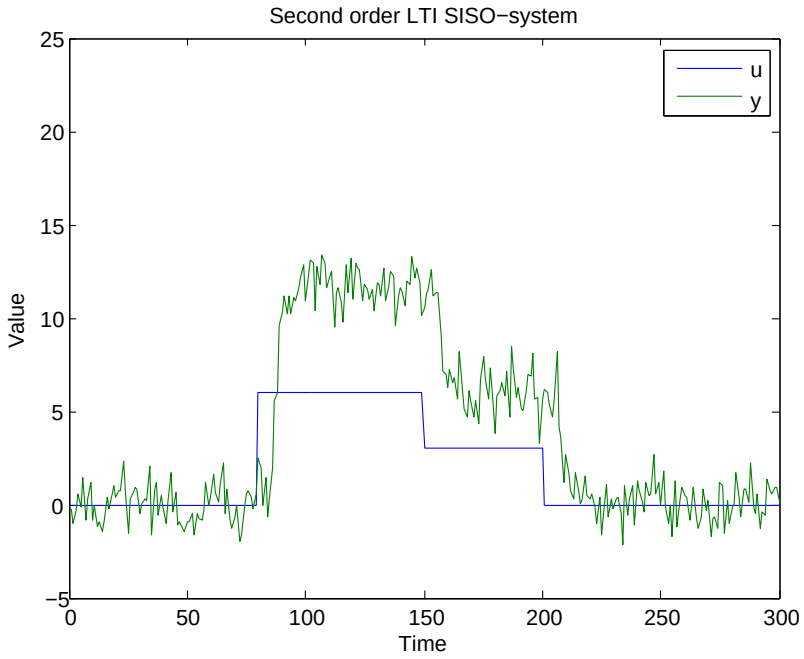

Figure 3.13: Example 5: Second order LTI SISO-system

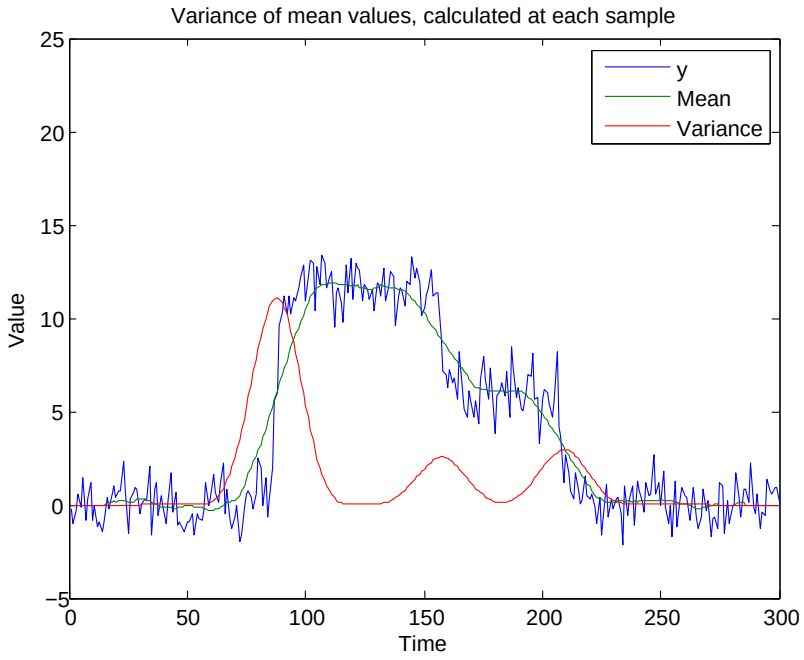

Figure 3.14: Variance of mean values of y calculated by $meanVariance.m$.

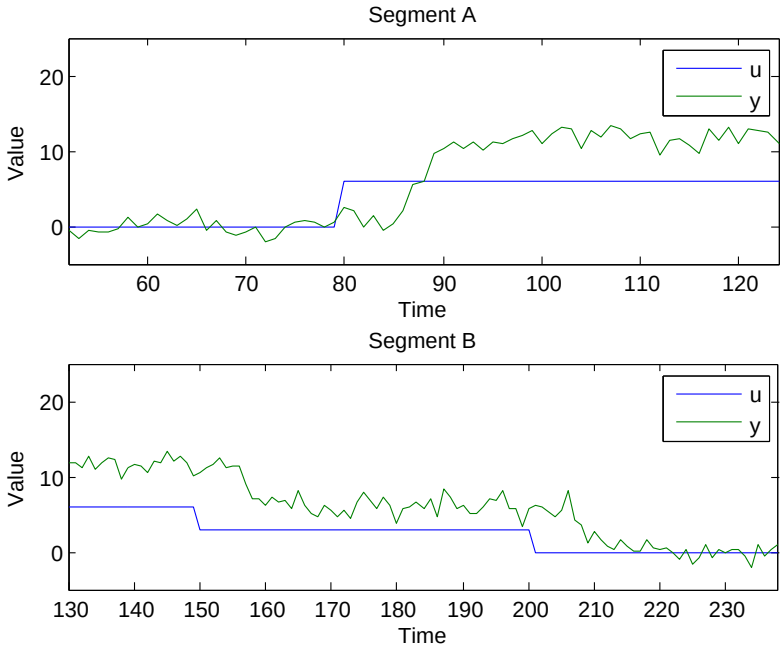

Figure 3.15: Data segments considered interesting by *meanVariance.m*.

steps/changes in the input data and the output data response to these. To ensure this the user is given the option to apply and combine several search criteria to the data set. By combining a search for steps/changes in input data with a search for response transients in output data, returned data segments will contain both.

criteriaCombination.m

The `Matlab` function *criteriaCombination.m* compares two data vectors returned from criterion functions, supplied as input arguments. Recall that these data vectors have the same amount of samples as the original data set, and differentiate between interesting and not interesting data samples in a binary fashion(zero for not interesting and one for interesting). When segments of interesting data in the compared vectors overlap, they are merged and returned as one segment in the returned data vector of *criteriaCombination.m*. Running *criteriaCombination.m* once, one can combine two search criteria. Running it again with the combined result from the first run as an input argument, i. e. comparing it with the result of another criterion function,

one can, in an additive manner, combine several search criteria/data vectors. More on how this is implemented in the main/framework function in Section 3.3.

Example 6

Consider the second order LTI SISO-system presented in Equation 3.11 and Figure 3.16, where y is affected by a random noise.

$$\frac{y}{u}(s) = \frac{k}{(T_1 s + 1)(T_2 s + 1)} e^{-\Theta s} \qquad k = 2,\ T_1 = \frac{1}{2},\ T_2 = 1, \Theta = 40 \quad (3.11)$$

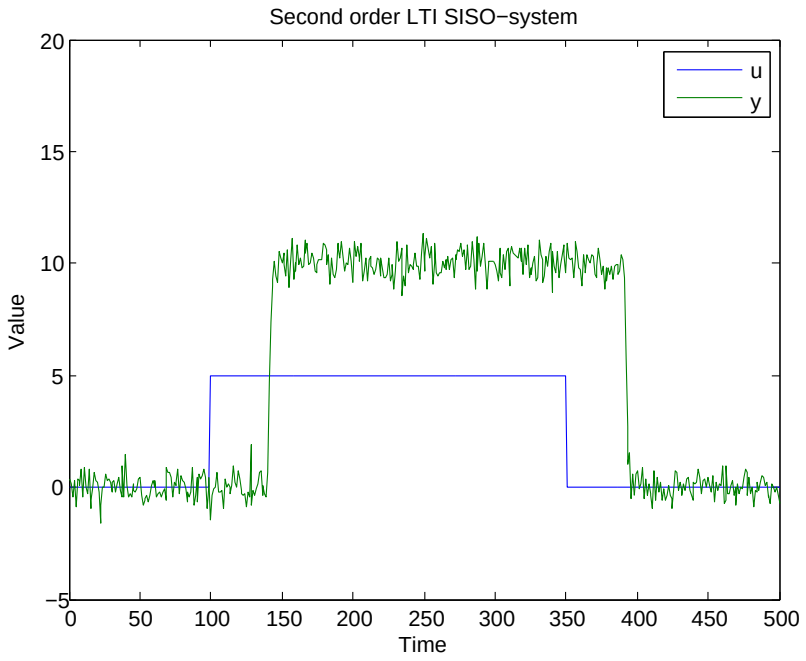

Figure 3.16: Example 6: Second order LTI SISO-system.

Searching for variance in u with *variance.m*, with a window size of 30 and a minimum variance criterion of 1, results in the segments in Figure 3.17 considered as interesting. Searching for variance of the mean values in y with *meanVariance.m*, with the same window size and criterion value(30,

1), results in the segments in Figure 3.18. Due to the time delay and selected window sizes none of these segments contain both the steps in u and responses of y. Using the returned data vectors from *varaince.m* and *meanVariance.m* as inputs to *criterionCombination.m*, overlapping segments are combined, and a new data vector suggesting the segments in Figure 3.19 as interesting, are obtained.

It is obvious that the selection of window sizes affect *criterionCombination.m*'s ability to combine search criteria. If a time delay is long enough, segments returned from different criterion functions will not overlap. This is discussed in Chapter 5.

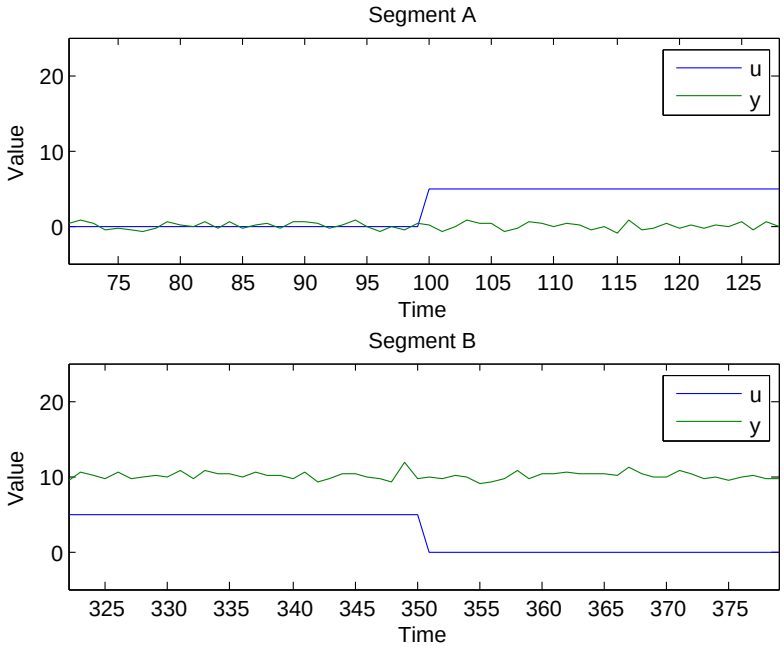

Figure 3.17: Data segments suggested by *variance.m* regarding u.

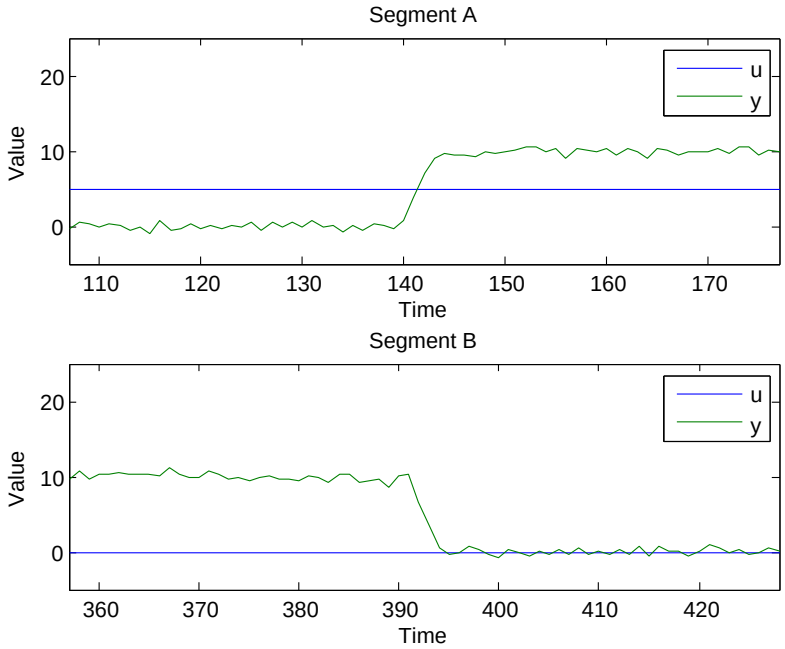

Figure 3.18: Data segments suggested by *meanVariance.m* regarding y.

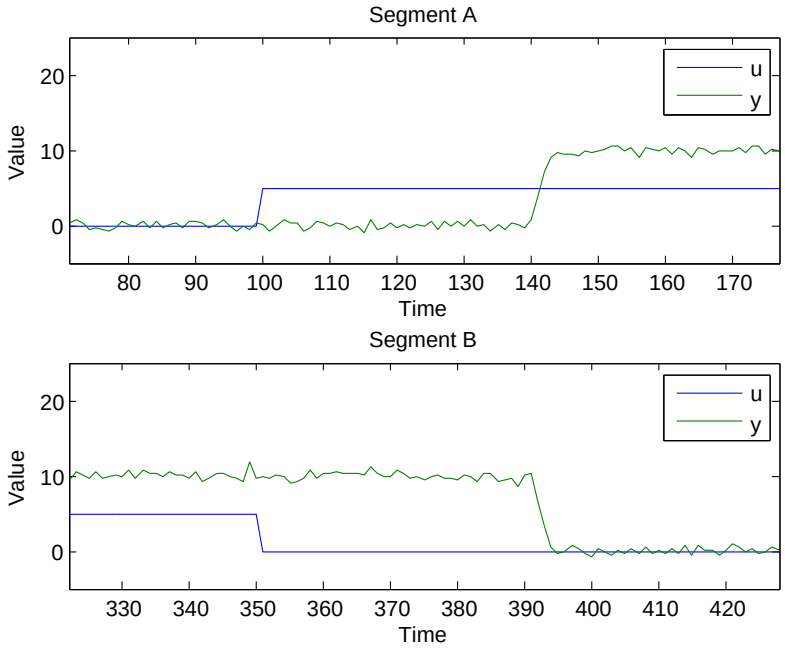

Figure 3.19: Data segments suggested by *criterionCombination.m*.

3.2 Choosing Segments from the Data Set

The last step in the system identification loop(Figure 2.1) is to validate the model. To do this it is recommended to use a different data set than the one used to identify/build the model. If a data set lacking some information about the system is used to build and evaluate a model, it could appear to be 'correct', even if its not. By evaluating/comparing the model to a different data set, one can determine if the 'whole' system modelled is emulated in a satisfactory way.

Selecting which segments to use to identify/build and validate a model is the last step in the concept. Searching a data set as described in previous sections can result in multiple segments of interesting/informative data. Allowing a user to manually choose which segments to use would be advantageous, but in this prototype of the concept they are selected by a function *segmentSelect.m*.

segmentSelect.m

The `Matlab` function *segmentSelect.m* traverse a data vector provided by a criterion function (or *criteriaCombination.m*), supplied as an input argument, and returns four vectors. The first two are to be used in model identification(identification data set), one containing a segment of input data and the other a segment of output data from the original data set(supplied as input arguments). The last two are to be used in model validation(validation data set), these also separated as input and output data. No criterion regarding which segment should be what is implemented in this prototype of the concept. As a simple mean to distinguish segments, the first segment encountered in the current data vector supplied to *segmentSelect.m* is returned as the two identification vectors, and the next (if there is one), is returned as the two validation vectors. E. g. using the data vector returned by *criteriaCombination.m* in Example 6 as input to *segmentSelect.m*, Segment A in Figure 3.19 would be returned as identification data and Segment B as validation data. If there indeed only is one segment considered interesting by the search criteria, the identification and validation data is set to be equal. If no data segments are considered interesting by the foregoing search,

the identification and validation sets are returned empty.

3.3 Framework

To utilize the criterion, combinatoric and selection functions presented in Section 3.1 and 3.2 as a concept/method for searching data sets, a framework is needed. This will be implemented in a 'main' function called Searching Historic Data for Model Identification(SHDMI), which the concept will be referred to as from now on.

shdmi.m

SHDMI as a tool for searching data sets, is presented to the user as the `Matlab` function *shdmi.m*. When executing the function/method the user needs to:

1. Supply input and output data of a system.
2. Select which criteria to use.
3. Set criterion parameter values.
4. Set a window size.

This is done by setting the input arguments of the function, which are as follows:

1. u: Input data of a data set.
2. y: Output data of a data set.
3. $windowSize$: Size of window used to fill in return vectors from criterion functions and in sliding window schemes.
4. $stepSize$: Criterion parameter used in *inputStep.m*.
5. $minMeanChangeU$: Criterion parameter used in $meanChange.m$ regarding input data.

6. $minMeanChangeY$: Criterion parameter used in *meanChange.m* regarding output data.
7. $minVarianceU$: Criterion parameter used in *variance.m* regarding input data.
8. $minVarianceY$: Criterion parameter used in *variance.m* regarding output data.
9. $minMeanVarU$: Criterion parameter used in $meanVariance.m$ regarding input data.
10. $minMeanVarY$: Criterion parameter used in $meanVariance.m$ regarding output data.

The window size set in *shdmi.m* is used by all selected criteria functions. Individual window sizes for each criterion parameter could be implemented and indeed advantageous, but the single one used commonly in this prototype of SHDMI is convenient as the list of input arguments is considerably long. Recall that the functions in Section 3.1 use half the window size when filling their returned data vectors. This implies that the window size must be an even number, as samples of the data set are integers.

Selecting which search criteria to use and setting their criterion parameters is done in one action. E. g. setting $minVarianceU$ to 5, the variance search criteria is selected for u and its criterion parameter is set to 5. By setting a criterion parameter to zero, the respective search is not run. E. g. by setting $minVarianceU$ to 5, $minMeanVarianceY$ to 1 and the remaining criterion parameters to zero, the only criterion functions run and combined are *variance.m* regarding u, and $meanVariance.m$ regarding y.

criterionCombination.m is run after each of the criterion functions. The result of each combination is carried on in *criterionCombination.m*'s returned data vector, and this is then combined with the result of the next criterion function selected in the sequence to combine all the selected criteria. The first criterion function run is combined with itself, as there is no result from a previous combination.

The last step of SHDMI is *segmentSelect.m*. Using the data vector supplied by the last instance of *criterionCombination.m*, containing the results

of all criteria combined, *segmentSelect.m* distinguishes two sets of input and output data to be used in system identification. These are returned from *shdmi.m* as a system identification set and a validation set, and the search is complete. If no segments are returned, it indicates that the criterion parameters must be lowered and/or the window size is to small, or that the original data set indeed does not contain any informative/interesting data.

Example 7

Consider a scenario where *variance.m* regarding u, and $meanVariance.m$ regarding y are selected as search criteria. The sequence of *shdmi.m* is as follows (pseudo code):

1. *stepSize.m* - not executed, not combined.
2. *meanChange.m* regarding u - not executed, not combined.
3. *meanChange.m* regarding y - not executed, not combined.
4. *variance.m* regarding u - executed, combined with itself. Result of combination stored in a data vector, *result*.
5. *variance.m* regarding y - not executed, not combined.
6. $meanVariance.m$ regarding u - not executed, not combined.
7. $meanVariance.m$ regarding y - executed, combined with *result*. Result stored in *result*.
8. *segmentSelect.m* - distinguishes identification and validation data sets, which are returned from *shdmi.m*.

---

k_m	1.9918	+ - 0.0087
T_{m1}	0.9407	+ - 0.3743
T_{m2}	3.1197	+ - 0.2440
Θ_m	5.0190	+ - 0.2254

Table 3.1: Model parameters calculated by the System Identification Toolbox.

3.4 System Identification

The objective of SHDMI is to search historic data and obtain identification and validation data sets to proceed in the system identification loop (Figure 2.1). Although it is not the focus of this thesis, an example of supplying a conventional system identification method with a identification and validation data set, will be given in this section by using the System Identification Toolbox in Matlab(Mathworks 2010).

Example 8

Consider the second order LTI SISO-system presented in Equation 3.12 and Figure 3.20, where y is affected by a random noise.

$$\frac{y}{u}(s) = \frac{k}{(T_1 s + 1)(T_2 s + 1)} e^{-\Theta s} \qquad k = 2,\ T_1 = \frac{1}{2},\ T_2 = 1, \Theta = 40 \quad (3.12)$$

Employing SHDMI to search for variance in u and variance of the mean values of y, with a window size of 30, results in the data sets in Figure 3.21. Using these data sets as identification and validation data in the System Identification Toolbox, and calculating a second order LTI SISO-model, gives the model parameters in Table 3.1. These are close to replicating the system parameters, and a comparison of the model output and y is shown in Figure 3.22 (validation data set).

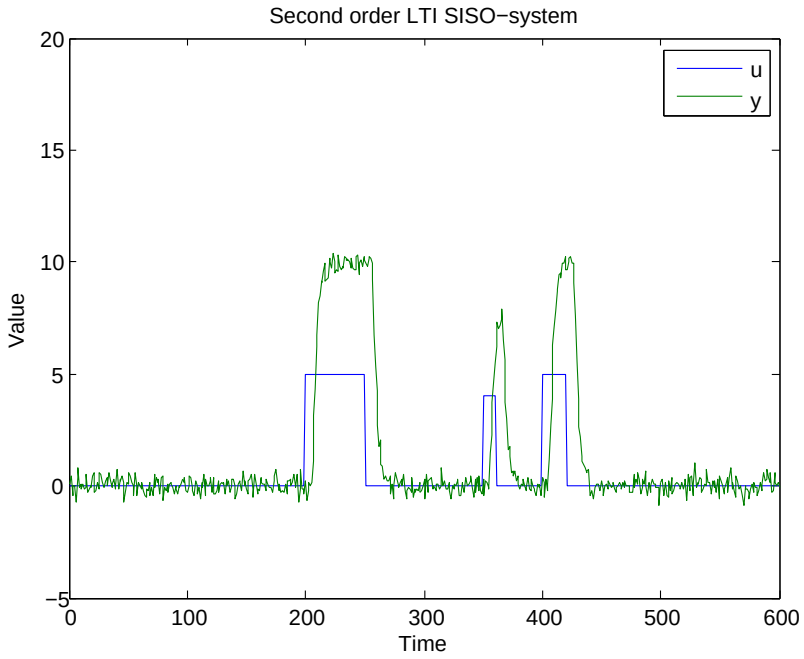

Figure 3.20: Example 8: Second order LTI SISO-system.

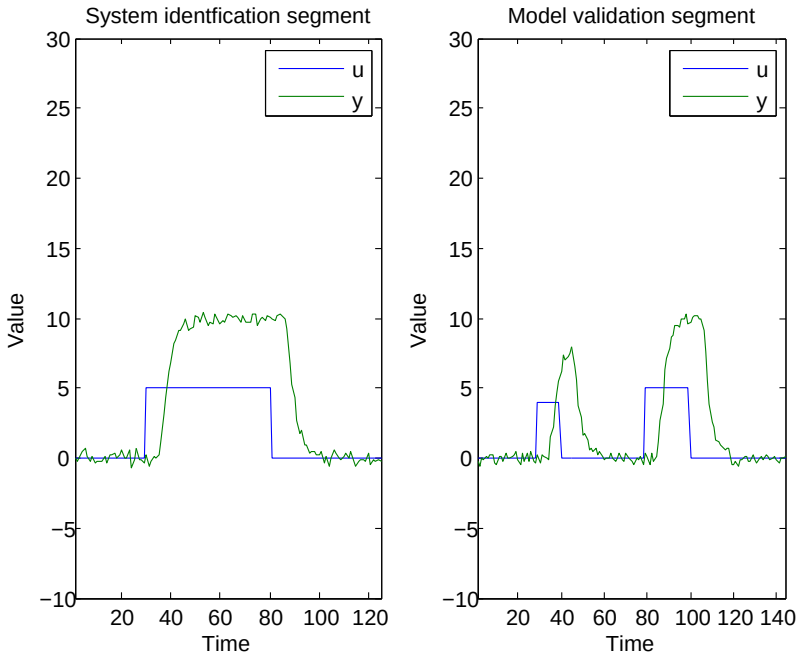

Figure 3.21: Data segments suggested by SHDMI.

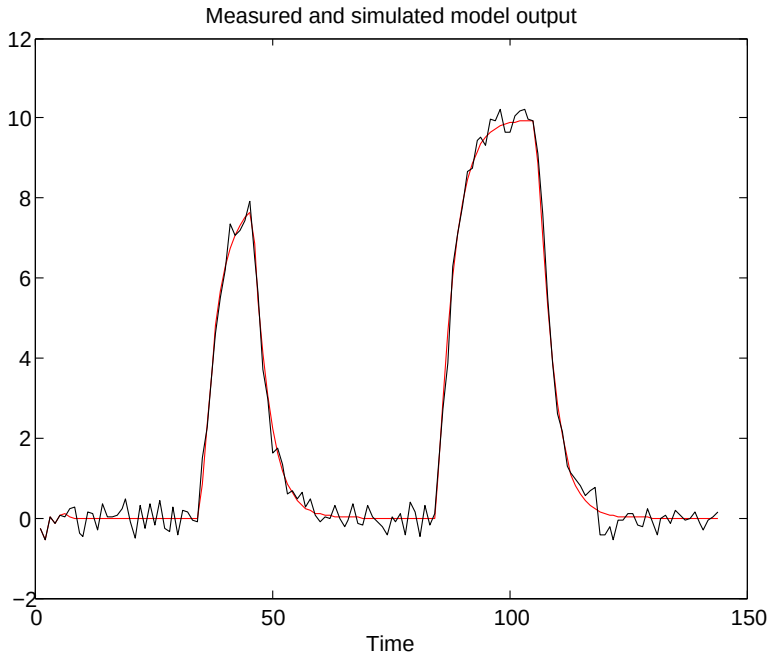

Figure 3.22: Comparison of system and model behaviour.

Chapter 4

Expanding SHDMI to use with MIMO-systems

SHDMI is implemented to deal with SISO-systems. In MIMO-system the behaviour of paired multiple input and/or output variables of a system must be present in a data set for a model identification to be successful. In the manner SHDMI is implemented, an expansion of system variable data arguments, and combining searches for each of these, could ensure that the systems behaviour is present in resulting data segments. As an example of this a version of SHDMI, which considers two input variables affecting one output variable of a system, will be implemented and presented in this chapter. The implemented `Matlab` code can be found in Appendix A

4.1 SHDMI - MISO case

shdmiMISO.m is a version of SHDMI which consider two-input one-output systems (MISO). It is similar to *shdmi.m*, but has two input system variables in the argument list, and returns identification and validation data sets containing three system variables[1]. In this example only *variance.m* and *meanVariance.m* will be used to search the systemss three variable data sequences, so the other criterion functions are left out. This results in a input

[1] A version of *segmentSelect.m*, *segmentSelectMISO.m*, includes both inputs and the output of the system in the data sets.

arguments list as follows:

1. $u1$: Input data 1 of a data set.
2. $u2$: Input data 2 of a data set.
3. y: Output data of a data set.
4. $windowSize$: Size of window used to fill in return vectors from criterion functions and in sliding window schemes.
5. $minVarianceU1$: Criterion parameter used in $variance.m$ regarding $u1$.
6. $minVarianceU2$: Criterion parameter used in $meanChange.m$ regarding $u2$.
7. $minVarianceY$: Criterion parameter used in $meanChange.m$ regarding y.
8. $minMeanVarU1$: Criterion parameter used in $meanVariance.m$ regarding $u1$.
9. $minMeanVarU2$: Criterion parameter used in $meanVariance.m$ regarding $u2$.
10. $minMeanVarY$: Criterion parameter used in $meanVariance.m$ regarding y.

Example 9

Consider the first order LTI MISO-system given in Equation 4.1, where u_1 and u_2 affect y with the same time constant, and opposite gain. A random noise have been added to y to mimic measurement noise. The simulated system is shown in Figure 4.1. To represent y being affected by a step in only one of the input variables, steps occur in u_1 and u_2 with some time between them. Later in the data set, a step in both input variables occur closer in time.

$$y(s) = \frac{k}{T_1 s + 1} e^{-\Theta s}(u_1(s) - u_2(s)) \qquad k = 4,\ T_1 = 3,\ \Theta = 5 \tag{4.1}$$

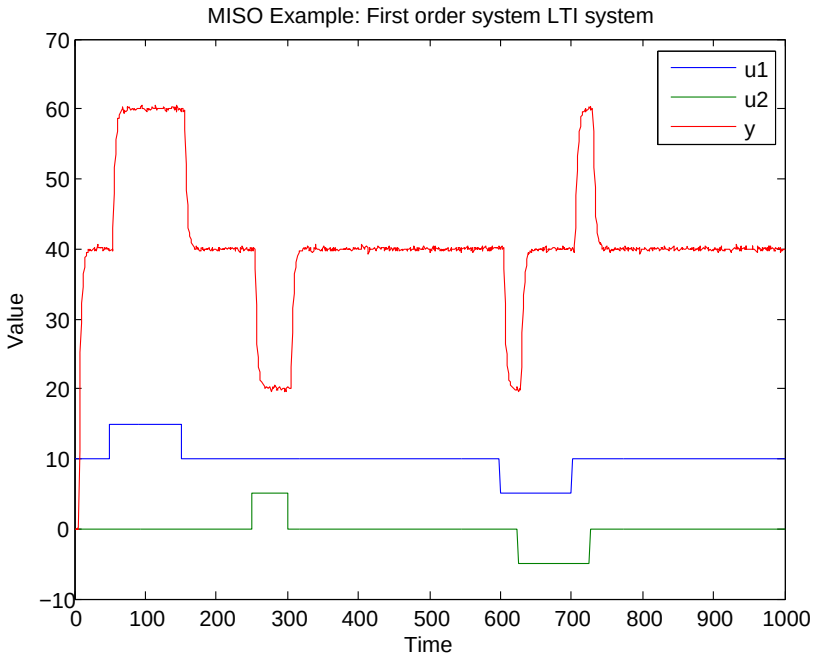

Figure 4.1: Example 9: First order LTI MISO-system.

Running *shdmiMISO.m* with a window size of 30, a criterion parameter for *variance.m* regarding both input variables set to 1, and a criterion parameter for *meanVariance.m* regarding y set to 1, suggests the data segments in Figure 4.2 as identification and validation data sets. These are set to be equal as only one segment meets all the criteria set. With this window size, the steps earlier in the data set does not overlap, and are considered not interesting.

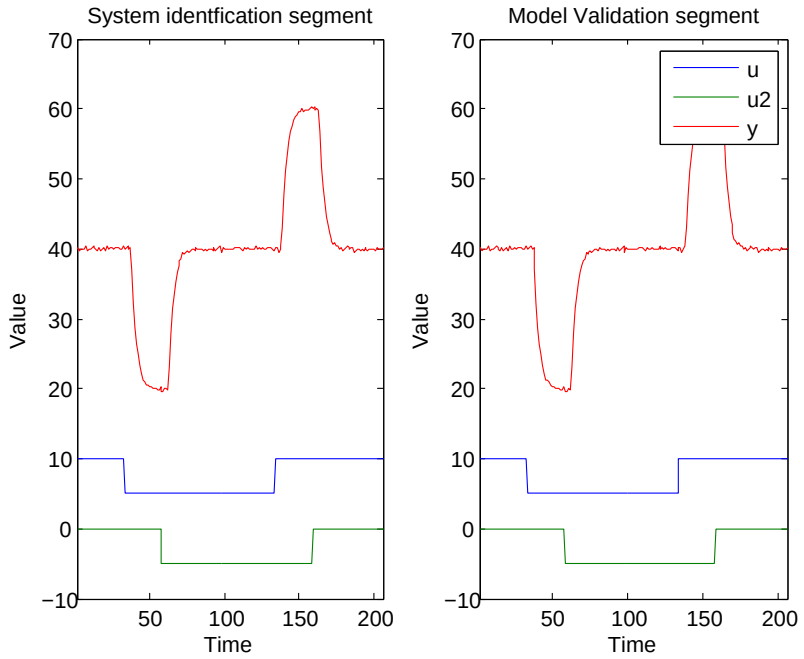

Figure 4.2: Data segments suggested by $shdmiMISO.m$.

Chapter 5

Discussion

In this chapter topics mentioned in Chapter 3 and 4 will be discussed.

5.0.1 Sensitivity to Noise

Noise affecting a variable is one of the most deciding factors considering a criterion function's ability to recognise an 'interesting' segment of data. If the function is sensitive to noise, steady state segments of the data set can arbitrarily trigger the function's criterion, and setting the criterion parameter becomes difficult. As mentioned in Chapter 3, the criterion function *inputStep.m* can be sensitive to noise. If the amplitude of the noise exceeds the criterion parameter, the criterion is triggered, and the criterion is rendered useless. In closed loop scenarios where the manipulated variable (input data) is responding to changes in the set point and the controlled variable (output data), it 'inherits' possible noise from the controlled variable, and may indeed be affected by a measurement noise directly. This, alongside physical constraints, results in a manipulated variable never performs steps. The set point of the controller however, if set directly by a user and not e. g. in a cascade control scheme. If one is to consider the set point as a variable, *inputStep.m* is included in SHDMI (and as a conceptual criterion function).

To cope with noise, calculation of mean values per sample in a sliding window scheme was introduced. The mean values lie close to the the actual variable values (Figure 3.8), and *meanChange.m* search for a difference quotient(between subsequent samples in the mean values) larger than its cri-

terion parameter. At high noise levels, the the window used in calculations have to be long to 'cancel' out the noise. Comparing window sizes of 30 and 10 at a relatively low noise level(Figure 5.1), one can see that the difference quotient between subsequent samples gets harder to distinguish when the window increases in length. This implies that the window size should be small, but considering noise it should be long. This renders *meanChange.m* to only be useful at low noise levels. Pre filtering the data set to reduce noise(low pass filter) would be beneficial, and this is often done in practice. Doing this, one can also calculate the difference quotients of the variable directly.

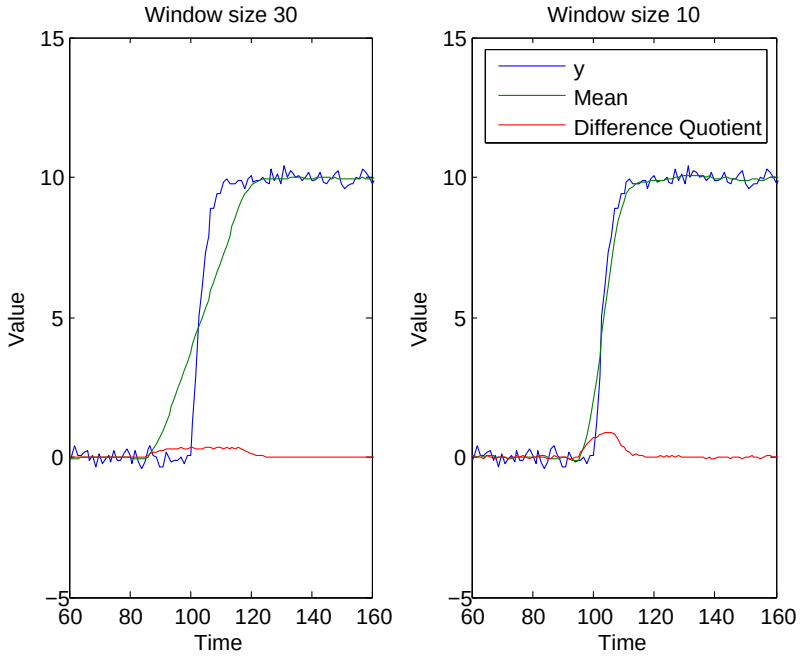

Figure 5.1: Mean values and difference quotients per sample, calculated by *meanChange.m*.

To apply criteria to whole windows/segments variance was introduced as a criterion. This was implemented in the criterion function *variance.m*, regarding a variable directly, and in *meanVariance.m*, regarding calculated mean values of a variable. Obviously *variance.m* is sensitive to noise, as variance calculated at steady state will be biased from zero. Figure 5.2 shows that at low noise levels the transient in y is clearly distinguished from steady

state by the variance calculated in *variance.m*, and the criterion parameter can be set low(as steady state variance is approximately zero). At a higher noise level, the transient still is distinguished from steady state, but setting the criterion parameter becomes more difficult, as one cannot count on a lower boundary of the parameter.

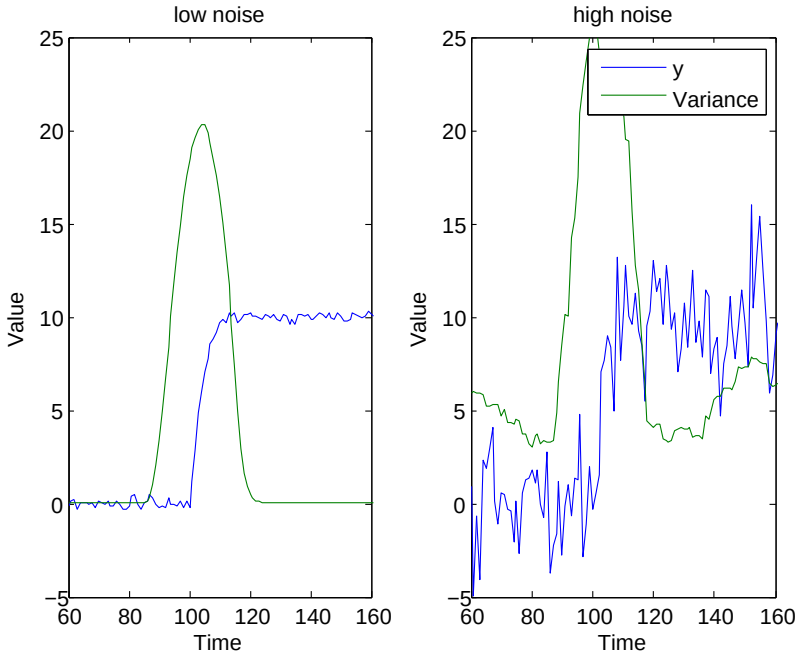

Figure 5.2: Variance per sample, calculated by *variance.m*.

meanVariance.m was implemented as a merging of the ideas in *meanChange.m* and *variance.m*. By calculating variance of calculated mean values, a transient can be distinguished from steady state at high noise levels(Figure **??**). Figure 5.3 illustrates that at low noise levels, variance of the variable directly might be a better criterion as the variance of its mean values are lower(the mean values transient is 'slower' than the actual variables transient). But Figure 5.3 also show that as the level of noise close to equal the transient in the variable, *meanVariance.m* is still able to distinguish the change of mean values in a satisfactory way(steady state variance of mean values close to zero). This enables the user to determine if a data set contains interesting data, even before pre filtering the data.

Figure 5.4 shows an 'impulse response' of a variable(low and high noise).

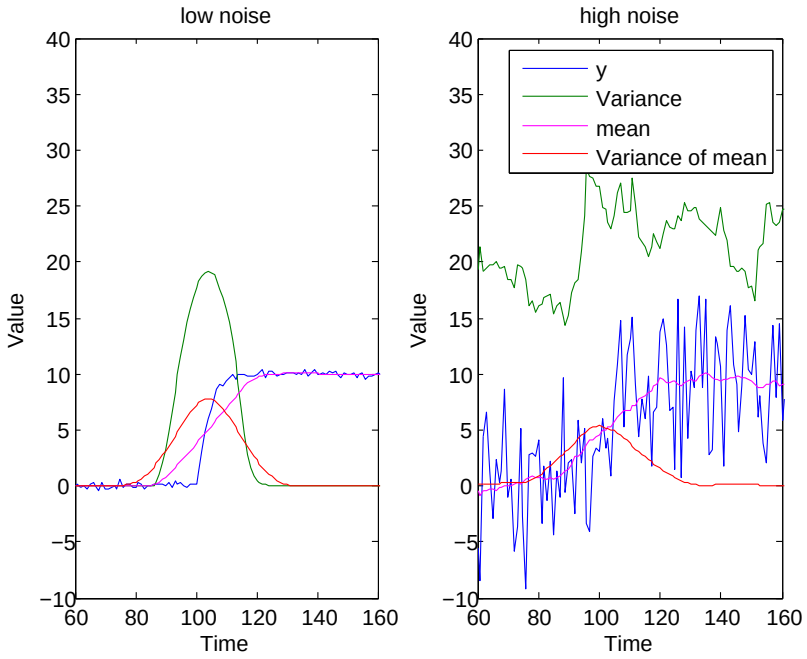

Figure 5.3: Variance, mean values and variance of mean values per sample, calculated by *variance.m* and *meanVariance.m*

Even though the variance of mean values calculated by *meanVariance.m* appear to be small, this variance is calculated to be approximately zero at steady state, and the user is able to set the criterion parameter to be close to zero.

5.0.2 Delay

With a time delay in the system, running a criterion search on only one system variable can result in a data segment that does not include both the input step and response transient of the system(Section 3.1.6). Extending the window used by the criterion functions could help, but by combining criteria regarding both input and output data help ensure that steps and responses are included in returned data segments. The size of the window, however, must be long enough to parallel the length of the time delay. The function *criteriaCombination.m* also allows the user to combine several search criteria for each system variable, if this is desired.

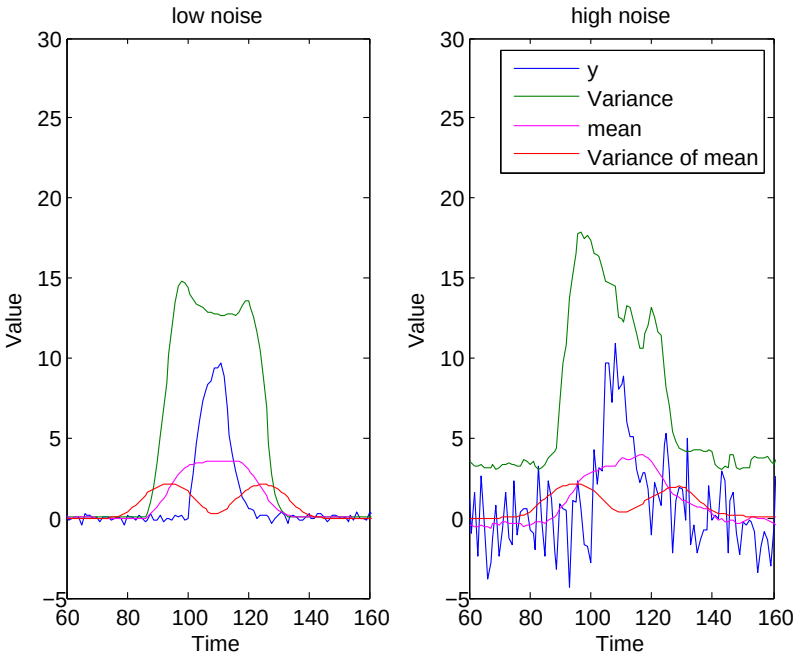

Figure 5.4: Variance, mean values, and variance of mean values per sample, calculated by *variance.m* and *meanVariance.m*

5.0.3 Setting Criterion Parameters and Window Size

SHDMI set out to be simple and easy to use. The criterion parameters appear easy to set for the user, but noise can make it more difficult (as discussed earlier). Implementing different criterion functions culminated in *meanVar.m*, which appear as the most practical. But a window size to small or large could render also *meanVariance.m* ineffective(small window: mean values does not 'cancel out' noise, large window: mean values change slowly at transients (Figure 5.4)). The window size parameter also affects the the combination of criteria, and have to be 'large enough'. In practice it seems difficult to set these parameters[1], but prior knowledge of the plant can be of great help. E. g. knowing the approximate time delay of the plant/system lets you choose a long enough window. Knowing this, coupled with the properties of *meanVariance.m* (criterion parameter can be set close

[1]Calculated mean values, variance etc. is plotted so that the user cam get an idea of what the criterion parameters/window size should be.

to zero), standards for that particular plant can be developed.

5.0.4 Choosing Data Segments

SHDMI separates interesting from not interesting data in a binary fashion (1: sample interesting, 0: sample not interesting). This enables the prototype of the concept to carry results from different criterion functions through combination, and deliver it to *segmentSelect.m*, but it has no qualitative measure for the selection of which data segments to use. An improvement as simple as increasing values (instead of binary) parallel the size of the variance of a variable in a segment, could enable SHDMI to differentiate between quality of the segments. *segmentSelect.m* truly is a prototype function as its only use is to set the first two, or only data segment, considered interesting, as identification and validation data sets, and improvements should be implemented, such as letting the user manually/visually decide which data segments to use as what.

5.0.5 MIMO Expansion

In Chapter 4 the possibility to expand SHDMI to use with MIMO-systems was introduced. The structure of SDHMI allows implementation to expand the input and output variables to as many as one would like. What needs to be done to achieve this is adding variable data and criterion parameters in the input argument list of *shdmi.m*, and criterion functions to consider these variables in the sequential framework. The advantage of using SHDMI to search MIMO-system data sets is its ability to combine search criteria for all the current paired variables of the system, to ensure data segments contain the correlation between them.

5.0.6 Other Criteria

The structure of SHDMI also allows simple implementation of additional criterion functions. Structured like the criterion functions in this thesis, a new criterion simply has to be added to the sequence of *shdmi.m* (followed

by a criterion combination), and be accounted for in the input argument list of *shdmi.m*.

Chapter 6

Conclusions

The main objective of this thesis has been to develop and implement a concept to find possible segments of historic data sets suited for model identification. The concept, named SHDMI after the title of the thesis, set out to be easy to use, drawing on commonly known terms (e.g. variance), and demanding only a basic understanding of a dynamic system's behaviour[1].

To distinguish interesting data from stationary data, different search criteria was implemented. The reliability of these search criteria varied. The most reliable criterion was implemented in $meanVariance.m$, as it showed little sensitivity to noise. This allows the user to set the criterion parameter of the function without worrying about stationary segments being considered interesting because of the noise. Other search criteria, such as the one implemented in *variance.m* is effective at low noise levels. The user was also given the ability to choose from, and combine the different search criteria, to ensure that time delays are accounted for in the returned data segments.

The resulting prototype of SHDMI shows promise as a tool in its simplicity, and structure. Expanding its concepts to use with MIMO-systems underlines this conclusion. However there is potential for improvements and expansions, such as guidelines for selecting criterion parameters and window sizes, and implementing criteria for selecting identification and validation data sets.

[1] I.e. how input and output data of a system correlate.

Further Work

- Implement other search criteria.
- Develop guidelines for setting criterion parameters.
- Propose, and implement a qualitative measure for information in data segments.
- Pursue the proposed expansion of SHDMI concerning MIMO-systems.
- Implement SHDMI in a visual environment.

Appendix A

Matlab Implementation

A.1 inputStep.m

```
function[returnedData] = inputStep(u, windowSize, stepSize)

size = length(u);

returnedData = zeros(size,1);

for i = 2:(windowSize/2)
    if(u(i) >= (u(i-1) + stepSize) || u(i) <= (u(i-1) - stepSize))
        returnedData(1:windowSize) = 1;
    end
end

for i = ((windowSize/2)+1):(size-(windowSize/2))
    if(u(i) >= (u(i-1) + stepSize) || u(i) <= (u(i-1) - stepSize))
        window = (i-(windowSize/2)):(i+(windowSize/2));
        returnedData(window) = 1;

    end
end
```

```
for i = (size-(windowSize/2)+1):size
    if(u(i) >= (u(i-1) + stepSize) || u(i) <= (u(i-1) - stepSize))
        returnedData((size-windowSize+1):size) = 1;
    end
end
```

A.2 meanChange.m

```
function [returnedData] = meanChange(y, windowSize, minChange)

    size = length(y);

    returnedData = zeros(size,1);
    means = zeros(size, 1);
    meanChanges = zeros(size, 1);

    for i = ((windowSize/2)+1):(size - (windowSize/2))

        sum = 0;

        for j = (i - (windowSize/2)):(i + (windowSize/2))

            sum = sum + y(j);

        end

        mean = sum/(windowSize + 1);

        means(i) = mean;

    end

    for i = ((windowSize/2)+1):(size - (windowSize/2))
```

```
        meanChange = means(i) - means(i-1);
        meanChanges(i) = meanChange;
        if(meanChange >= minChange || meanChange <= (-minChange))

            window = (i-(windowSize/2)):(i+(windowSize/2));
            returnedData(window) = 1;

        end

    end

     figure
     plot(1:length(means), means)
     legend('Mean calculated by meanChange.m')
     xlabel('Time')
     ylabel('Mean')

     axis([0 length(means) -10 30])

end
```

A.3 variance.m

```
function[returnedData] = variance(y, windowSize, minVariance)

    size = length(y);

    returnedData = zeros(size,1);
    var = zeros(size, 1);
    for i = ((windowSize/2)+1):(size - (windowSize/2))
```

```
        sum = 0;
        sumSecond = 0;

        for j = (i - (windowSize/2)):(i + (windowSize/2))

            sum = sum + y(j);
            sumSecond = sumSecond + ((y(j))^2);

        end

        mean = sum/(windowSize + 1);
        meanSecond = sumSecond/(windowSize + 1);

        variance = meanSecond - (mean^2);
        var(i) = variance;
        if(variance >= minVariance)

            window = (i-(windowSize/2)):(i+(windowSize/2));
            returnedData(window) = 1;

        end

    end

    figure
    plot(1:length(var), var)
    legend('Variance')
    xlabel('Time')
    ylabel('Variance calculated by variance.m')

    axis([0 length(var) -10 30])
```

A.4 meanVariance.m

```
function [returnedData] = meanVariance(y, windowSize, minVariance)

    size = length(y);

    returnedData = zeros(size,1);
    means = zeros(size, 1);
    meanVar = zeros(size, 1);

    for i = ((windowSize/2)+1):(size - (windowSize/2))

        sum = 0;

        for j = (i - (windowSize/2)):(i + (windowSize/2))

            sum = sum + y(j);

        end

        mean = sum/(windowSize + 1);

        means(i) = mean;

    end

     figure
     plot(1:length(means), means)
     legend('Mean calculated by meanVariance.m')
     xlabel('Time')
     ylabel('Mean')

     axis([0 length(means) -10 30])
```

```
for i = (windowSize + 1):(size - windowSize)

    meanSum = 0;
    meanSumSecond = 0;

    for j = (i - (windowSize/2)):(i + (windowSize/2))

        meanSum = meanSum + means(j);
        meanSumSecond = meanSumSecond + ((means(j))^2);

    end

    meanMean = meanSum/(windowSize + 1);
    meanMeanSecond = meanSumSecond/(windowSize + 1);

    variance = meanMeanSecond - (meanMean^2);
    meanVar(i) = variance;

    if(variance >= minVariance)

        window = (i - (windowSize/2)):(i+ (windowSize/2));
        returnedData(window) = 1;

    end

end
 figure
 plot(1:length(meanVar), meanVar)
 legend('Variance')
 xlabel('Time')
 ylabel('Variance calculated by meanVariance.m')

 axis([0 length(means) -10 30])
```

```
end
```

A.5 criterionCombination.m

```
function[vectorOut] = criteriaCombination(vectorOne, vectorTwo)

    size = length(vectorOne);
    vectorOut = zeros(size,1);

    for i = 1:size

        switchCaseFlag = 0;

        if (vectorOne(i) == 1)

            switchCaseFlag = 1;

        end
        if (vectorTwo(i) == 1)

            switchCaseFlag = 2;

        end

        switch switchCaseFlag

            case 1
                match = 0;
                for j = i:size

                    if(vectorOne(j) == 0 && vectorTwo(j) == 0)

                        i = j;
```

```
                break;
            end

            if(vectorTwo(j) == 1)

                match = 1;

            end

            if(match == 1)
                vectorOut(i:j) = 1;
            end

        end

    case 2

        match = 0;
        for j = i:size

            if(vectorTwo(j) == 0 && vectorOne(j) == 0)

                i = j;

                break;
            end

            if(vectorOne(j) == 1)

                match = 1;

            end
```

```
                    if(match == 1)
                        vectorOut(i:j) = 1;
                    end

                end

        end

    end
```

A.6 segmentSelect.m

```
function[sysIdSegmentU, sysIdSegmentY, valSegmentU, valSegmentY] =
                                        segmentSelect(u, y, returnedData)

    size = length(u);

    sysIdSegmentU = [];
    sysIdSegmentY = [];
    valSegmentU = [];
    valSegmentY = [];
    k = 1;

    for i = 1:size

        if(returnedData(i) == 1)

            for j = i:size

                if(returnedData(j) == 0)

                    sysIdSegmentU = u(i:(j-1));
                    sysIdSegmentY = y(i:(j-1));
```

```
                k = j;
                break;

            end

        end
        break;
    end

end

flag = 0;
for m = k:size

    if(returnedData(m) == 1)
        flag = 1;
        for n = m:size

            if(returnedData(n) == 0 || k == size)

                valSegmentU = u(m:(n-1));
                valSegmentY = y(m:(n-1));
                break;

            end

        end
        break;
    end

end

if(flag == 0)
    valSegmentU = sysIdSegmentU;
```

```
        valSegmentY = sysIdSegmentY;
    end
```

A.7 shdmi.m

```
function [sysIdSegmentU, sysIdSegmentY, valSegmentU, valSegmentY] =
shdmi(u, y, windowSize, stepSize, minMeanChangeU, minMeanChangeY,
minVarianceU, minVarianceY, minMeanVarU, minMeanVarY)

    size = length(u);

    firstCriterion = 0;     % 0: first vector to be combined
                            % 1: not first vector to be combined

    returnedData = [];      % vector to hold returned data from functions


    %%%%%%%%%%%%%%%%%% inputStep.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if(stepSize ~= 0)

        returnedDataStep = inputStep(u, windowSize, stepSize);

        if(firstCriterion == 0)

            returnedData = criteriaCombination(returnedDataStep,
                                               returnedDataStep);
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataStep,
                                                   returnedData);

        end
```

```
end

%%%%%%%%%%%%%%%%%% meanChange.m U %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(minMeanChangeU ~= 0)

    returnedDataMeanChangeU = meanChange(u, windowSize,
                                         minMeanChangeU);

    if(firstCriterion == 0)

        returnedData = criteriaCombination(returnedDataMeanChangeU,
                                           returnedDataMeanChangeU);
        firstCriterion = 1;

    else

        returnedData = criteriaCombination(returnedDataMeanChangeU,
                                                 returnedData);

    end

end

%%%%%%%%%%%%%%%%%% meanChange.m Y %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(minMeanChangeY ~= 0)

    returnedDataMeanChangeY = meanChange(y, windowSize,
                                         minMeanChangeY);

    if(firstCriterion == 0)

        returnedData = criteriaCombination(returnedDataMeanChangeY,
                                         returnedDataMeanChangeY);
```

```
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataMeanChangeY,
                                                    returnedData);

        end

    end


    %%%%%%%%%%%%%% variance.m U %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if(minVarianceU ~= 0)

        returnedDataVarU = variance(u, windowSize, minVarianceU);

        if(firstCriterion == 0)

            returnedData = criteriaCombination(returnedDataVarU,
                                                returnedDataVarU);
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataVarU,
                                                    returnedData);

        end

    end

    %%%%%%%%%%%%%%%%% variance.m Y %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if(minVarianceY ~= 0)
```

```
        returnedDataVarY = variance(y, windowSize, minVarianceY);

        if(firstCriterion == 0)

            returnedData = criteriaCombination(returnedDataVarY,
                                                returnedDataVarY);
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataVarY,
                                                  returnedData);

        end

    end


    %%%%%%%%%%%%% meanVariance.m U %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if(minMeanVarU ~= 0)

        returnedDataMeanVarU = meanVariance(u, windowSize, minMeanVarU);

        if(firstCriterion == 0)

            returnedData = criteriaCombination(returnedDataMeanVarU,
                                                returnedDataMeanVarU);
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataMeanVarU,
```

```
                                                    returnedData);

        end

    end

     %%%%%%%%%%%%% meanVariance.m Y %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if(minMeanVarY ~= 0)

        returnedDataMeanVarY = meanVariance(y, windowSize, minMeanVarY);

        if(firstCriterion == 0)

            returnedData = criteriaCombination(returnedDataMeanVarY,
                                               returnedDataMeanVarY);
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataMeanVarY,
                                                    returnedData);

        end

    end

    %%%%%%%%%%%%% segmentSelect.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    sysIdSegmentU = [];
    sysIdSegmentY = [];
    valSegmentU = [];
    valSegmentY = [];
```

```
    [sysIdSegmentU, sysIdSegmentY, valSegmentU, valSegmentY] =
                                        segmentSelect(u, y, returnedData);
```

A.8 shdmiMISO.m

```
function [sysIdSegmentU, sysIdSegmentU2, sysIdSegmentY, valSegmentU,
    valSegmentU2, valSegmentY] = shdmiMISO(u, u2, y, windowSize,
    minVarianceU, minVarianceU2, minVarianceY, minMeanVarU, minMeanVarU2,
    minMeanVarY)

    size = length(u);

    firstCriterion = 0;     % 0: first vector to be combined
                            % 1: not first vector to be combined

    returnedData = [];      % vector to hold returned data from functions

    %%%%%%%%%%%%% variance.m U %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if(minVarianceU ~= 0)

        returnedDataVarU = variance(u, windowSize, minVarianceU);

        if(firstCriterion == 0)

            returnedData = criteriaCombination(returnedDataVarU,
                                                    returnedDataVarU);
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataVarU,
                                                      returnedData);

        end
```

```
end

%%%%%%%%%%%%%%% variance.m U2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(minVarianceU2 ~= 0)

    returnedDataVarU2 = variance(u2, windowSize, minVarianceU2);

    if(firstCriterion == 0)

        returnedData = criteriaCombination(returnedDataVarU2,
                                           returnedDataVarU2);
        firstCriterion = 1;

    else

        returnedData = criteriaCombination(returnedDataVarU2,
                                             returnedData);

    end

end

%%%%%%%%%%%%%%%%% variance.m Y %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(minVarianceY ~= 0)

    returnedDataVarY = variance(y, windowSize, minVarianceY);

    if(firstCriterion == 0)

        returnedData = criteriaCombination(returnedDataVarY,
                                           returnedDataVarY);
        firstCriterion = 1;
```

```
        else

            returnedData = criteriaCombination(returnedDataVarY,
                                                    returnedData);

        end

    end

    %%%%%%%%%%%%% meanVariance.m U %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if(minMeanVarU ~= 0)

        returnedDataMeanVarU = meanVariance(u, windowSize, minMeanVarU);

        if(firstCriterion == 0)

            returnedData = criteriaCombination(returnedDataMeanVarU,
                                                returnedDataMeanVarU);
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataMeanVarU,
                                                        returnedData);

        end

    end

    %%%%%%%%%%%%% meanVariance.m U2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if(minMeanVarU2 ~= 0)
```

```
        returnedDataMeanVarU2 = meanVariance(u2, windowSize, minMeanVarU2);

        if(firstCriterion == 0)

            returnedData = criteriaCombination(returnedDataMeanVarU2,
                                                returnedDataMeanVarU2);
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataMeanVarU2,
                                                    returnedData);

        end

    end

     %%%%%%%%%%%%% meanVariance.m Y %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if(minMeanVarY ~= 0)

        returnedDataMeanVarY = meanVariance(y, windowSize, minMeanVarY);

        if(firstCriterion == 0)

            returnedData = criteriaCombination(returnedDataMeanVarY,
                                                returnedDataMeanVarY);
            firstCriterion = 1;

        else

            returnedData = criteriaCombination(returnedDataMeanVarY,
                                                    returnedData);
```

```
        end

    end

    %%%%%%%%%%%% segmentSelect.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    sysIdSegmentU = [];
    sysIdSegmentU2 = [];
    sysIdSegmentY = [];
    valSegmentU = [];
    valSegmentU2 = [];
    valSegmentY = [];

    [sysIdSegmentU, sysIdSegmentU2, sysIdSegmentY, valSegmentU,
   valSegmentU2, valSegmentY] = segmentSelectMISO(u, u2, y, returnedData);
```

A.9 segmentSelectMISO.m

```
function[sysIdSegmentU, sysIdSegmentU2, sysIdSegmentY, valSegmentU,
    valSegmentU2, valSegmentY] = segmentSelectMISO(u, u2, y, returnedData)

    size = length(u);

    sysIdSegmentU = [];
    sysIdSegmentU2 = [];
    sysIdSegmentY = [];
    valSegmentU = [];
    valSegmentU2 = [];
    valSegmentY = [];
    k = 1;

    for i = 1:size

        if(returnedData(i) == 1)
```

```
        for j = i:size

            if(returnedData(j) == 0)

                sysIdSegmentU = u(i:(j-1));
                sysIdSegmentU2 = u2(i:(j-1));
                sysIdSegmentY = y(i:(j-1));
                k = j;
                break;

            end

        end
        break;
    end

end

flag = 0;
for m = k:size

    if(returnedData(m) == 1)
        flag = 1;
        for n = m:size

            if(returnedData(n) == 0 || k == size)

                valSegmentU = u(m:(n-1));
                valSegmentU2 = u2(m:(n-1));
                valSegmentY = y(m:(n-1));
                break;

            end
```

```
            end
            break;
        end

    end

    if(flag == 0)
        valSegmentU = sysIdSegmentU;
        valSegmentU2 = sysIdSegmentU2;
        valSegmentY = sysIdSegmentY;
    end
```

Bibliography

Beck, M. (1985). Structures, failure, interference and prediction, *Identification and System Parameter Estimation (Proceedings of 7th Symposium Volume 2, July 1985)* pp. 1443–1448.

Beck, M. (1987). Water quality modelling: a review of the analysis of uncertainty, *Water Resources Research* **23**: 1393–1442.

Beven, K. (2004). Generalized likelihood uncertainty estimation (GLUE), PUB-IAHS Workshop.

Blanke, M., Kinnaert, M., Lunze, J. & Staroswiecki, M. (2006). *Diagnosis and Fault-Tolerant Control*, second edn, Springer Berlin Heidelberg.

Brown, R. G. & Hwang, P. Y. (1997). *Introduction to Random Signals and Applied Kalman Filtering*, third edn, John Wiley & Sons Inc.

Fordal, A. O. (2008). Identifiability analysis of process data, *Technical report*, Department of engineering cybernetics, Norwegian University of Science and Technology.

Fordal, A. O. (2010). *Process data mining for parameter estimation*, Master's thesis, Department of engineering cybernetics, Norwegian University of Science and Technology.

Goodwin, G. C. & Payne, R. L. (1977). *Dynamic System Identification: Experiment Design and Data analysis*, Academic Press.

Goswami, J. C. & Chan, A. K. (1999). *Fundamentals of wavelets - theory, algorithms, and applications*, John Wiley & Sons Inc.

Hornberger, G. M. & Spear, R. C. (1981). An approach to the preliminary analysis of environmental systems, *Journal of environmental management* **12**: 7–18.

Ljung, L. (1999). *System Identification*, second edn, Prentice Hall P T R.

Mathworks (2010). *System Identification Toolbox, User's Guide*.

Pintelon, R. & Schoukens, J. (2001). *System Identification: A Frequency Domain Approach*, IEEE Press.

Spear, R. C. & Hornberger, G. M. (1980). Eutrophication in peel inlet - ii - identification of critical uncertainties via generalized sensitivity analysis, *Water research* **14**: 43–49.

Stedinger, J. R., Vogel, R. M., Lee, S. U. & Batchelder, R. (2008). Appraisal of the generalized likelihood uncertainty estimation (GLUE) method, *Water resources research* **44**: 17 pp.

Wagener, T., Camacho, L. A. & Wheater, H. S. (2002). Dynamic identifiability analysis of the transient storage model for solute transport in rivers, *Journal of Hydroinformatics* **04**(3): 199–211.

Wagener, T., McIntyre, N., Lees, M. J., Wheater, H. S. & Gupta, H. V. (2003). Towards reduced uncertainty in conceptual rainfall-runoff modelling: Dynamic identifiability analysis, *Hydrological Processes* **17**: 455–476.

Wagener, T. & Weather, H. S. (2004). *Monte-Carlo Analysis Toolbox User Manual, Version 5*.