Developing continuous approximation models of manufacturing systems

J.A.W.M. van Eekelen

SE 420337

Master's thesis

Supervisor: prof.dr.ir. J.E. Rooda
Coach: dr.ir. A.A.J. Lefeber

Technische Universiteit Eindhoven
Department of Mechanical Engineering
Systems Engineering Group

Eindhoven, May 2003

EINDSTUDIE-OPDRACHT

TECHNISCHE UNIVERSITEIT EINDHOVEN januari 2001
Faculteit Werktuigbouwkunde
Sectie Systems Engineering

Student	J.A.W.M. van Eekelen
Hoogleraar	Prof.dr.ir. J.E. Rooda
Begeleider	Dr.ir. A.A.J. Lefeber
Start	januari 2001
Einde	februari 2003
Titel	Modellen voor het regelen van fabrieken: bewerken kost tijd

Onderwerp

Fabricagesystemen worden steeds complexer. Een gevolg hiervan is dat het steeds moeilijker wordt een productiesysteem te beheersen. De producent, de markt en de klant vragen om productie waaraan steeds hogere eisen worden gesteld. Regelen van fabricagesystemen wordt daarom steeds belangrijker. De klassieke regeltheorie gaat uit van continue modellen, terwijl fabricagesystemen vaak een discrete-event karakter hebben. In het stroommodel van Kimemia en Gershwin (1983) wordt lijnproductie gemodelleerd als een continue stroom in plaats van discrete delen. Deze vereenvoudiging opende de weg naar het gebruik van technieken uit de optimale besturingstheorie en dit heeft vervolgens tot een aantal nieuwe inzichten en oplossingen geleid. Deze modellen bevatten echter geen informatie over doorlooptijden.
Een kenmerkend voorbeeld is de semi-conductor industrie, waar honderden bewerkingsstappen nodig zijn voor de fabricage van IC-wafers. De aanwezige variabiliteit en het re-entrant gedrag van de productie maken het regelen van een IC-productiesysteem tot een zeer complex probleem. Bovendien is voor deze complexe systemen niet alleen doorzet, maar ook doorlooptijd van belang.

Opdracht

Inventariseer middels een literatuuronderzoek het gebruik van continue modellen voor het modelleren en regelen van discrete event systemen. Kijk daarbij in het bijzonder naar het gebruik van deze technieken voor het regelen van semi-conductor fabricagesystemen.
Onderzoek mogelijkheden om continue modellen te ontwikkelen waarin onderkend wordt dat bewerkingen tijd kosten. Bovendien zou het mogelijk moeten zijn om op deze modellen technieken uit de besturingstheorie toe te passen, waarbij de praktische toepasbaarheid niet uit het oog mag worden verloren.
Maak eenvoudige discrete-event modellen van zowel eenvoudige fabricagelijnen als van een semiconductor fabricagesysteem. Zorg dat deze modellen een aantal essentiële kenmerken bevatten. Ga hierbij in eerste instantie uit van volledig deterministische systemen. Benader deze discrete-event modellen met de voorgestelde continue modellen en valideer de voorgestelde continue modellen kritisch.
Geef de resultaten van het verrichte onderzoek weer in een verslag en geef aanbevelingen voor verder onderzoek op dit gebied.

Prof.dr.ir. J.E. Rooda Dr.ir. A.A.J. Lefeber

Systems Engineering  Faculteit Werktuigbouwkunde

Preface

Traditionally, an educational period is finished with a 'masterpiece'. This thesis describes my masterpiece of the five years curriculum Mechanical Engineering at the Technische Universiteit Eindhoven. My graduation project took place at the Systems Engineering group. The Systems Engineering Group aims to develop methods, techniques and tools for the design of advanced industrial systems. Within this framework, my research was about the development of continuous approximation models of discrete event manufacturing systems. The different phases in the development process have been described in this report.

My life as a student is about to come to an end. It has been a period with many fantastic activities in addition to the courses, lectures and exams. Many people supported me, helped me and believed in me during my life as a student. I would like to express a sincere word of thanks to all of them.

First of all, I would like to thank my family. They made it possible for me to start my education and supported me all the time. My parents and brother have always been willing to listen to me. An enormous impact on our family was the sudden disease and death of my father halfway through my graduation project. At that time, everything seemed unimportant, so did my graduation project. Recommencing after a period of intense grief has been both a physical and a mental challenge. I have a great admiration for my mother who, plunged into her own mourning, encouraged me to pick up the thread and bring this masterpiece to an end.

I am grateful to the coach of my graduation project, Erjen Lefeber, for his ideas, constructive criticism, mental support, humour and patience during the more than two years of this research. Erjen taught me what performing research is about and made me enthusiastic about continuation of my research activities. This will probably come true in the near future. I would also like to thank my supervisor, professor Rooda, for his help and understanding during the last years of my education. His experience and insight gained over the years gave me a perfect guideline for my academical training and education.

I would like to take the opportunity to thank all fellow students and numerous friends who supported me with advice, understanding and love during the final years of my life as a student. Having a group of people around is of an enormous and invaluable importance. I can not mention all of you explicitly, but a great "thank you" goes to you all.

Joost van Eekelen

Samenvatting (in Dutch)

De sectie Systems Engineering van de faculteit Werktuigbouwkunde aan de Technische Universiteit Eindhoven verricht onderzoek op het gebied van analyseren, modelleren, simuleren, besturen en regelen van het dynamische gedrag van fabricagesystemen. Voor de specificatie van met name discrete event systemen (productie waarbij de producten en/of het productieproces een discreet karakter hebben) is het formalisme χ ontwikkeld [Roo95]. Door gebruik te maken van χ is veel inzicht verkregen in het dynamische gedrag van discrete en hybride fabricagesystemen.

Omdat de complexiteit van fabricagesystemen toeneemt, groeit ook het verlangen naar 'slimme' en robuuste regelsystemen voor dit soort systemen. Hierbij kan 'slim' opgevat worden als tijds-efficiënt, kosten-efficiënt, materiaal-efficiënt, etc. Verschillende heuristieken en algoritmen zijn de afgelopen decennia ontwikkeld voor grondstoffenbeheer, productieplanning en machinetoewijzing. Een andere benadering is het gebruik van 'klassieke' regeltheorie voor fabricagesystemen. Deze regeltheorie is op grote schaal beschikbaar voor het regelen van (liefst lineaire) continue systemen. Het in dit rapport beschreven onderzoek is gericht op het ontwikkelen van continue benaderingsmodellen van discrete fabricagesystemen. Zowel eenvoudige korte productielijnen als meer ingewikkelde 're-entrant' lijnen komen aan bod. Een typisch voorbeeld van zo'n systeem is een chipfabriek, waarin de producten (wafers) meerdere malen langs dezelfde machines moeten voordat ze af zijn. Kenmerkend voor de productiesystemen in dit onderzoek is het feit dat nergens assemblage of demontage plaatsvindt.

Belangrijke prestatie-indicatoren van een productiesysteem zijn de doorzet (het aantal producten dat per tijdseenheid geproduceerd kan worden) en de doorlooptijd (hoe lang doet een product over zijn ronde door de fabriek). Omdat doorlooptijd zo belangrijk is, dient de bewerkingstijd van machines opgenomen te worden in de te ontwikkelen modellen. Dit is geen triviaal probleem met een pasklare oplossing. Een belangrijk deel van dit onderzoek is daarom gericht op de modellering van tijdsvertraging in de productenstroom die ontstaat als gevolg van een bewerkingsstap. Bij deze modellen is het belangrijk te weten wat de systeemgrenzen zijn, wat de toepasbaarheid van het model is en de validiteit ervan. Omdat verschillende soorten regeltechnieken verschillende soorten modellen verwachten, wordt uitgegaan van één van de meest elementaire modelstructuren: de toestandsruimte beschrijving (state space model). Deze modelstructuur is bruikbaar in een breed scala van regeltechnieken, of converteerbaar naar een bruikbaar model.

Doelstelling van dit onderzoek is dus om discrete fabricagesystemen te benaderen met een continue benaderingsmodel waarin de bewerkingstijden van machines verwerkt zijn. In dit onderzoek is het productiesysteem voorgesteld als een χ-model, terwijl dit in feite al een benadering is van een daadwerkelijk fysisch fabricagesysteem. Die vereenvoudigingsstap is buiten beschouwing gelaten.

De stroom van producten door een fabriek wordt gezien als een continue stroom waarbij afzonderlijke producten niet meer te onderscheiden zijn. De snelheid van de productstroom wordt bepaald door de machinesnelheden en de snelheid waarmee producten het systeem binnenkomen.

Afhankelijk van de hoeveelheid producten in buffers en de momentane machinesnelheden kunnen de machinesnelheden voor een volgende tijdstap bepaald worden. Dit leidt tot een model waarbij de dynamica van een productielijn goed beschreven wordt, maar waarbij de grootte van de tijdsvertraging niet volledig juist voorspeld wordt. Daarnaast heeft het model verschillende dynamica waartussen geschakeld dient te worden. Slechts een beperkte klasse van regeltechnieken kan hiermee overweg en bovendien neemt het aantal dynamica waartussen de regelaar moet schakelen exponentieel toe bij vergroten van de productielijn.

Een andere manier van modelleren is die van systeemidentificatie. Hierbij wordt op basis van metingen aan een systeem en het verwerken van deze meetdata door een identificatie-algoritme een toestandsbeschrijving berekend die het meest overeenkomt met de gemeten waarden. Dit is een zogenaamde 'black box' benadering. In feite wordt geen of beperkte kennis van het fysieke systeem meegegeven aan het algoritme. Dit is een nadeel. Daarnaast is de verkregen toestandsbeschrijving enkel geldig binnen de bandbreedte waarbinnen de metingen hebben plaatsgevonden. Een garantie voor een goed voorspellend model buiten deze grenzen is er niet. Bovendien blijkt deze methode erg rekenintensief te zijn, met alle numerieke en geheugenproblemen van dien. Systeemidentificatie levert goede toestandsbeschrijvingen op voor zeer eenvoudige en kleine modellen, maar bij iets grotere modellen met meer toestanden levert het teveel problemen en nadelen op.

Omdat systeemidentificatie niet tot de gewenste resultaten heeft geleid, is geprobeerd een verklaring hiervoor te vinden. Door een andere modelleertechniek te gebruiken is meer inzicht verkregen in de problemen die optraden. Analytisch zijn overdrachtsfuncties opgesteld die het dynamisch gedrag van buffers en machines benaderen. De tijdsvertraging als gevolg van de bewerkingstijd van een product is benaderd met behulp van Padé-benadering. Het koppelen van deze overdrachtsfuncties en gebruik maken van realisatietechniek levert een toestandsbeschrijving op van het fabricagesysteem. Deze toestandsbeschrijving is niet bruikbaar in situaties waarbij buffers leeg kunnen raken. Noch houdt het rekening met gelimiteerde machinecapaciteiten. Het is alleen bruikbaar in situaties waarin buffers een oneindige capaciteit hebben. Dit lijkt de modellen geschikt te maken voor een beschrijving rond een evenwichtssituatie. De systeemgrenzen dienen in een regelaarontwerp meegenomen te worden.

Summary

The Systems Engineering Group of the Department of Mechanical Engineering of the Eindhoven University of Technology performs research in the field of analysis, modelling, simulation and control of the dynamic behavior of industrial systems. To specify these systems and in particular discrete event systems (production in which products and/or manufacturing system have a discrete character), the χ specification language has been developed [Roo95]. Using the χ specification language a clear insight into modelling both discrete and hybrid manufacturing systems has been gained.

As industrial complexity increases, the desire for 'smart' and robust control strategies for manufacturing systems becomes greater. 'Smart' can be interpreted as time-efficient, cost-efficient, resources-efficient, etc. Several scheduling heuristics and algorithms were developed during the last decades for resource management, production scheduling and machine (time) allocation. A different approach is using 'classical' control theory on manufacturing systems. Classical control theory is widely available for control of (preferably linear) continuous systems. This study focuses on developing continuous approximation models of discrete manufacturing systems. Both simple flowlines and more complex re-entrant flowlines are examined. A typical example of these manufacturing systems is a wafer fabrication facility, in which wafers enter different areas of the plant more than once before being completed. Typical for the manufacturing systems in this research is that neither assembly nor disassembly takes place.

Important performance measures of a manufacturing system are throughput (the production rate) and flow time (the time span a product is in the manufacturing system). As flow time is important, the process time of a machine needs to be included in the continuous approximation models to be developed. This is by far not a trivial problem with an instant solution. Therefore, an important part in this research is focused on modelling this time delay due to production steps. It is important to be aware of the system boundaries, limitations, usability and validity of the models. Different classical control techniques expect different model structures. Therefore, main focus is on developing so called state space models. This model class can be used in a wide variety of control structures, or can be converted into a usable model.

Main objective of this research is to approximate discrete event manufacturing systems with a continuous model in which the process time of machines has been included. All

manufacturing systems are presented as χ-models. Although this χ-model is already a model of the real physical system, this simplification is not taken into account.

The stream of products going through the manufacturing facility is presented as a continuous flow, in which individual products are not distinguishable anymore. The product flow rate is determined by the machine process rates and the rate at which products enter the manufacturing system.

Dependant on the buffer contents and the momentary machine production rates, the production rates for the next time increment can be computed. This leads to a model that describes the dynamics of the system well, but that does not predict the magnitude of the time delay in a proper way. In addition to that, the model has several dynamic modes, between which has to be switched. Only a small class of control techniques is able to cope with this kind of models. Moreover, the number of dynamic modes increases exponentially when expanding the manufacturing system.

Another modelling technique is system identification. Based on measurements of the real system, an algorithm computes the state space description that fits the measurements best. This is a so called 'black box' approach. In fact, little or no knowledge of the physical system is passed to the algorithm. This is a disadvantage. Another disadvantage is the limited bandwidth in which the model can be used. This bandwidth is the bandwidth on which the measurements were taken. Extrapolating the model outside the bandwidth gives absolutely no guarantee of the validity of the results. Moreover, system identification proves to be computationally intensive and time consuming, with all concomitant problems regarding numerics and memory. System identification provides good state space descriptions for very small models. For larger problems this method yields too many problems and disadvantages.

Since system identification has not led into the desired results, an explanation for this has been searched for. Using a different modelling technique, more insight has been obtained in the problems that arose. Analytically, transfer functions have been formulated, representing the dynamical behavior of buffers and machines. The time delay, resulting from the process time of a machine, has been approximated using the Padé method. Coupling these transfer functions and using realization techniques yields a state space description of the manufacturing system. This description is only useful in situations where buffers are always filled and have infinite capacity. In addition, it does not deal with limited machine capacities either. This makes the model usable only in steady state situations or small variations around some equilibrium point. The controller will have to account for the physical limitations.

Contents

Chapter 1

Introduction and concepts

1.1 General

Consumer market has always been a matter of two-way traffic. On the one hand, companies develop new products and create a new market with it. On the other hand, consumers ask for new products. These new products get more and more complex. This complexity expresses itself in functionality, features, design, shape and production method. As production complexity increases, the need for reliable, smart and robust production control strategies increases too. In the past, common sense of workers in a production facility has given several companies a leading position, but common sense is limited too in a demanding, complex and stressful environment. Over the last decades, more and more scheduling heuristics and algorithms emerged to help the worker decide on the shop floor. To develop these algorithms, the manufacturing system had to be described in a very detailed way. Nowadays, modelling of manufacturing systems is a major part of controlling a production facility.

Especially in bulk production, a 'smooth' production is desired: equally spread yield over time. To facilitate a smooth production, control theory might be used. Control theory could provide better solutions in decision-making for complex systems than human decision making. Different control strategies (like PID-control, adaptive control, optimal control or robust control) can be used in production systems. Most of these 'classic' control techniques use continuous models to determine a control law. Most manufacturing systems however are discrete (event) systems. Kimemia and Gershwin [KG83] introduced a method for the continuous representation of a discrete manufacturing system. This approximation method is the basis for the models developed in this study.

The main goal of this research is to develop continuous state models that approximate the discrete event systems. An important point of interest is the modelling of the process time of a machine. The process time of a machine causes a delay in the product

flow. If this delay can be modelled, the continuous approximation models can be used to predict flow times and throughput of both existing and virtual production systems.

In this research, simple manufacturing systems are studied. Simplicity is needed to reveal all phenomena occurring in the discrete manufacturing system and the translation into the continuous model. Next to simple models, re-entrant lines are examined, in which products enter workstations several times before completion. This type of production system is typical for semiconductor manufacturing. Both the complexity of the production process and the high demands on market-time and delivery schedules make the semiconductor industry an ideal example of a manufacturing system at which a newly developed control strategy could be applied to.

This chapter gives an overview of what semiconductor industry is, explains the concepts of 'discrete' and 'continuous' as well as some counterintuitive concepts, details the framework in which this study fits and explains the structure of this report.

1.2 Semiconductor wafer fabrication

In semiconductor industry wafers (round silicium discs) are manufactured. Wafers may contain one hundred to one thousand integrated circuits (ICs or 'chips'). These ICs can be cut from a wafer, attached to a lead frame and packaged in housings (to protect from breaking and corroding and to facilitate handling) to be soldered into a printed circuit board [Cam01]. ICs are used in a very wide variety of products, from computers to coffee machines and from industrial control stations to electric tooth brushes. Production of single wafers takes a few hundred production steps. These processes add (deposition, metallizing or oxidation), alter (diffusion, ion implantation) or remove (etching) a layer of material in selected regions of the wafer surface; the lithography process determines which regions are affected by these processes. Some processes are repeated many times in the recipe. This is called re-entrant behavior.

1.3 Concepts: discrete versus continuous

This report often uses the concepts of 'discrete' and 'continuous'. These words do not always refer to the same things. It is important to know what is meant. Therefore, a short explanation is given here.

Discrete product:	Countable, individual product, like a box, a device, a wafer.
Continuous product:	Product that is not countable, like oil, water, gas.
Discrete time signal:	Signal that is defined only at certain specific values of time.
Continuous time signal:	Signal that takes a value at every point in time.
Discrete state:	A state is discrete if it can only take values of a countable predetermined set.
Continuous state:	A state is continuous if it can take values of an arbitrary set.
Discrete event system:	System in which the dynamics is based on events. These events may possibly have a continuous evolution once they start, but this is not what one is interested in: the primary focus is on the beginning and the end of such events, since ends can cause new beginnings.
Discrete event simulation:	A simulation of a time-evolving stochastic or deterministic system in which changes to the state of the system can only occur at discrete instants.

When a continuous model is referred to, a model containing continuous states, either discrete time of continuous time, is meant. All experiments carried out on a computer system are in fact discrete time experiments, since a computer does not know continuous (analog) time. All discrete event models in this research are deterministic. In fact, the results of the experiments taken with these models can be computed by hand, but this is difficult and very time consuming. That is the reason why they have been simulated. The real benefit of discrete event simulations appears if stochastic computations are made. Performing these calculations by hand is almost impossible if not utterly impossible.

1.4 Against intuition: non-integer or negative products

Constructing continuous models requires thinking on a higher abstraction level. One should leave the idea that only integer number of products exists. In the continuous state situation, a buffer content of 2.5 products is possible, as is 0.23 products. In a model with ordinary linear differential equations, it even is not possible to indicate that a buffer level can only be non-negative. A continuous model can easily contain negative buffer levels, since it simply does not know the physical impossibility. To what extent this is a desirable behavior, depends on the definition of the continuous state. In most real situations, negative buffer levels are undesired. Therefore, watching the continuous states closely is important. In addition to this counterintuitive concept, the probability of a buffer level being exactly 0 or $n \epsilon \mathbb{R}$ equals zero, assuming that the level has left the initial state.

1.5 Against intuition: parallel processing on a single machine multi product workstation

Now that non-integer products are not surprising anymore, the next step in abstraction can be taken. Individual products can not be distinguished anymore in continuous modelling. Therefore, it is unknown which product or what product type a machine is processing at a certain point in time. We only know the rate at which the machine processes products. If a machine can produce more than one producttype, these production rates can both be defined greater than zero. For example, define production rate of product type 1 (m_1) to be 2 products/hour and the production rate of product type 2 (m_2) to be 3 products/hour, then the sequence in which a multi product single machine workstation processes the products is still undefined. Figure 1.1 shows two of the possible sequences. The figure shows two possible product orders. It is also clear

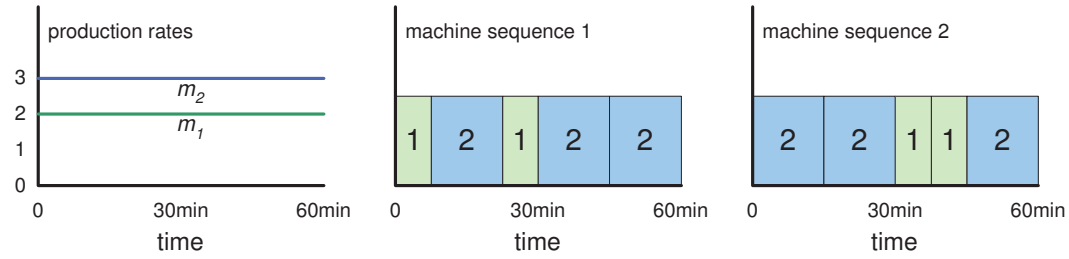


Figure 1.1: Machine production order is not determined by the production rates.

that processing product type 2 takes longer than processing product type 1. This information can not be obtained from the production rate variable only. Therefore, more detailed variables are needed:

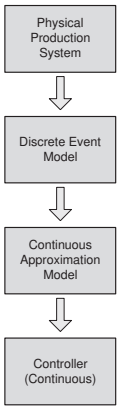
$m_{1,\max}$	maximum production rate of product type 1:	8 lots/hour
$m_{2,\max}$	maximum production rate of product type 2:	4 lots/hour
$m_{1,\text{des}}$	desired production rate of product type 1:	2 lots/hour
$m_{2,\text{des}}$	desired production rate of product type 2:	3 lots/hour
$m_{1,\text{real}}$	real production rate of product type 1:	2 lots/hour
$m_{2,\text{real}}$	real production rate of product type 2:	3 lots/hour

In this case, no difference between the desired and the real production rates exists. In general, this will not be the case. It is possible that a controller asks for more production than the machine capacity. The real machine production rate are then below the desired rate. Also in cases of machine breakdown, the real rate is below the desired rate.

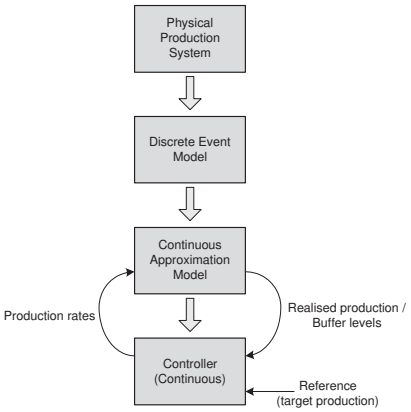
The abstraction level of this research has been stated now. Understanding the global research framework, which this study is part of, is possible now. The research framework is elaborated in the next section.

1.6 Research framework

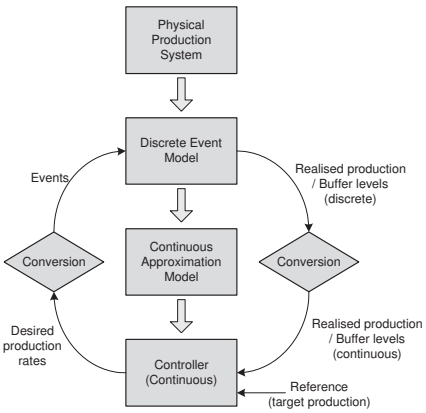
The global structure of this and future research is shown in Figure 1.2. The first step is to translate a physical manufacturing system into a discrete event model (Figure 1.2(a)). Only the material flow is to be modelled, for the control part of the production system is to be redesigned. In this research, the discrete event model is supposed to be equal to the physical manufacturing system and represented as a χ-model. This discrete event model is approximated by a continuous model, which is the main goal of this study.



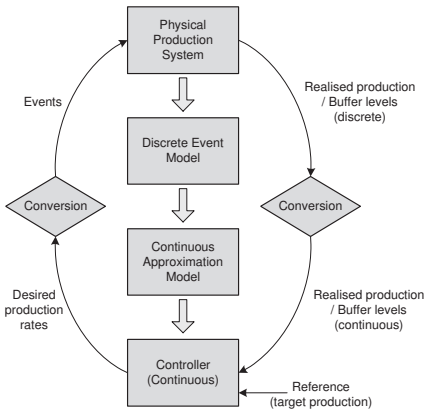
(a) Steps in controller design



(b) Control of the continuous approximation model



(c) Control of the discrete event model



(d) Control of the production system

Figure 1.2: Global overview of the research. The framework presented is partially covered in this study.

Using any regular closed loop control technique, a controller can be designed for this approximation model. The controller can be validated and scenario's can be tested by directly connecting it to the continuous model (Figure 1.2(b)). If the results are satisfactory, the controller can be connected to the discrete event model (Figure 1.2(c)). However, since the controller output is a continuous signal and the discrete event model's input are events, a conversion step is needed. The left conversion block contains an algorithm that translates the continuous desired production rates into explicit events, i.e. allocating jobs to a machine when it becomes idle. This conversion block is very important, because it is the block that triggers the real manufacturing system. A second conversion block translates the discrete event output (for example the buffer levels) into a continuous controller input. In fact, it smoothes the signal that contains only integer values to a continuous signal to facilitate the control process. If the controller works well on the discrete event model, it can be implemented in the real manufacturing system (Figure 1.2(d)). But this can only be done after extensively testing all imaginable scenarios that can take place in the production facility.

This research focuses on the first steps in this framework. The approximation of the discrete event system with a continuous model is very important, because it is the basis of the controller design. A model which does not behave like the discrete event system will most probably not lead to a controller that controls this discrete event system well. A second point of attention in this research is the left conversion block (Figure 1.2) that translates the desired production rates into events. To simulate the discrete event system and to validate the designed continuous models, the conversion block is necessary.

1.7 Social paragraph

Like a lot of academic studies, this research does not directly lead to quickly noticeable payoff for normal consumers. What makes this research valuable?

Complexity of production systems increases. Strict delivery schedules and high demands on reliable high-tech products have a big influence on the production system and its control. Especially products with a short life-cycle and short delivery-time, like cellular phones, must be produced on demand with high volumes and with very specific product requirements and specifications.

Semiconductor wafer fabrication is one of the most complex manufacturing processes. The complexity is brought about by the high-tech processes being applied, their constant development, the multitude of process steps and their reoccurring nature [Cam01]. A well tuned controlled manufacturing system may lead to cost reduction and as a result of this to cheaper products. In addition to this: delivery times may be predicted better, which makes the supplier a more reliable partner.

Moreover, a controlled manufacturing system has some environmental benefits. More economical use of resources leads to less pollution, saves energy and gives less stress to

people. On the other hand, the workforce in production facilities might be restricted due to the savings. Whether this is desirable, is left to the reader to judge.

1.8 Outline of this report

After a literature review of continuous modelling of discrete event systems and control of these systems (Chapter 2), an initial study of continuous approximations of simple discrete event flowlines has been made, to encounter all issues involved in continuous modelling (Chapter 3). Of course, the development of continuous models is started with very simple production models and is then extended to larger models with more than one product type. Then in Chapter 4, an academic case study of the continuous representation of a specific re-entrant flowshop has been made, which in turn reveals some difficulties in the modelling method and software. After this case study, in which discrete event performance is compared to the continuous approximation, the continuous modelling method is evaluated critically. A second modelling method is then proposed in Chapter 5. This is the black box method of system identification. Again, very simple models are used to become familiar with the method. More complex models seem to cause difficulties in the identification process. Therefore, a third continuous modelling method, on analytical basis, is proposed and elaborated in Chapter 6. Transfer functions of the manufacturing system are formulated. This leads to a useful continuous model. Reflections are made on this model regarding usability and limitations. Finally, in Chapter 7, conclusions are drawn and recommendations for future research have been formulated.

Chapter 2

Literature review

2.1 General

Knowing the scientific developments on the subjects covered in this research is important. A short overview of different topics is presented here and is used as a starting point for this research. First, continuous modelling of discrete event systems is studied. Different research fields use this type of modelling. Another topic is re-entrant manufacturing systems, like semiconductor industry, and its scheduling and control. A lot of literature is available on this subject. The control part of those systems is mainly based on priority rules and scheduling heuristics, instead of a continuous controller, based on classical control techniques.

2.2 Continuous modelling of discrete event systems

Continuous modelling is best known from fluid flow simulations and computations. This type of modelling can be projected upon discrete systems, i.e. systems without continuous flows, but characterized as continuous flows due to the large amount of both equivalent events and moving of materials.

Early use of continuous modelling of discrete systems is in road traffic computations. An overview of the historic traffic stream models is given in [LMG+96] by Lieu. Traffic is often described in terms of flow, concentration and speed. In the fluid flow analogy, the traffic stream is treated as a one-dimensional compressible fluid. Two basic assumptions in the continuous models are traffic conservation and the one-to-one relationship between speed and density or between flow and density. The first solutions to the conservation equations applied to traffic flow were proposed by Lighthill and Whitham in 1955 [LW55]. Lieu describes both analytical and numerical solutions to traffic flow problems. The latter is widely used in visco-elastic traffic flow models and

high congestion and bottleneck problems. The models evolved are partial differential equations (PDEs)

Another continuous modelling application field of discrete event systems is the field of the manufacturing systems. Kimemia and Gershwin introduced a continuous model for an FMS (Flexible Manufacturing System) in [KG83], based on ordinary differential equations (ODEs). They define the buffer contents of a discrete manufacturing system as a continuous variable $x(t)$. The derivative of the state is the difference between the production rate of the previous machine $u(t)$ and the production rate of the following machine $d(t)$:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = u(t) - d(t). \tag{2.1}$$

The vector $x(t)$, the buffer state, measures the cumulative difference between production and demand. A negative value for a component of $x(t)$ represents the backlogged demand for the corresponding part. A positive value is the size of the inventory stored in the buffer. Ideally, parts in an FMS are produced as they are required keeping the buffer state close to zero. Instead of elaborating further on the continuous model and its properties, Kimemia and Gershwin implemented this model structure in a control policy without further explanation of comments on the boundary conditions or physical limitations. A hierarchical control structure designed to compensate for workstation failure (characterized as an Markov chain, based on probabilities of machine state transitions) and changes in part requirements is proposed in [KG83]. The hierarchical production control scheme is shown in Figure 2.1. At the highest level of the control scheme (generation of decision tables), the control policies are calculated, based on the production schedules. This has to be redone whenever new estimates of parameters are found. At the second level (calculation of short term production rates), the individual production rates are computed, taking into account the buffer states and the machine states. At the next level (calculation of route splits), the routes of individual products are determined through the factory (only possible with multiple machines per workstation). The lowest level of control are scheduling algorithms that dispatch parts into the system and supervise the operations of the workstations.

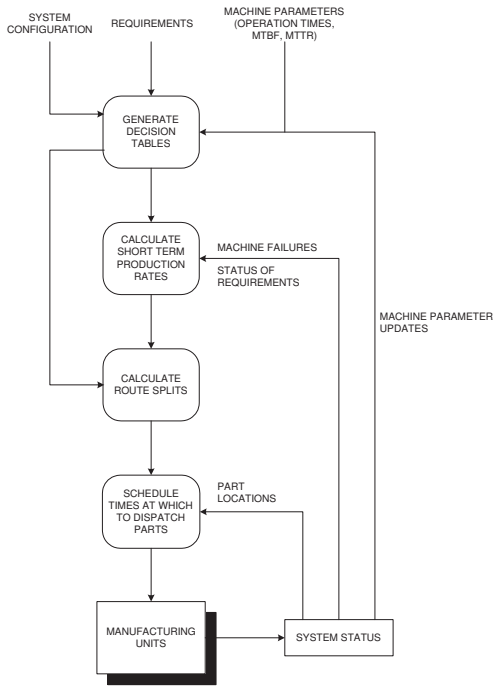


Figure 2.1: Hierarchical production control scheme as proposed in [KG83].

To compute optimal control settings for a job-shop, a cost-function has to be defined. With the cost function and the continuous model, different control settings can be computed taking into account the different machine states (due to failures). In realistic job-shops with a lot of different parts processed, multiple machines at the workstations and a variety of product recipes, the exact solution to this problem is difficult to compute, if possible at all. Therefore, an estimate based control scheme is used. The continuous model is used in the upper two control levels. A simple example in [KG83] shows good results between required and realized production in simulation.

The continuous model proposed by Kimemia and Gershwin is only useful in feedback control situations. The model itself is not a good representation of the discrete event situation. Buffers, for example, can become negative in the continuous model (representing a backlog). In reality, buffers cannot become negative. Moreover, a delay is involved in manufacturing systems. It takes a while before a product leaves the manufacturing system after it had been dispatched into it. The minimum delay is the sum of all process times of the recipe-steps. In Chapter 3 a more detailed continuous model of a discrete event manufacturing system is made.

Almost no extensions have been made in the field of continuous modelling of discrete manufacturing systems. Most literature focuses on control policies and scheduling algorithms rather than on the model itself. Discrepancies between reality (discrete) and the continuous model are taken care of by the controller or scheduling rules.

Recent developments in continuous modelling of discrete systems is handling of internet data traffic or computer network traffic. Similar computations as in road traffic are carried out and very complex data handling algorithms and heuristics have been developed, which is not discussed here.

2.3 Re-entrant lines

Another research topic in this research is the re-entrant line. Semiconductor wafer fabrication takes place in a re-entrant environment. Kumar describes re-entrant lines in [Kum93] as follows: “Traditionally, manufacturing systems have mainly been treated as either job shops or flow shops. In job shops, parts may arrive with random routes, with each route having a low volume. In flow shops, the routes are fixed and acyclic, as in assembly lines. With the advent of semiconductor manufacturing plants, this dichotomy needs to be expanded to consider another class of systems, which are called *re-entrant lines*.” The term ‘re-entrant’ line was used for the first time in this article.

Re-entrant lines differ from flow shops since the part flow route is re-entrant, i.e., non-acyclic. This behavior gives rise to important scheduling problems. Moreover, these re-entrant lines may have few flow routes (sometimes only one), which are high-volume. Depending on whether the re-entrant line is a full scale production facility or a research and development facility used for prototyping, and on whether products manufactured

are memory or logic chips, the system may be classified as either a make-to-stock or make-to-order facility.

One of the most important issues on re-entrant lines is scheduling. In other words: which part should a machine process when it becomes idle? The re-entrant behavior may result in conflicting interests. Processing a part in an early state of completion will generate stock at a later stage, while processing 'older' parts (parts that are further in the product recipe) may cause the buffer to starve if not filled up again with the 'younger' parts.

Kumar describes several kinds of re-entrant lines, discusses scheduling policies, examines the stability of the line for a deterministic model, introduces set-up times and machine failure and stochastic process times. The control of the systems is divided into two parts: the release policy for introducing new parts into the system, and the scheduling policy for determining which part a machine should work on when it becomes idle.

Although written before [Kum93], Lou and Kager describe in [LK89] a robust production control policy for VLSI (Very Large Scale Integration) fabrication. The term *re-entry* has also been used in this article, but generally, Kumar is seen as the first who used this term. Lou describes a production control policy, which he calls *flow rate control*. This control policy is based on the *hedging point control*. Hedging points are determined from demand rates and machine failure behavior. An inventory hedging point is a buffer level which must be kept steady to have the least negative effects of machine breakdown. Based on the machine failure characteristics (mean time before failure MTBF and mean time to repair MTTR) the buffer behind the machine can be given a level so that if the machine breaks down, the buffer can be emptied. Statistics and probability theory are used to compute these hedging points. A different hedging point is the surplus hedging point. The surplus is the difference between the actual and the target production. A surplus hedging point is a computed desired surplus, so that machine failures have minimal effect on the actual production. Lou and Kager add some rules to this policy and then call it flow rate control. This is a closed loop production control policy. In [LK89], it is compared with the open loop uniform loading method and proves better (smoother production than the uniform loading method and no constant backlog). A model of a wafer fabrication model, similar to the structure of Figure 2.2 was used for simulations.

In [LYS+91] Lou c.s. compare different scheduling policies by means of simulation (model shown in Figure 2.2. Four different methods are investigated: *WIP Control* (sets WIP threshold level for each part type at each stage), *Uniform Loading* (open loop policy which uniformly loads parts into the manufacturing system at the beginning of each predefined period), *WIP-to-bottleneck control* (identifies bottleneck of system and sets threshold levels for WIP from the first workstation to the bottleneck workstation) and *Two-Boundary Control* (analogous to flow-rate control in [LK89], with WIP and surplus threshold levels). Robustness of the policies is tested. Robustness is defined as coping with random interferences, like machine breakdown or sudden demand changes. The two-boundary control policy is the most robust one. WIP control outperforms

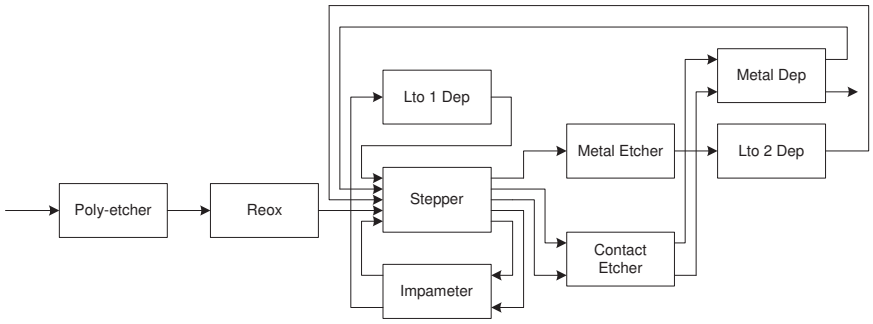


Figure 2.2: Wafer fabrication model used in [LYS+91].

Uniform Loading and WIP-to-bottleneck. The Uniform Loading policy always presents a backlog (as Lou already mentioned in [LK89]) and shows high variability in WIP and backlog.

A more general, but very elaborate, overview of different production planning and scheduling methods is given by Uzsoy in [ULMV92]. This paper also includes a clear overview of production processes in semiconductor industry. Bispo [BT97] reviews related work and distinguishes three categories:

- closed queueing models that address the control of the input of new material into the system;
- open queueing models assuming no control over the input process to address the scheduling problem of each server;
- hierarchical flow control, control with hedging points and thresholds (for instance [KG83, LK89]).

Bispo focuses on multi-product re-entrant flow lines subject to random demand. Simulations were done with the uniform loading method.

In the following chapters, new continuous approximation models of manufacturing lines are developed. In a case study, a re-entrant flowline is examined. This research focuses on the inclusion of process times in the approximation models. The absence of process times in the models found in literature is a lack and therefore investigated in this study.

Chapter 3

The continuous model of a discrete event system

3.1 General

Classical control theory, like PID-control and adaptive control, uses continuous models to develop a controller. These models consist of (ordinary) differential equations, in most cases without discontinuities. Most manufacturing systems can be modelled as discrete event systems. Control theory for discrete event systems is available, but as all possible states of these systems must be included in the model's 'state', this number of states grows exponentially and computation of a controller becomes highly time consuming, if possible at all.
Discrete event systems, as it is in the name, do not behave like continuous systems. However, in particular situations (e.g. high volume production) they might be translated into a continuous model. This model, like every model, is a simplification of the real system, leading to model uncertainties. One of these uncertainties is the fact that in the continuous model, the buffer state can be non-integer as explained in Section 1.4. To be able to use a wide variety of control techniques, the approximating continuous model can be used, keeping the simplifications and uncertainties in mind.

In this report, a continuous model is assumed to be a model with continuous states, but not-necessarily continuous time. Before a continuous model of a semi-conductor plant is presented in an academic case, first of all a simple GBMBME (generator-buffer-machine-buffer-machine-exit) flowline is modelled as a continuous system. Typical problems arise and are dealt with. Then a GBMBME flowline processing two different products is modelled accordingly. This reveals other simple, but necessary, modelling issues, such as machine capacity, utilization and simple scheduling. Finally, a continuous model of a wafer fabrication facility has been made (Chapter 4), using all modelling experience obtained in modelling the simple flowlines.

3.2 Continuous modelling

A discrete event manufacturing system deals with single lots or products. Every product can be distinguished in the manufacturing system. Buffers contain only an integer number of products. In the continuous model, this is taken to a higher abstraction level, as explained in Section 1.4. A machine is described by a rate at which it processes lots. A buffer has an incoming product rate and an outgoing product rate and is modelled as a tank that is filled up and tapped simultaneously. As a result, a buffer may contain a non-integer number of products. This is only possible in the continuous model (in literature also known as 'fluid model': [DW96, DW99]). The main differences between continuous models and discrete event models are described in [Dia96] and stated in Table 3.2.

Since the continuous model does not 'know' that it represents a discrete event system, it allows for negative production rates and negative buffer contents. These model failures are generally not desired in this study, contrary to the model of Kimemia and Gershwin [KG83], for it is not a good representation of the real system. Therefore, production rates and buffer contents are lower-bounded.

The discrete event systems in this research all contain a generator, buffers, machines and an exit-process. The generator puts lots into the manufacturing system. Modelling the lot-generator of a discrete event system in 'push-mode' reveals a tricky problem. When a generator releases e.g. 4 lots per time unit, lots can be re-

Factor	**Continuous modelling**	**Discrete event modelling**
What is modelled	Flows.	Items.
Characteristics	Random number 'simulates' characteristics of flows and must be repeated for each query or junction.	Characteristics are assigned to items by attributes and priorities which can then be tracked throughout the model.
Time steps	Interval between time steps is usually constant. Model recalculations are sequential and time dependant.	Interval between time steps is dependant on when events occur. Model only recalculates when events occur.
Ordering	Flows are in FIFO order.	Items can flow in FIFO, LIFO, priority, or customized order.
Routing	Flows need to be explicitly routed by being turned off at one branch and turned on at the other (flows can go to multiple places at the same time).	Items are automatically routed to the first available branch (items can only be in one place at a time).
Statistical detail	Only general statistics about the system: amount, efficiency, transit time.	In addition to general statistics, each item can be individually tracked: count, utilization, flow time.

Table 3.1: Continuous modelling versus discrete event modelling [Dia96].

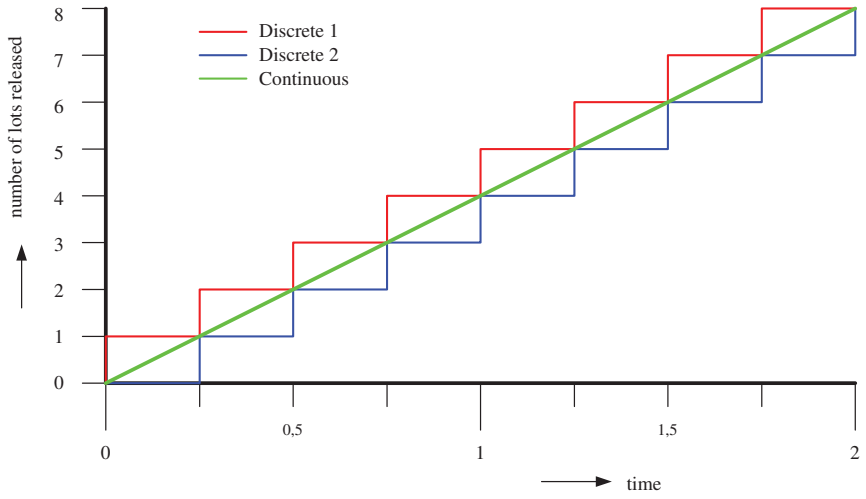


Figure 3.1: Different lot-releasing methods.

leased at time=0, 0.25, 0.5 and 0.75. Another possibility is to release the lots at time=0.25, 0.5, 0.75 and 1. In the first situation, the first machine can start producing at time=0. In the second situation, the first machine can only start at time=0.25. The two situations can be distinguished as the red and blue lines in Figure 3.1.

The tricky thing of both release policies is that the green line in Figure 3.1 represents both of them, since the green line has to cross the origin with a fixed angle. One of the policies has to be chosen for the discrete event model, since the continuous model for the first machine is different for both cases. It is obvious that this choice does not affect the validity of the model. The lot release policy corresponding with the blue line of Figure 3.1 has been chosen.

3.3 The continuous model of a GBMBME flowline

Consider a simple GBMBME flowline as shown in Figure 3.2. Assume that the flowline produces one product type. A discrete event model (χ, version 0.7) of this manufacturing system has been included in Appendix A.1. This model is assumed to be 'reality': the real manufacturing system. This system is modelled with a continuous approximation

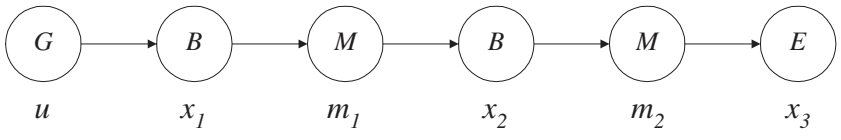


Figure 3.2: Iconic model of the GBMBME-flowline.

model. The rate in which parts are generated by generator G is u, the maximum rate in which the machines can handle products are m_1 and m_2 respectively. The buffer contents are x_1 and x_2. The amount of lots processed (finished products) is denoted by x_3. Note that x_1, x_2 and x_3 have to be non-negative and may have non-integer values.

The following modelling method has been used:

The production rate of a single machine is assumed to be constant. This rate equals its maximum if the value of the preceding buffer is positive. If the buffer is empty, the rate at which lots are processed is equal to the rate at which lots enter the buffer, provided that this rate is lower than the maximum production rate of the machine.

Unless stated differently, the time unit in this report is hours. So $u = 5$ means a rate u equal to 5 products/hour.

The process time of a machine is important in a production environment that is characterized as multi-process and multi-product (see [Cam01]). This means that a machine can only process a job when it has been finished on the previous machine. The buffer between two machines must therefore contain at least one product before the second machine can start with the product. On the other hand, this rule means that every buffer should contain at least one product permanently, to continue the flowline. But in times of machine-failure or termination of lot-releasing by the generator, the buffers may be emptied. In other words: a machine can produce if the preceding buffer contains at least one product and it can also produce if the buffer contains less than one product provided that the preceding machine is not producing.

The mathematical representation of the GBMBME flowline with maximum throughput policy is stated in equation set (3.1).

$$
\begin{array}{lcllcll}
\dot{x}_1 & = & u - m_{1\mathrm{r}} & \text{with} & m_{1\mathrm{r}} & = & m_1 \quad \text{for } x_1 \geq 1 \ \forall u \\
 & & & & m_{1\mathrm{r}} & = & m_1 \quad \text{for } 0 < x_1 < 1 \text{ and } u = 0 \\
 & & & & m_{1\mathrm{r}} & = & 0 \quad \text{for } 0 < x_1 < 1 \text{ and } u > 0 \\
\dot{x}_2 & = & m_{1\mathrm{r}} - m_{2\mathrm{r}} & \text{with} & m_{2\mathrm{r}} & = & m_2 \quad \text{for } x_2 \geq 1 \ \forall m_{1\mathrm{r}} \\
 & & & & m_{2\mathrm{r}} & = & m_2 \quad \text{for } 0 < x_2 < 1 \text{ and } m_{1\mathrm{r}} = 0 \\
 & & & & m_{2\mathrm{r}} & = & 0 \quad \text{for } 0 < x_2 < 1 \text{ and } m_{1\mathrm{r}} > 0 \\
\dot{x}_3 & = & m_{2\mathrm{r}} & & & &
\end{array}
\tag{3.1}
$$

A Simulink[1] model of this system is shown in Appendix A.1. To validate this model, it is compared with the discrete event model. First, the machine capacities are chosen in a way that the manufacturing line saturates ($u = 5$, $m_1 = 4$ and $m_2 = 3$). The results of the simulation are shown in Figure 3.3. The red circles are the output of the discrete event simulation of the model; lines represent the continuous approximation. Note that the discrete event output can only contain integer values and that a machine starts only when the preceding buffer contains at least one product.

[1]Simulink is a registered trademark of The MathWorks Inc. and part of their Matlab bundle.

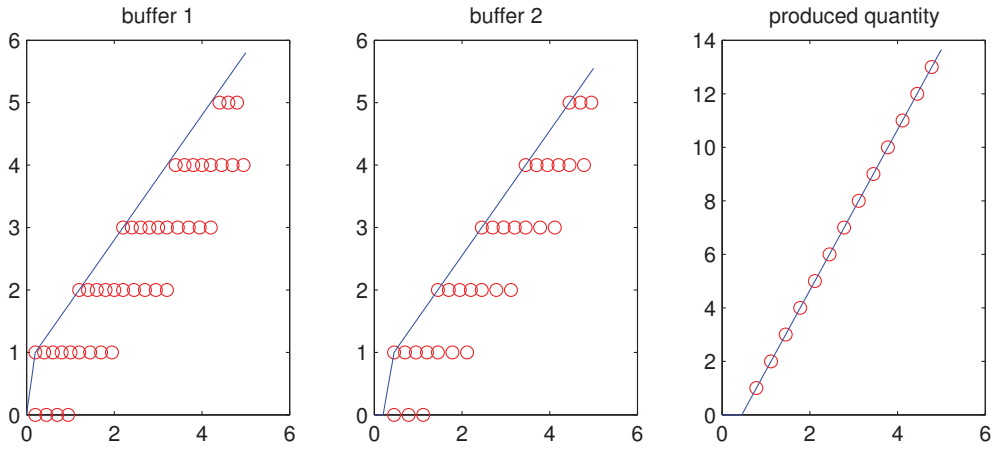


Figure 3.3: Simulation results of the saturating GBMBME-flowline.

When assigning the machine capacities in a way that the second machine is more productive than the first machine ($m_1 = 3$ and $m_2 = 4$), the results are not completely satisfactory (Figure 3.4). The time shift between the actual production (discrete event) and the continuous approximation is equal to the difference between the process times of the two machines. The second machine can only produce at the speed of the first machine, so according to the continuous model the first product is completed after 20 minutes on the second machine. In fact, the first product is completed after 15 minutes and then every 20 minutes on the second machine. If the flowline consists of more machines, the differences between the process times accumulate, resulting in a greater time shift. This only occurs when a machine further in line has a bigger capacity than a preceding machine. One remark about this situation that the experiment constantly switches between two dynamica due to the buffer level of buffer 2. It switches between $x_2 \geq 1$ and $x_2 < 1$. This 'chattering' needs to be handled with care by a numerical solver.

We can conclude that the continuous model performs better in a saturating flowline than in a non-saturating flowline. A practical flowline has both parts. But that is not a problem. When implementing the controller, an observer can be constructed. An observer typically denotes another dynamical system (in this case the continuous model) that is driven by the measurements of the system (the discrete event system), and which produces an estimate for the state [Nij00]. In case time-shifts are involved, the measurements of the discrete event system lead to a correction of the continuous state. Based on these corrections, new control output signals can be computed.

The former simulations were made with constant machine capacities and a constant input from the lot generator. But in the re-entrant flowshop, lots are most likely not released with a constant rate. Therefore, a new simulation is made using a piecewise linear input signal. The results are shown in Figure 3.5. The results of the discrete event

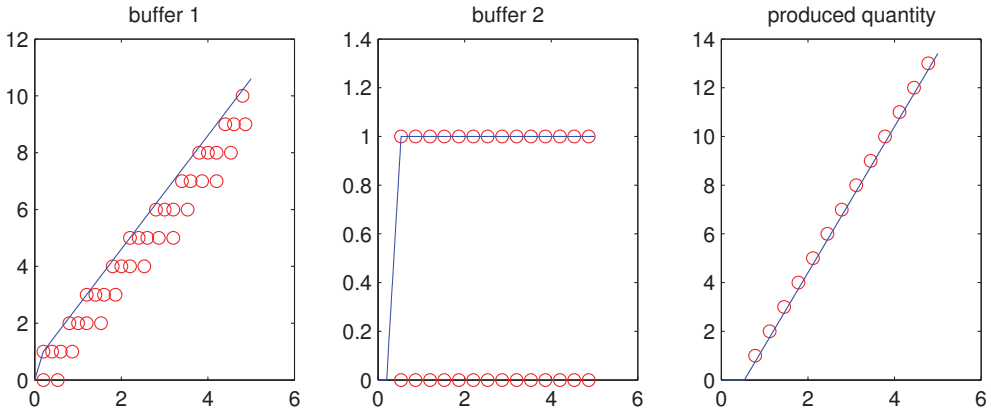


Figure 3.4: Simulation results of the non-saturating GBMBME-flowline.

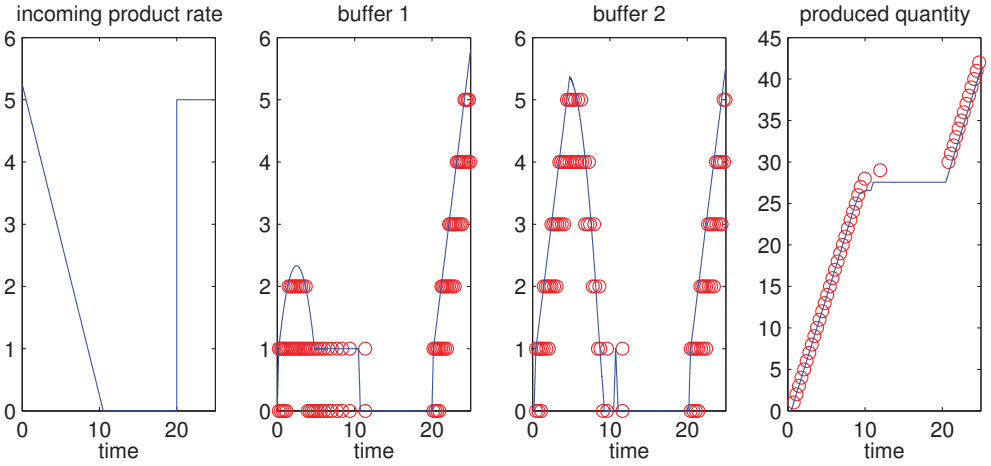


Figure 3.5: Simulation results of the GBMBME-flowline with piece-wise linear input.

system and the continuous model are not completely equivalent, but very acceptable, for reasons mentioned in the previous paragraph. The results are not elaborated on or discussed further here.

3.4 The continuous model of a GBMBME flowline with 2 product types

When more than one product type is produced on a single flowline, machine time has to be divided among the product types. Machines have finite capacity, which is here stated as the maximum production rates of the individual product types. The assignment of

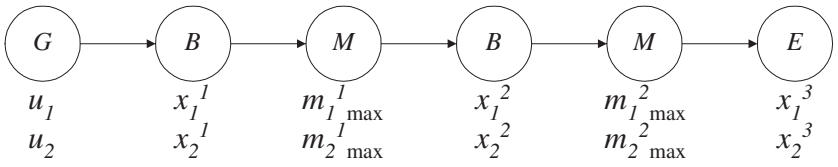


Figure 3.6: Iconic model of the GBMBME-flowline (2 products).

machine time to the product types will have to be determined by a controller. A controller may ask for more production than a machine can handle. When this occurs, the demand rate must be scaled in a way that the maximum production capacity is not exceeded. This must be done in both the discrete event system and the continuous approximation model. The scaling algorithm is discussed first here.

In general, define the maximum individual production rates of the product types for machine j as the vector $\underline{m}^j_{\max} = [m^j_{1,\max} \dots m^j_{n,\max}]^T$ (with $n = 2$ for 2 different product types). Assume that a controller desires the production rates of the product types to be $\underline{m}^j_{\text{des}} = [m^j_{1,\text{des}} \dots m^j_{n,\text{des}}]^T$. If the controller desires more production than the machines can handle, the desired production rates are proportionally scaled down towards manageable production rates. These scaled production rates are computed as:

$$\begin{aligned} \underline{m}^j_{\text{sc}} &= \frac{\underline{m}^j_{\text{des}}}{\max\left(1, \sum_{i=1}^{n} \frac{m^j_{i,\text{des}}}{m^j_{i,\max}}\right)} \\ \underline{m}^j_{\text{sc}} &= [m^j_{1,\text{sc}} \dots m^j_{n,\text{sc}}]^T. \end{aligned} \tag{3.2}$$

Note that, if the preceding buffer of product type i before machine j is empty, $m^j_{i,\max} := \min\left(m^{j-1}_{i,\text{sc}}, m^j_{i,\max}\right)$. The operator $:=$ means 'becomes'. Then, if $m^j_{i,\max}$ equals zero and $m^j_{i,\text{des}}$ is strictly positive, then $m^j_{i,\text{sc}}$ explodes. To prevent this, $\frac{m^j_{i,\text{des}}}{m^j_{i,\max}} := 0$ if $m^j_{i,\max} = 0$. Finally, the denominator of the fraction contains $\max(1, \dots)$ to avoid producing more than the desired quantity.

In the discrete event situation (Figure 3.6), machines constantly have to make the choice between producing product 1 or product 2. To make the 'right' choice, an integrator has been modelled. This integrator integrates the real (scaled) machines rates. The value of the integrator is the cumulative quantity of products that should have been produced. The discrete event machine counts the actual produced products. The largest (relative) difference between cumulative desired output and actual output determines which product to choose, provided that the buffer contains products of that type. If not, the second best option has to be chosen. For this purpose, the function `decision` was developed. This function first determines which product can be processed (difference between desired production and realized production is greater than zero and buffer

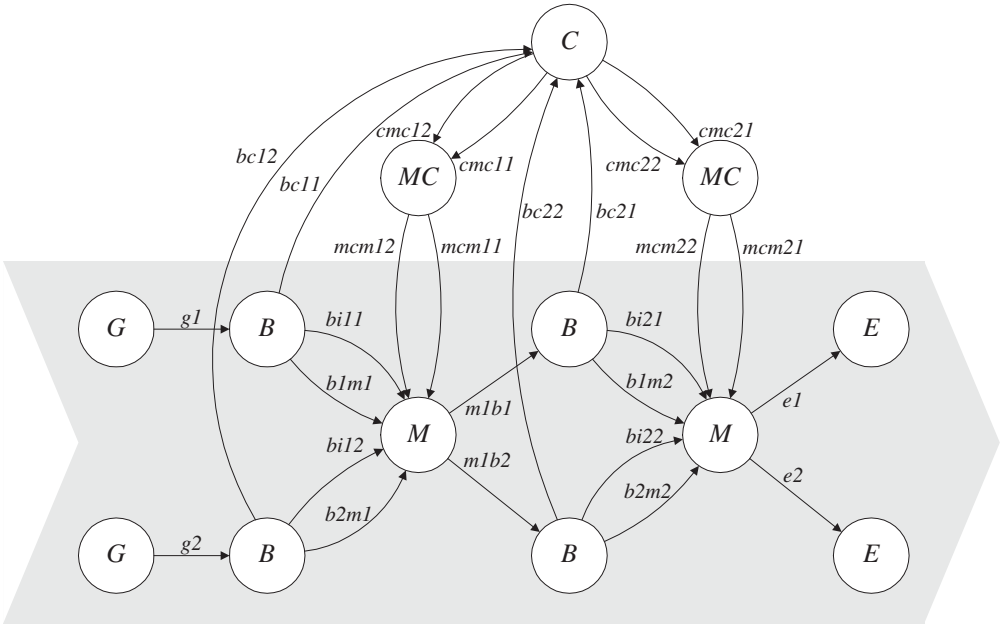


Figure 3.7: Iconic model of the discrete event model of the GBMBME flowline with two product types.

contents is greater than zero) and then chooses the product type with the greatest difference. The function `decision` is suitable to be implemented in systems with more than two products or larger systems with similar situations applying to a different number of products. That is the reason why lists are used instead of arrays. (A useful extension to the χ standard function library could be an array-to-list converter.)

```
func decision(diffs: real*, conts: nat*) -> nat =
|[ i, number, index: nat, record: real, c: bool
 | i:=1; number:=len(diffs); record:=0.0; index:=0
 ; *[ i <= number -> c:= hd(diffs) > 0 and hd(conts) > 0 and hd(diffs) > record
                   ; [ c     -> index:=i
                               ; record:=hd(diffs)
                     | not c -> skip
                     ]
                   ; diffs:=tl(diffs)
                   ; conts:=tl(conts)
                   ; i:=i+1
    ]
 ; ret index
]|
```

The complete iconic description of the discrete event model is shown in Figure 3.7.

The controller C in the discrete event model calculates the scaled production rates as in equation (3.2) plus the additional notes. The machine controllers MC calculate the

cumulative desired production that should have been realized and send this information to the machines to provide the necessary information to choose either product 1 or product 2. The χ-model is included in Appendix A.2.

For the continuous approximation model, the complete set of differential equations for the GBMBME flowline with 2 product types (Figure 3.6) is presented in equation set 3.3.

$$\begin{aligned}
\dot{x}_1^1 &= u_1 - m_{1,\mathrm{sc}}^1 \\
\dot{x}_2^1 &= u_2 - m_{2,\mathrm{sc}}^1 \\
\dot{x}_1^2 &= m_{1,\mathrm{sc}}^1 - m_{1,\mathrm{sc}}^2 \\
\dot{x}_2^2 &= m_{2,\mathrm{sc}}^1 - m_{2,\mathrm{sc}}^2 \\
\dot{x}_1^3 &= m_{1,\mathrm{sc}}^2 \\
\dot{x}_2^3 &= m_{2,\mathrm{sc}}^2.
\end{aligned} \tag{3.3}$$

A Simulink model of this system is shown in Appendix A.2.

Simulation results for constant inputs u_1 and u_2 and constant desired machine capacities are shown in Figure 3.8 ($u_1 = 5$, $u_2 = 6$, $m_{1,max}^1 = 4$, $m_{2,max}^1 = 3$, $m_{1,max}^2 = 5$, $m_{2,max}^2 = 2$, $m_{1,des}^1 = 8$, $m_{2,des}^1 = 2$, $m_{1,des}^2 = 3$, $m_{2,des}^2 = 6$). The figure shows that the red circles and the blue line do not coincide exactly anymore. The reason is that a machine has to choose which product is to be made, while the continuous model can process the two products in parallel. But overall, the picture shows a good equivalence between continuous and discrete event model. The somewhat strange circles in Figure 3.8 for buffer 2, product 2, are most probably caused by the non-deterministic character of the discrete event simulator. If two events can take place at the same time, the order in which the events take actually place remains undetermined. This explains that at time=4.5 and time=8, the buffer level becomes 2: the buffer is filled before it is emptied.

As in the GBMBME case with one product, the model with two product types has been tested again with a piecewise linear input signal. The results are shown in Figure 3.9. The results indicate that the model handles the occurring phenomena very well and is not discussed or elaborated on further here. Again, the two circles at buffer level 2 for product 2, buffer 2, are most likely caused by the non-deterministic character of the discrete event simulator.

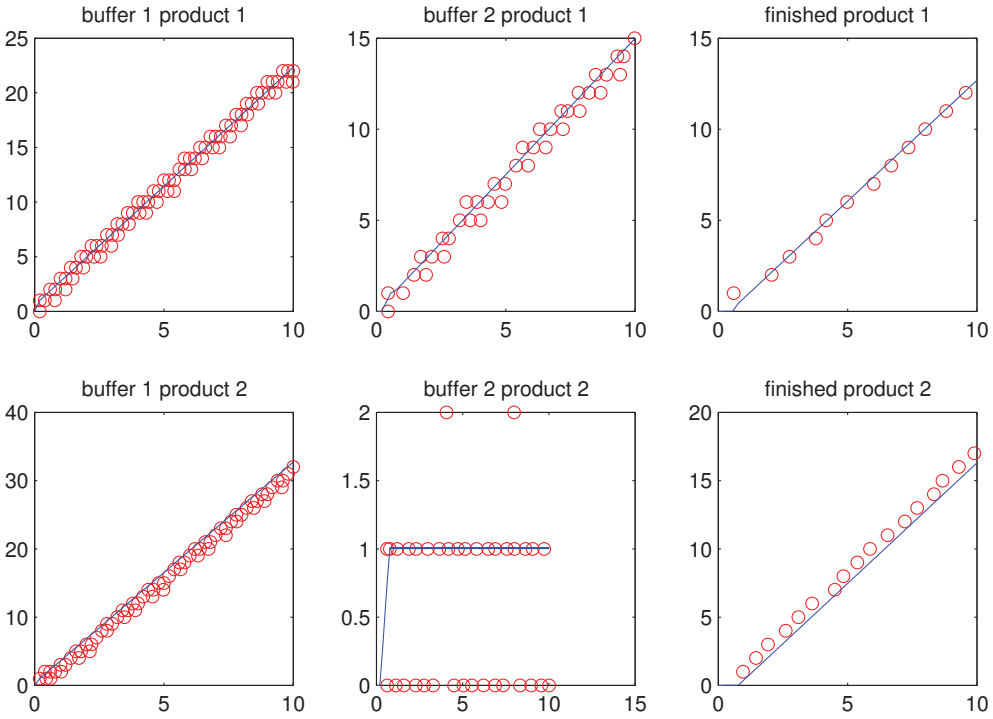


Figure 3.8: Simulation results of the GBMBME-flowline (2 products).

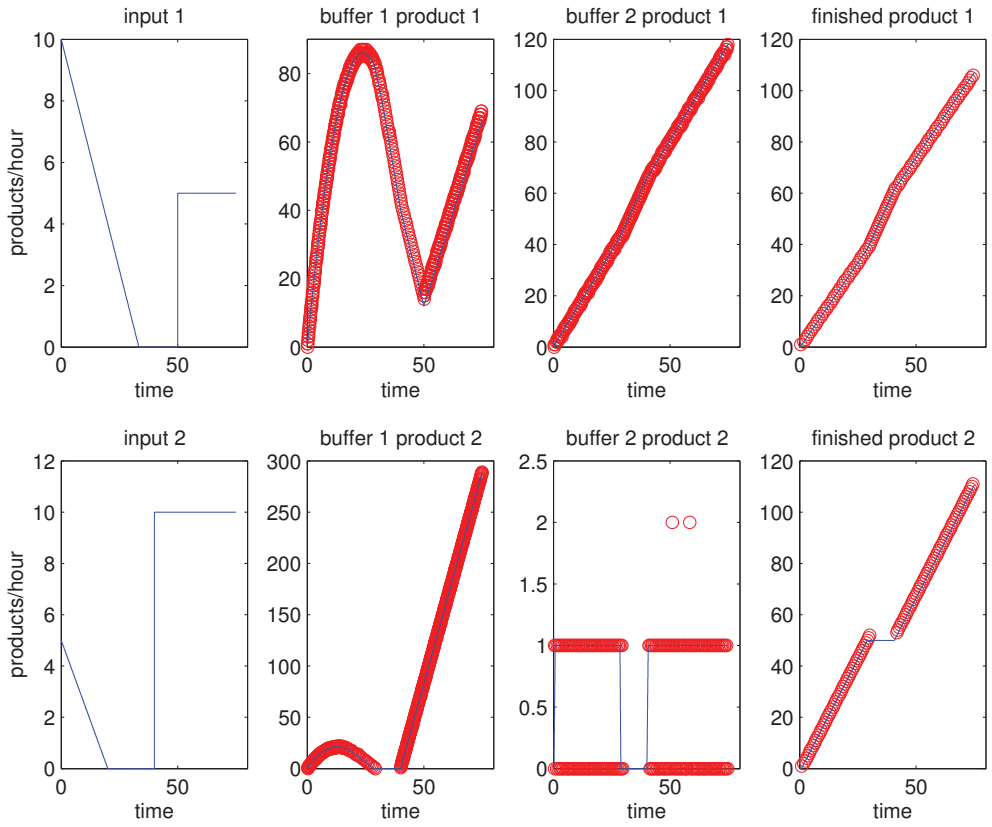


Figure 3.9: Simulation results of the GBMBME flowline with two product types and piecewise linear input signals.

3.5 Reflections on the hybrid model

The model evolved in this chapter is a so called 'hybrid model'. The hybrid model consists of a number of (continuous state) dynamics and switching between dynamics is based on events. An event may cause the model to switch from the one $\dot{x} = B_n u$ model to the other $\dot{x} = B_m u$ model. Within one dynamics, the model is perfectly continuous. Overall, a piecewise continuous state emerges.

The number of dynamics increases exponentially when the system is enlarged. Depending on the machine's and buffer's states, a machine can take three possible production rates: zero production, maximum production rate or somewhere in between due to (scaling of) desired production rates. Scaling only takes place if the future controller desires more production than the machines can handle.

The model described in this chapter is very basic but shows promising results on modelling discrete event systems with a hybrid model.

In future research, this model can be elaborated thoroughly. One could think of modelling buffers with finite capacity or batch machines. The first can be implemented by not only setting a lower limit for buffers, but also an upper limit, the latter by increasing the threshold level for production (which has been set to 1 now). Another interesting topic is a manufacturing line with assembly or disassembly. If these topics are explored, a wide range of manufacturing systems can be explored.

In the next chapter, the new simple hybrid model is going to be applied to a case study. The method is used to get a continuous model of a re-entrant flowshop with multiple products and recipes.

Chapter 4

Modelling a re-entrant flow-shop

4.1 General

A re-entrant flow-shop is modelled as a central buffer with surrounding machines. A generator sends lots to the central buffer. This buffer sends lots to the different machines, according to their recipe. The central buffer is not an ordinary n-place buffer. Actually, it is a collection of buffers, modelled as one. If the flow-shop produces 2 different product types with each 3 production steps, then the central buffer consists of 6 separate buffers. When a product is finished, it is sent to the exit process. An iconic model of a flowline with re-entrant behavior is displayed in Figure 4.1.

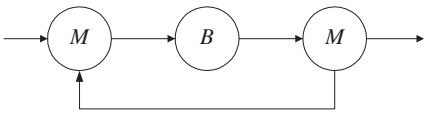


Figure 4.1: Schematic iconic representation of re-entrant behavior.

The exit process does not have to be modelled separately. In fact, it is a storage of finished products and products are only accumulated. This can be seen as a buffer with only incoming products. Instead of an exit process, products can be kept in the central buffer. The number of buffers is thus increased with the number of different product types. In general, the number of subbuffers in the central buffer, N, can be computed as in Equation 4.1 (with n the number of product types and p_i the number of production steps for product type i).

$$N = n + \sum_{i=1}^{n} p_i \tag{4.1}$$

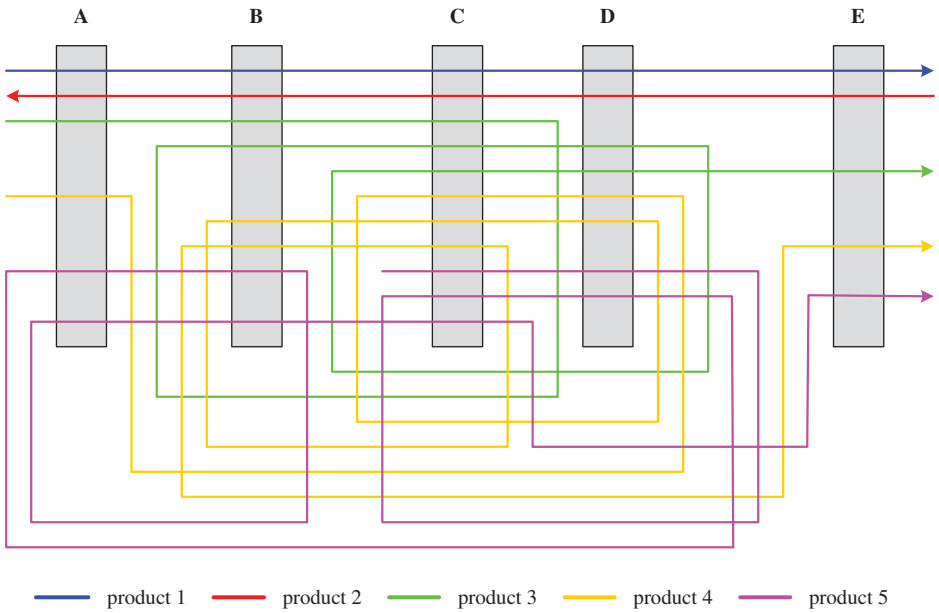


Figure 4.2: Schematic route of the 5 different product types.

4.2 Case study: A specific re-entrant flow-shop

As a case study, a non-realistic re-entrant flow-shop has been modelled. It produces 5 different products on 5 different machines. Each product has its own recipe. A picture of the product routes is displayed in Figure 4.2. Machines are displayed as the grey boxes $A \ldots E$. An important property of this type of production facilities is that no assembly or disassembly takes place.

Product type 1 is processed on the machines as if it were a simple flowline (Section 3.3). Product 2 travels the same flowline in opposite direction. Product types 3, 4 and 5 show re-entrant behavior (products are processed on a machine more than one time). The extreme recipes (type 1 and type 2 in opposite direction) have been chosen to test the behavior of the model. Banks and Dai show in [BD97] that in some cases this configuration may become unstable.

The way the product routes are displayed is rather inconvenient. Another way to visualize the routes is to display them as virtual flowlines (Figure 4.3). This is a convenient way to visualize the recipes, since no assembly or disassembly of parts takes place. One should notice that all processes on the same machine (same color in the figure) share the capacity of that machine (only one machine A (blue) exists instead of 6 machines).

To model this flow-shop as a continuous model or a discrete event model, the *state* of the system has to be defined. This state is defined as the column of individual buffer contents in the central buffer. The buffer contents are sorted per machine and then per

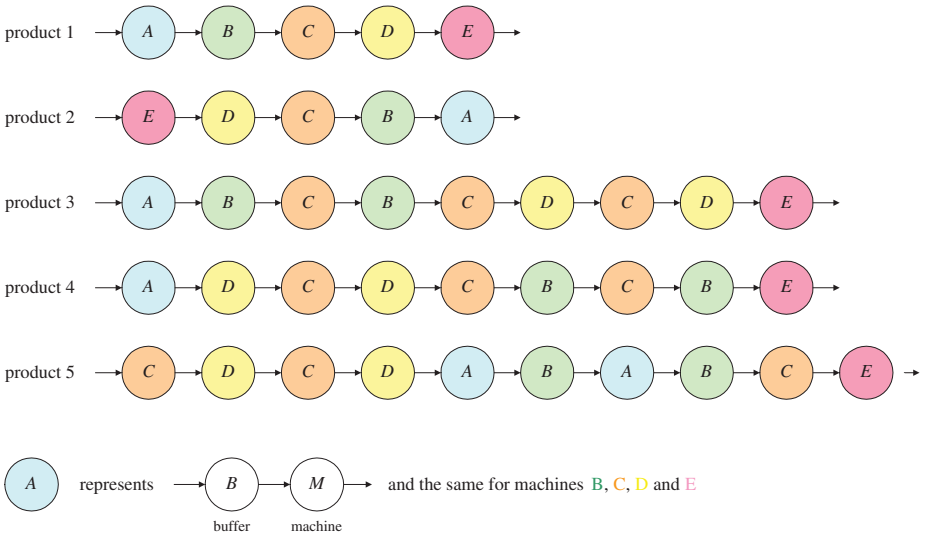


Figure 4.3: Virtual Flowlines of the 5 products.

recipe. The state $\underline{x}$ is thus defined by:

$$\underline{x} = \left(\underline{x}_{\mathrm{A}} \quad \underline{x}_{\mathrm{B}} \quad \underline{x}_{\mathrm{C}} \quad \underline{x}_{\mathrm{D}} \quad \underline{x}_{\mathrm{E}} \quad \underline{x}_{\mathrm{Finished}}\right)^T$$

$$\begin{aligned}
\underline{x}_{\mathrm{A}} &= \left(x_1^1 \quad x_5^2 \quad x_1^3 \quad x_1^4 \quad x_5^5 \quad x_7^5\right) \\
\underline{x}_{\mathrm{B}} &= \left(x_2^1 \quad x_4^2 \quad x_2^3 \quad x_4^3 \quad x_6^4 \quad x_8^4 \quad x_6^5 \quad x_8^5\right) \\
\underline{x}_{\mathrm{C}} &= \left(x_3^1 \quad x_3^2 \quad x_3^3 \quad x_5^3 \quad x_7^3 \quad x_3^4 \quad x_5^4 \quad x_7^4 \quad x_1^5 \quad x_3^5 \quad x_9^5\right) \\
\underline{x}_{\mathrm{D}} &= \left(x_4^1 \quad x_2^2 \quad x_6^3 \quad x_8^3 \quad x_2^4 \quad x_4^4 \quad x_2^5 \quad x_4^5\right) \\
\underline{x}_{\mathrm{E}} &= \left(x_5^1 \quad x_1^2 \quad x_9^3 \quad x_9^4 \quad x_{10}^5\right) \\
\underline{x}_{\mathrm{Finished}} &= \left(x_6^1 \quad x_6^2 \quad x_{10}^3 \quad x_{10}^4 \quad x_{11}^5\right).
\end{aligned} \tag{4.2}$$

The state contains the 38 production steps of the total system together with the 5 storage buffers for finished products. The state contains 43 elements. Individual elements are denoted by x_i^j, with i and j representing the production step of the product and the product type respectively.

4.3 The continuous model

4.3.1 Simulink model

The Simulink continuous model of the re-entrant flowshop has the same structure as the ones of the flowlines, but more elaborate. The complete model is shown in Figure 4.4. The five machines (details in Figure 4.5) can be depicted clearly. The five incoming product rates have been grouped in the upper left corner of the diagram. The cobweb in the center of the diagram is the connection between buffers and machines according to the recipe. Due to the complex structure, this Simulink model cannot be adjusted in an easy way to use it for different products and product recipes. The selector in the lower right corner regroups the real (scaled) machine rates to be fed back into the machines as the scaled production rates of the previous machine.

Simulink encounters an algebraic loop in the model and as a result difficulties to solve the differential equations. The algebraic loop is a direct result of the loop products have to make in the system. The production rate of a product is a function of the production rate of the previous machine, which in turn is a function of its own previous machine rate, which can be the first machine mentioned, because of the re-entrant behavior. This numerical difficulty was first solved by inserting a time-delay of a single timestep in the feedback of the machine rates, but this does not work properly in case the machines have to scale down the desired production rates very much. Matlab itself provides a different solution for an algebraic loop in its documentation: inserting a 'Data Memory' block in the system. This (partial picture shown in Figure 4.6) gives a suitable solution for the algebraic loop numerical problem.

In testing the Simulink model, another problem arose. The discontinuities in the model make the production rates highly discontinuous, resulting in false triggering of the production rate computation algorithm. A buffer can be emptied, as soon as the preceding buffer contains one product, or less than one product provided the preceding machine to have production rate zero. The discontinuities make this production rate to be maximum-zero-maximum-zero-maximum-zero, etc. So every two steps, the buffer is emptied. This (mal)behavior has been eliminated by introducing a sort of memory function: a machine is only allowed to produce if the buffer contains at least one product or less than one product, provided the preceding machine production rate to be zero for a few timesteps (5 or 10, this can be adjusted). This was implemented using a discrete state space function $x(t+1) = Ax(t) + Bu(t)$ and $y(t) = Cx(t) + Du(t)$ with for the matrices $A, \ldots, D$:

$$A = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ 1 & \ddots & & & \vdots \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix}, \qquad C = I, \qquad D = 0.$$

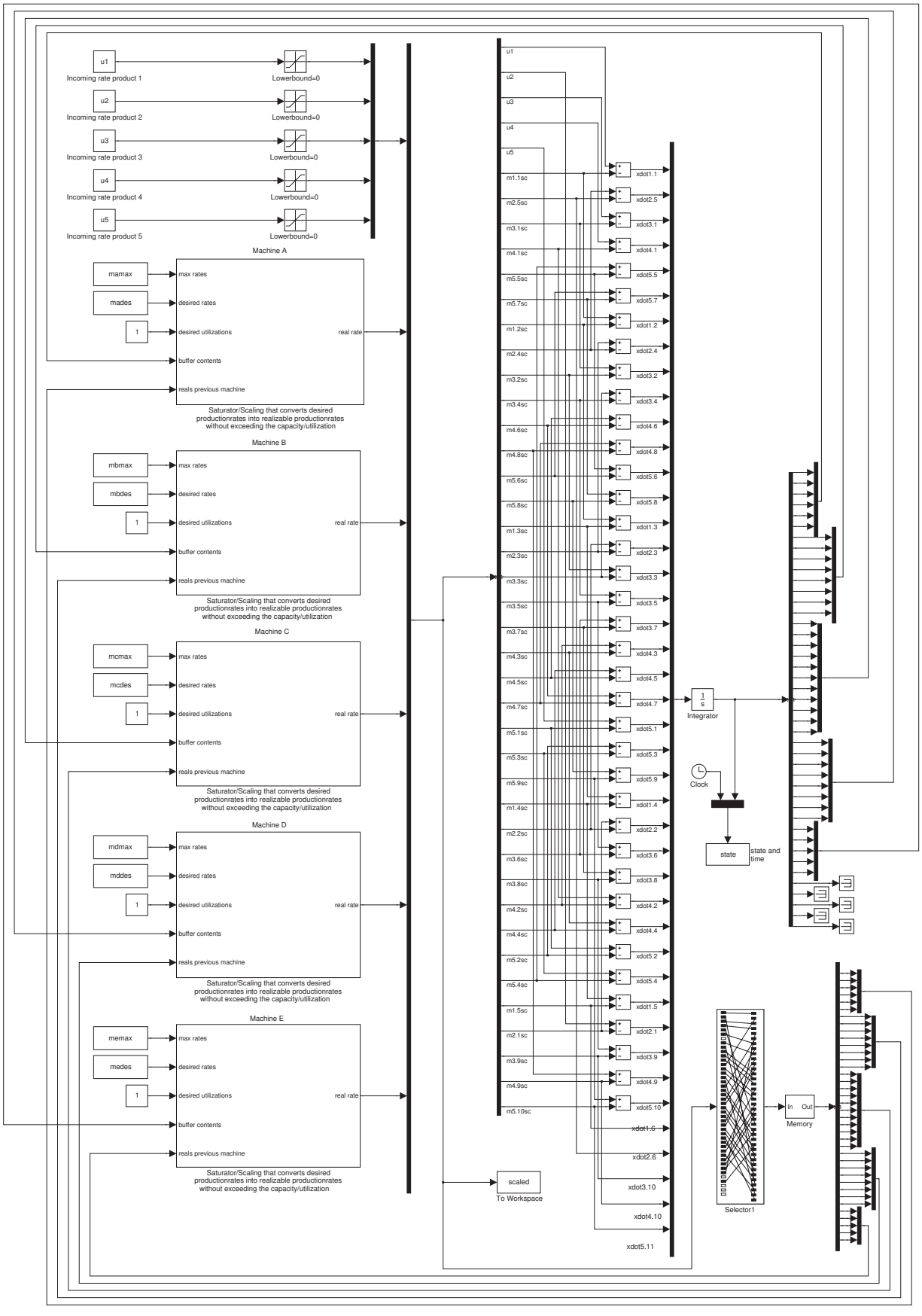


Figure 4.4: Simulink model of the re-entrant flow-shop.

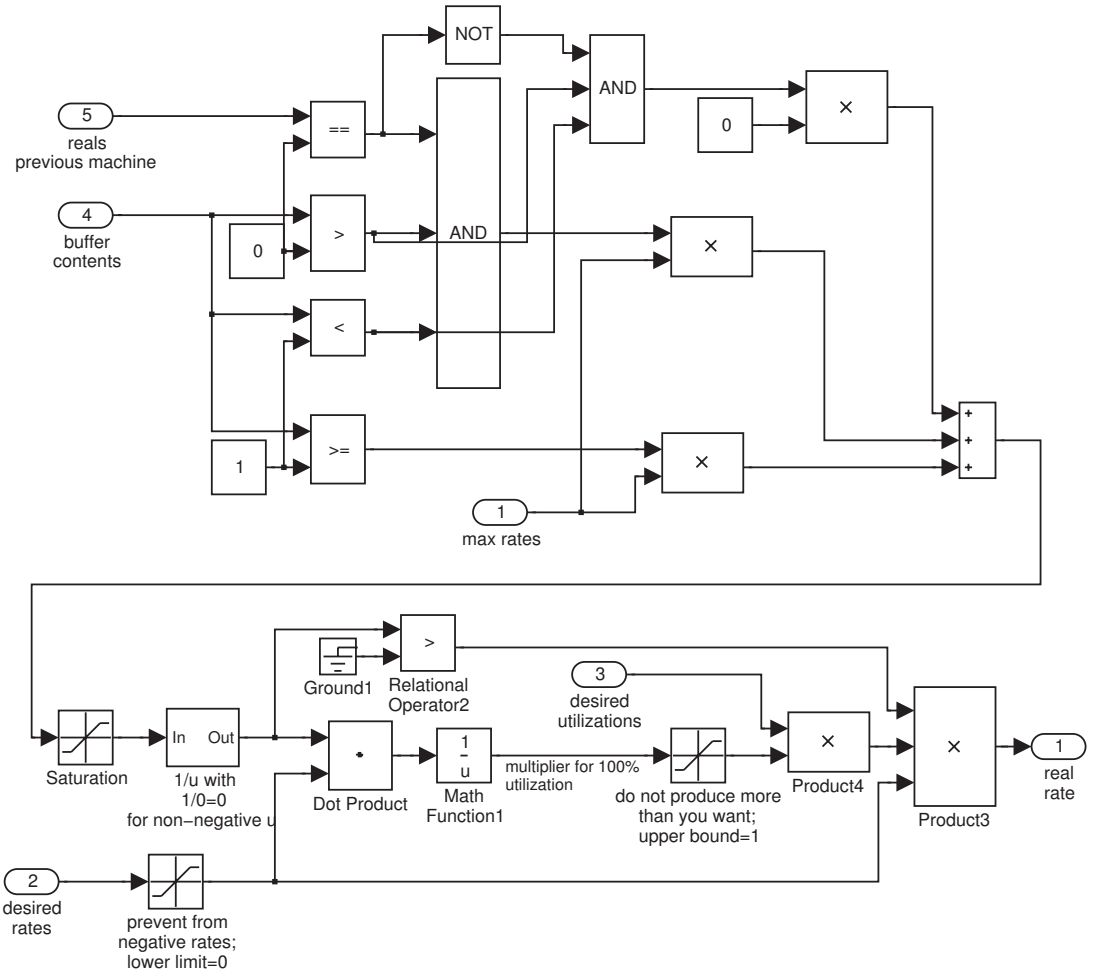


Figure 4.5: Simulink model of the re-entrant flow-shop, magnification of the Machine-block.

Matrix A is a $n \times n$ matrix. n is the number of timesteps to be monitored. If $y(t)^2 = 0$, the machine is allowed to produce. This means that for the last n timesteps, the preceding machine production rate has been equal to zero. The allowance to produce is stored as a boolean variable. Note that this work-around leads to additional model errors, because production starts slightly later than it should start.

The results become with this Simulink model however were not satisfactory. In some specific situations (like a constant input on the fifth product) the model produced little inaccuracies, which (accumulating) resulted in large errors in the output, as can be seen in Figure 4.7. Production looks well until the fifth production step, but zooming in strongly on the first graphs already shows small errors. The explanation for these errors lies in the order in which Simulink computes all variables. Before computing the scaled production rates, the booleans whether a machine is allowed to produce has to be

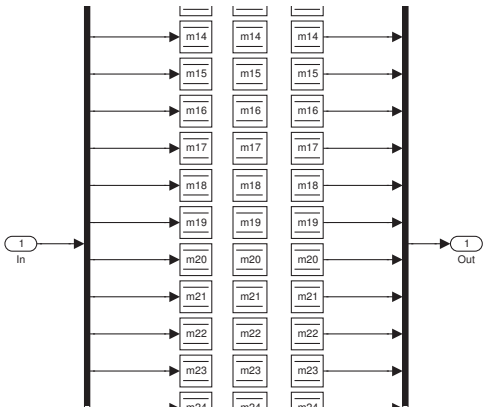


Figure 4.6: Simulink model of the re-entrant flow-shop, partial magnification of the Memory-block.

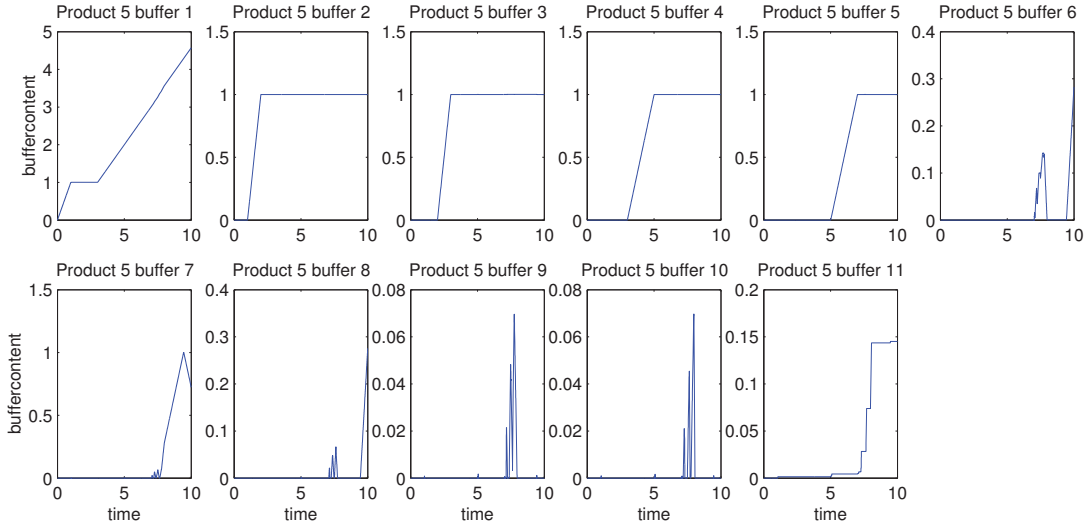


Figure 4.7: Accumulating errors in the Simulink model for product 5.

computed. After those computations, a new integration step can be made. The order in which Simulink computes all blocks can be set with priority statements. If these priorities interfere with the pre-defined priorities Simulink has been programmed with, Simulink uses its own priority rules. Therefore, in practice this function is only usable in cases where the blocks to be computed have equivalent priority status according to Simulink's rules. In the model of the re-entrant flowshop, this seemed not to be the case. Therefore, a radical step was made: abandon Simulink and program the complete flowshop in Matlab m-files.

4.3.2 Matlab model

The standard ODE solvers implemented in Matlab (Runge-Kutta, Adams, Rosenbrock) do not accept a minimum step size. An integration method with a minimum or fixed step size is needed, to step over the discontinuities in the model. Therefore, an external, user programmed integration method was looked for and found at the University of Texas at Austin [Lon98]. This script uses a fixed-step Runge-Kutta 4 method with integrated Simpson's rule. Due to the discontinuities and the unknown buffer states at next time steps, the 6 derivatives to be computed are all the same. The Runge-Kutta script therefore reduces to the trapezoid rule for integration. This integration scheme does not contain an error-correction algorithm, so errors can diverge easily. With linear input signals and small timesteps, comparisons between continuous and discrete event simulations are made in Section 4.5. Under these two circumstances, a trapezoid rule will hold.

All functions and blocks from Simulink have been reprogrammed in Matlab. The great advantage of this method is the controllability of the script. Moreover, the new product recipes of machines can be implemented a lot easier than in the Simulink model. The user decides which computations have to be made in which order. The complete (modified for the flowshop) Matlab script of this program is shown in Appendix B.

In each step of the loop, computations take place in the following order:

1. the buffer contents for the individual machines are extracted from the state;
2. the real machine production rates vector of the previous timestep are transformed to obtain the vector of production rates of the preceding machines;
3. the booleans which indicate whether or not a machine is allowed to produce are determined;
4. the real (scaled) production rates for the current timestep are computed;
5. the new derivatives of the state are computed;
6. using the trapezoidal rule, the state is integrated;
7. all computed values are stored for postprocessing and the loop starts over again.

To compare the Matlab model to the Simulink model, the same experiment as in Figure 4.7 has been carried out. The results are shown in Figure 4.8 ($u5 = 1$, $maxrate5 = 1$, $desrate5 = 5$). Contrary to what one might expect, the picture shows that the last machine starts to produce at $t = 18$, although the system has only 10 production steps. This is correct, because some machines have to share their capacity, as can be seen in Figure 4.3. In each re-entrant step, the product flow is slowed down. And as the throughput is determined by the slowest link, all following production steps are slowed down. In the discrete event situation however, the first product will have been finished a lot earlier, as is explained in the following section.

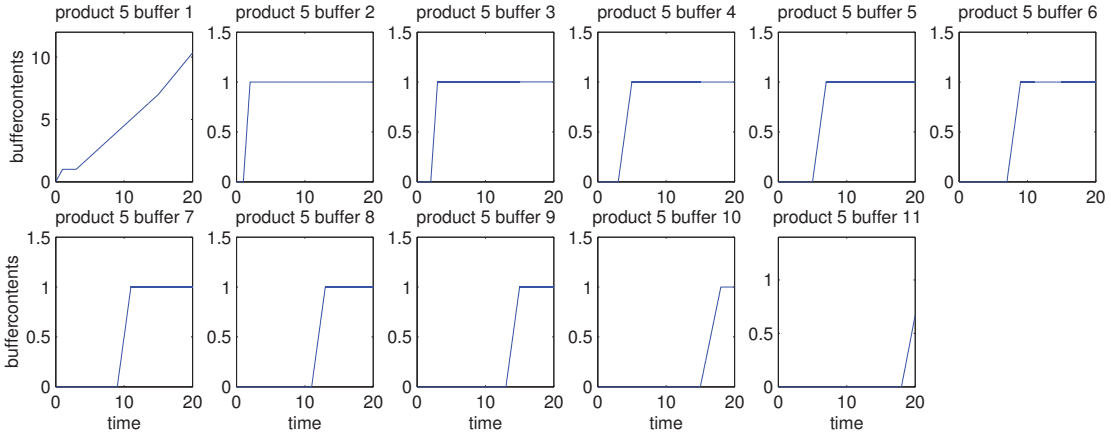


Figure 4.8: Product 5 simulation using the Matlab continuous model.

4.4 The discrete event model

4.4.1 Model structure

The discrete event model of the reentrant flowline with 5 products and 5 machines has been modelled in χ version 0.7. To be able to simulate other configurations as well, a universal model has been built. The iconic representation is shown in Figure 4.9; the explanation of the processes and the channels is given in Table 4.1. As can be seen, the number of different products and their recipes do not affect the model structure. The number of machines however does affect the model structure, but this number turns out to be a variable.

Process		Channel	
G	Generator	cd	communicate desired machinerates
M	Machine	gi	get info about the product to be made
C	Controller	ig	input lots from generator to buffer
IO	Communication with Matlab	im	input lots from machine to buffer
B	Buffer	mr	send maximum rates to controller pocess
BC	Buffer contents process	mu	update machine status to controller
		om	output lots from buffer to machine
		rg	request buffer contents from generator
		rl	request buffer contents from controller
		ul	update length of buffers

Table 4.1: List of symbols χ-model.

A single product (lot) has been modelled as a tuple: $<id$, *timestamp*, *product type*, $age>$ The *id* is a unique number for the product type, an auto-incrementing natural number. The *timestamp* is the time a lot is released from the generator. *Product type* is a natural number representing the type of the product $(0 \ldots numberofproducts - 1)$. *Age* is the number of production steps the lot has completed, a natural number ascending from

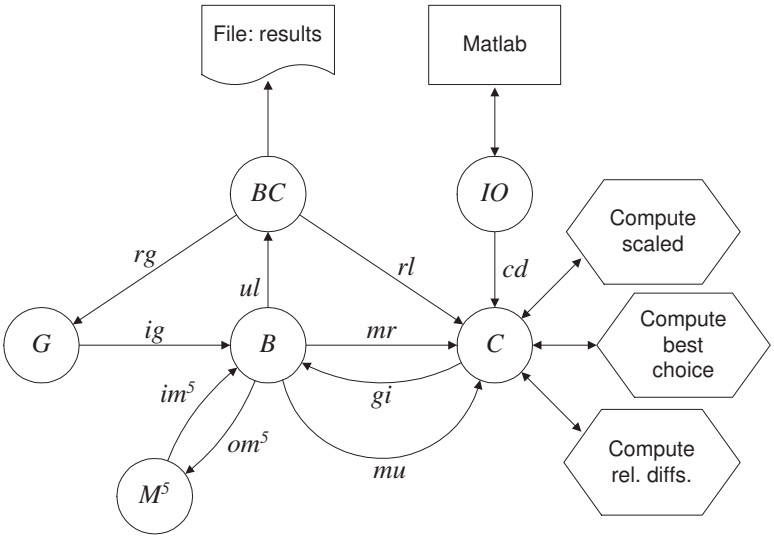


Figure 4.9: Iconic model of the reentrant flowshop.

zero.

The generator G sends lots to the buffer, which sorts the lots according to their product type. The central buffer B consists of 43 separate buffers (lists) in which each different product phase is stored, including finished products. The buffer sends lots to the machines and receives them back. Which product is to be sent, is decided by the controller C.

All machines M are identical. Products are sent to them together with the process time. These process times are stored in a tuple in the buffer process and are the reciprocals of the maximum machine rates of the individual products (tuple *maxrates*).

Every time an event takes place, the actual length of the changing buffer is communicated to the buffer contents process BC. This process writes to the output file. On each event, the time and all buffer contents are written to the output file. The buffer contents process also communicates the updated buffer contents to the controller and the generator.

The generator fills the first buffer of a manufacturing line with raw materials, so that it can not run out of materials. The quantity of the incoming material flow is not important, because it can be set equal to the production rate of the first machine. If bigger, a stock is created. If smaller, the line starves. To get the same behavior in the discrete event model as in the continuous model, the incoming material flow has been decoupled from the production rate of the first machine and can be set separately.

The discrete event model is prepared for communicating with other programs, like Matlab or Python. For this reason, the process IO exists. This process communicates with the outside to obtain new desired production rates. Since it is still unknown what

information the continuous controller needs, new communication channels might have to be implemented in the future. For now, the process IO contains the values of the desired production rates and does not communicate with the outside.

The controller is the most complex part of the discrete event model. Based on the desired production rates and the status of all machines and buffers, it has to decide which product must be processed when a machine becomes idle. To do this, on each event the controller computes the *best choice* for each machine. This is the list of products the machines should process if they were idle. When a machine becomes idle, the *best choice* for that particular machine is sent to the buffer and the buffer in turn sends a lot of the specific product type to the machine, according to the FIFO-principle (all products in a separate buffer are the same, so the longest waiting lot is processed).

But how does the controller compute this *best choice*? Every (fixed) timestep, the discrete event controller gets the new desired production rates from the continuous controller (i.e. Matlab, via process IO). These desired rates are scaled with the function *compute scaled*. The scaled rates are integrated to compute the amount of products that should have been completed. Next, the controller computes the relative difference (function *compute rel diffs*) between desired output and actual output of all products (at every age). The product with the biggest, but positive, relative difference for a single machine gets the label 'best choice' (function *compute best choice*). Every time an event takes place, this best choice is recomputed.

4.4.2 Matlab GUI

To facilitate the use of the universal reentrant model, a Graphical User Interface (GUI) was written in Matlab (see Figure 4.10). This GUI can be used to enter all properties of the reentrant flowshop. By clicking the 'generate chi-file'-button, the specified χ-file is generated. All necessary tuples are computed in Matlab and written in the header of the file generated. This has been done to allow for easy manually editing of the chi-file. A sample output (for the specific 5 products, 5 machines flowshop) is provided in Appendix C.

4.5 Comparing the continuous and discrete event models

Although most of the presented work in this report so far is about continuous modelling, the continuous model should be a representation of the discrete event model. Therefore, a comparison has been made between the two models.

First, consider only the first product with incoming material flow rate $u1 = 3$, the maximum individual production rates of the machines are $maxrate1 = 1$, and the desired production rates $desrate1 = 5$. The results of both simulations are shown in Figure 4.11. Again, the blue lines are the results of the continuous model, the red dots are the results

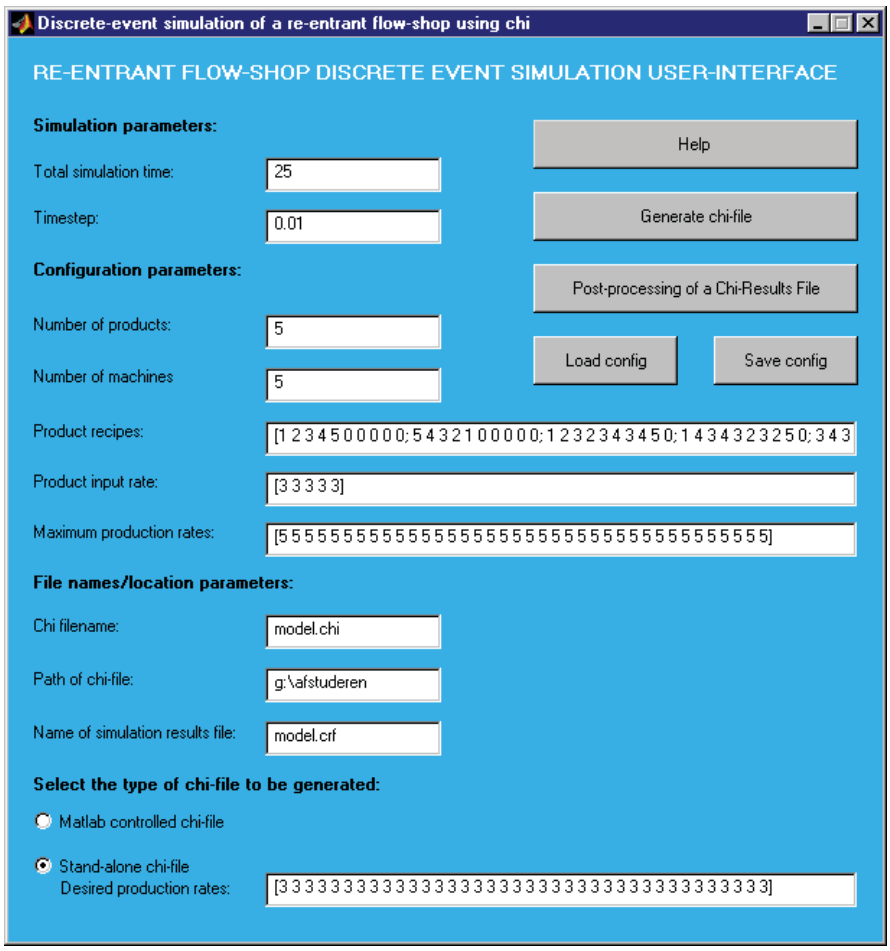


Figure 4.10: Matlab GUI to generate χ-files.

of the discrete event model. The picture shows very good equivalence of the continuous and discrete event model.

Product 1 represents a simple flowline. Product 3 however shows re-entrant behavior. Therefore, a test run for product 3 alone has been made ($u3 = 3$, $maxrate3 = 3$, $desrate3 = 1$). The results are shown in Figure 4.12. The equivalence is not very good. The discrete event model (reality!) has finished its products a lot earlier than the continuous approximation. The reason for this is the fact that the desired production rates are below the maximum production rates (machine C has utilization 1 in this configuration and with this settings). As soon as a machine is allowed to produce (preceding buffer > 1) the machine starts producing with the desired production rate. In reality, the machine starts to produce and finishes it at the maximum production rate. The following machine can start earlier on its turn and so on. As the desired production rate is three times smaller than the maximum production rate, the first completely fin-

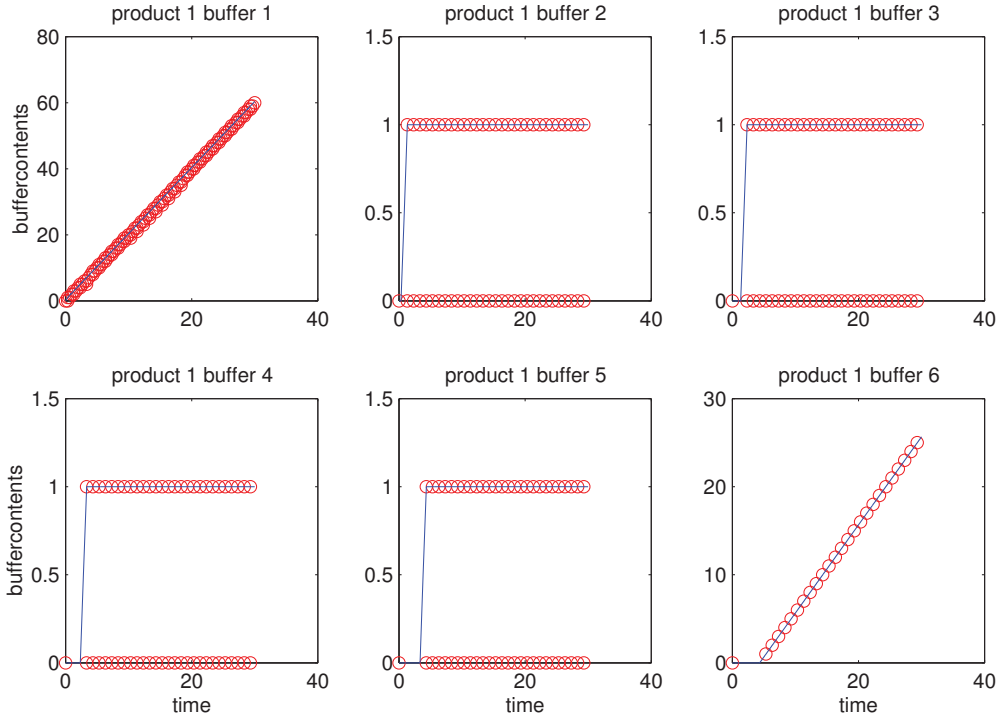


Figure 4.11: Comparison continuous vs. discrete event model for product 1.

ished product leaves the manufacturing system three times faster in the discrete event situation than in the continuous approximation.

Although this continuous model does not predict the state of the discrete event system very accurately, it can be very useful in a feed-back situation. Measurements of the state can be used to correct the state. This observer method has been explained in Section 3.3. The dynamics of the system have been modelled quite well (no buffers become negative, a machine starts only to produce if the preceding buffer contains one product). Although presented here only for product 1 and product 3, the model behaves similar for the other product types and in combination with each other.

4.6 Reflections on the re-entrant flowshop case study

The hybrid model developed in Chapter 3 has been used to specify a re-entrant flowshop with 5 machines and 5 product types, each having its own recipe. Some good results have been obtained. The re-entrant flowshop can be modelled with the hybrid model method. In Chapter 3, it had been stated that the number of dynamics diverges exponentially with an increasing number of machines and buffers. This has been overcome by constructing the B-matrix in the $\dot{x} = Bu$ model every timestep instead of choosing

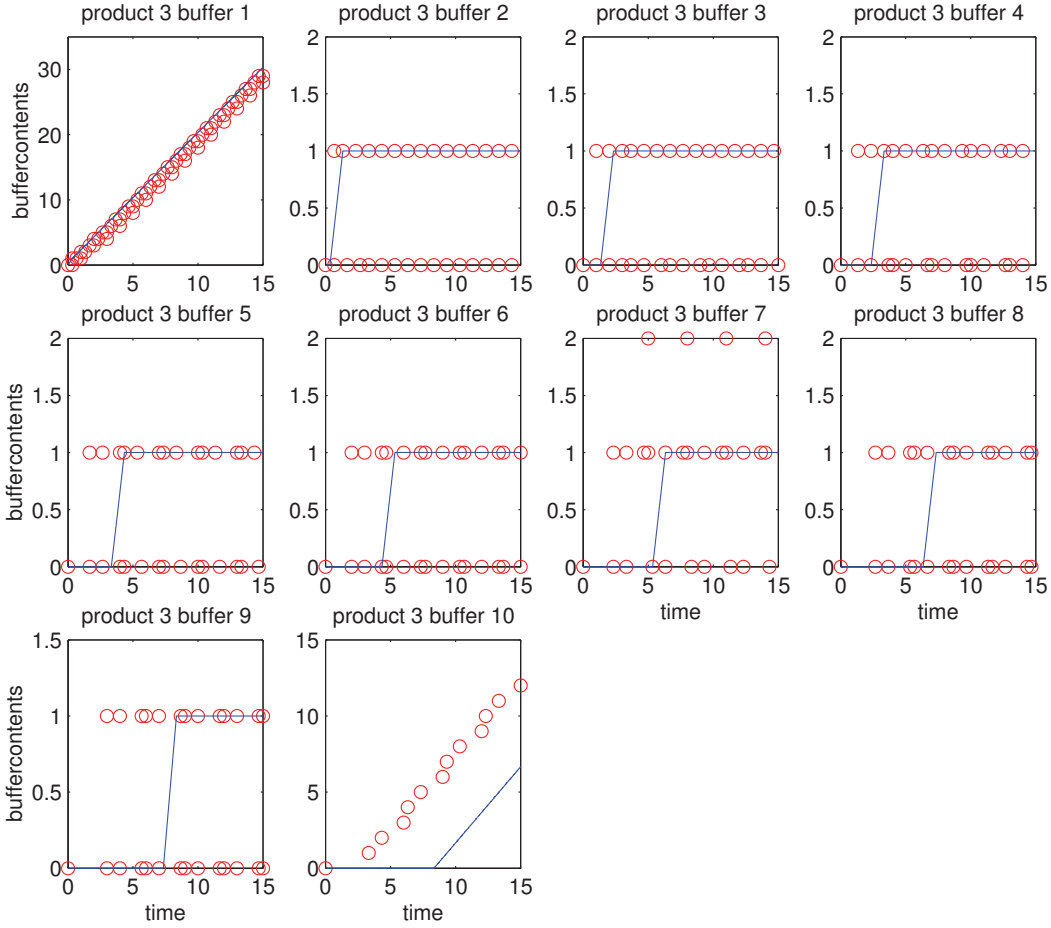


Figure 4.12: Comparison continuous vs. discrete event model for product 3.

the right B-matrix from a predefined library of all possible B-matrices. This library would be too large to implement and to construct.

The Simulink models have been very helpful in understanding the dynamical behavior of the manufacturing systems that have been investigated. Due to numerical problems that arose and the inflexibility of Simulink, the complete flowshop has been reprogrammed in Matlab. This relieved some of the numerical problems, but the need for a well-considered integration method exists.

The hybrid method describes the dynamics of the manufacturing systems quite well, but the time shifts involved due to parallel processing and producing far below capacity make the model not a good prediction model. Moreover, the hybrid model can only be used in a small number of control techniques, because of the switching dynamics. Because of this restriction, different modelling methods are investigated in the following

chapter(s). The goal is to obtain a state space description of the manufacturing systems in which the process times of machines are included into the dynamics of the state space model instead of the switching dynamics.

Chapter 5

Identification of discrete event systems

5.1 General

As stated before, the manufacturing systems in this research are presented as discrete event systems. The continuous approximations described in Chapter 3 and 4 can not be used in most (linear) classic control, because most control techniques require only one state space model. A state space model is a linear model. A manufacturing system is assumed to behave like a linear system, since it consists only of buffers and machines: the outflow of the one buffer is the inflow of the other buffer. The process time is assumed to be approximated by a linear model. Hence, instead of implementing process times of machines in a hybrid model, the process time has to be included in the dynamics of one state space model. A useful technique to estimate a state space model is system identification. System identification is a technique to build mathematical models of a dynamic system based on measured data. In this research, the System Identification Toolbox of Matlab has been used. The toolbox is capable of estimating different kinds of mathematical models, including state space models. The toolbox adjusts the matrix elements of the state-space model until its output coincides as good as possible with the measured output. The obtained model can be validated by looking at the model's output compared to the measured output on a data set that was not used for the model fit.

The standard Matlab system identification notation for discrete-time state space models is:

$$\begin{aligned} \underline{x}(t+1) &= A\underline{x}(t) + B\underline{u}(t) + K\underline{e}(t) \\ \underline{y}(t) &= C\underline{x}(t) + D\underline{u}(t) + \underline{e}(t) \end{aligned} \tag{5.1}$$

with $\underline{u}\epsilon\mathbb{R}^m$, $\underline{y}\epsilon\mathbb{R}^k$ and $\underline{x}\epsilon\mathbb{R}^n$. The model order is n. The matrix K determines the disturbance properties. If $K = 0$, the noise source $\underline{e}(t)$ only affects the output and no

specific model of the noise properties is built by the toolbox. Note that discrete state space models can only be a good representation of the discrete event system under the assumption that no discontinuities are involved. This means that buffers have infinite capacity and buffer levels are not lower-bounded and machines have infinite capacity.

The input signals imposed on the discrete event system to be identified must be 'sufficiently rich'. This means that the signal must contain a lot of frequencies. Therefore, so called 'chirp' signals have been chosen. A chirp signal is a swept-frequency cosine signal. For the identifications used in this research, linear swept-frequencies have been used. Parameters involved are f_0 and f_1, the frequency at time=0 and at the end of the simulation (time=t_1) respectively. The mathematical representation of the chirp signal is:

$$\begin{aligned} \beta &= \frac{f_1 - f_0}{t_1} \\ y &= \cos\left(2\pi\left(f_0 + \beta t\right) t\right). \end{aligned} \tag{5.2}$$

Of course, the chirp signals can be accumulated with a constant (mean) and multiplied with a desired amplitude. Completing a relatively long simulation time is necessary, in order to include a reasonable amount of periods in the input signals.

First, identifications of simple systems are made to become familiar with the technique. For this purpose, a one-machine manufacturing system processing only one product type has been identified (section 5.2). Then, an identification of a simple flowline (three machines) is made (section 5.3). A simple re-entrant system (one machine, one product entering twice) follows (section 5.4), introducing the need of scheduling rules. Finally, the same configuration is identified with two different product types (section 5.5). Unexplainable problems arise in this situation, so a suggestion to investigate this is made and elaborated on in Chapter 6.

5.2 Identification of a GBME flowline with one product type

As a first finger exercise in system identification, an identification of a simple one-machine manufacturing system is made (Figure 5.1). A discrete event model (χ version 0.7, Appendix D.1) has been made and identified. The input signals are the incoming product rate u and the maximum machine production rate m. This production rate is varied to simulate measurable process time variations. The input signals' parameters are presented in Table 5.1. The outputs of interest are the buffer contents (from process B) and the number of finished products in process E. A simulation of 10000 time units has been done. This is ten times the longest period in the input

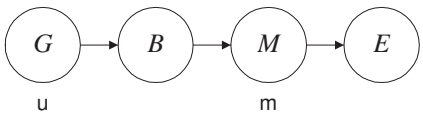


Figure 5.1: Iconic representation of the GBME flowline.

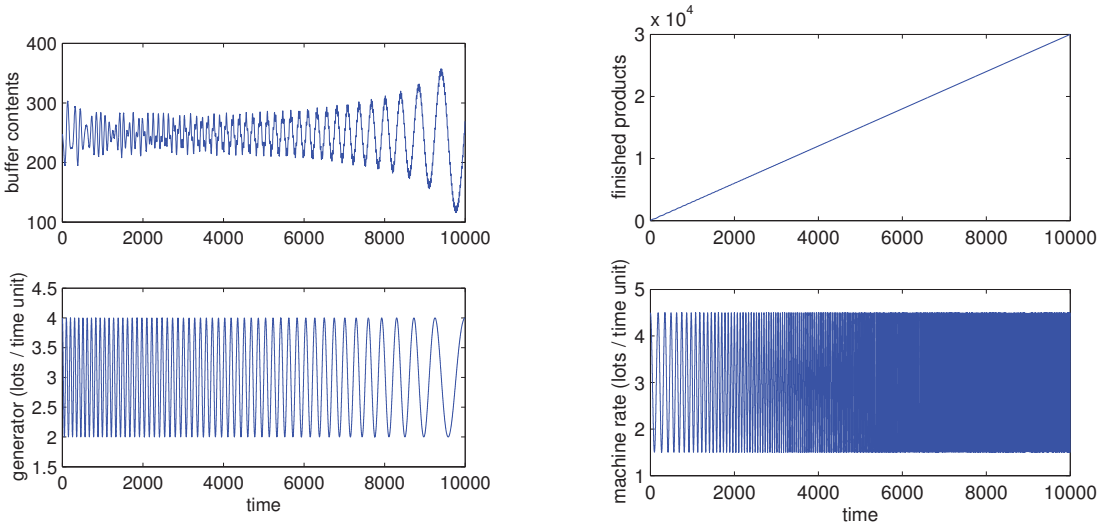


Figure 5.2: Input signals and measured output data used for identification.

signals, in this case input u. An important condition to use state space notation is that no discontinuities are involved. Therefore, the buffer B is not allowed to be or become empty. A (large) variation around a stable situation has been simulated with initial buffer contents greater than zero, to prevent them from becoming empty. The model output data with corresponding time values have been resampled with timestep 1, since the Identification Toolbox expects equidistant samples. The timestep is bigger than the smallest interval between two events. It is not necessary to be able to point out each event in the resampled dataset. The global phenomena and trends are more important in the continuous approximations. Moreover, it is possible that two events affecting the same buffer level occur at exactly the same time. Besides that it is not possible, keeping this information in the resampled dataset is useless.
Multiple state-space models have been fit on the data. The input and output data used for identification are shown in Figure 5.2.

The state space models have been fitted with the Matlab `n4sid` routine, available in the System Identification Toolbox. A second order state space model computed with focus

	f_0 **(Hz)**	f_1 **(Hz)**	**mean**	**amplitude**
Identification				
u	0.01	0.001	3.0	1.0
m	0.005	0.05	3.0	1.5
Validation				
u	0.007	0.003	3.0	2.0
m	0.007	0.04	3.0	0.5

Table 5.1: Parameters of input signals for identification and validation.

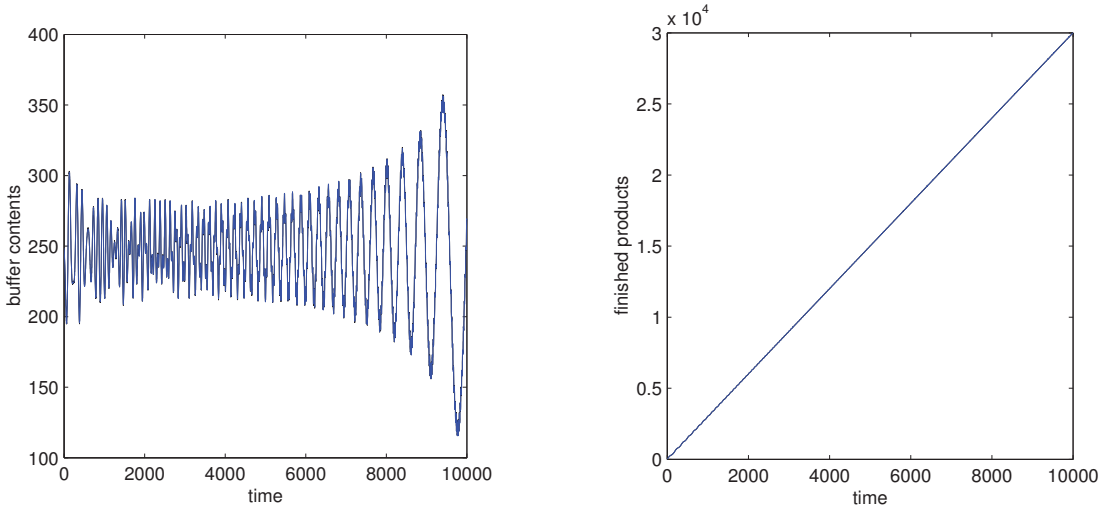


Figure 5.3: Second order state space model (blue) compared with original measurement data (black).

on simulation (option of the toolbox, other options are prediction, stability or custom filter) gives best results in model output and model validation, although all computed models fit with an accuracy of almost 100%. To compare the state space model with the measured data, this model was simulated with the original input signals. Comparison is shown in Figure 5.3. As the original measurement data (black line) is almost invisible, it is concluded that the state space model is a good fit. The state space description of the second order state space model is (structure of Equation 5.1):

$$A = \begin{pmatrix} 1 & -1.978e-08 \\ -7.798e-07 & 0.9999 \end{pmatrix}, \quad B = \begin{pmatrix} 5.004e-07 & 6.339e-07 \\ -2.321e-04 & 2.319e-04 \end{pmatrix},$$
$$C = \begin{pmatrix} 341.45 & -4306.5 \\ 8.812e05 & 1899.5 \end{pmatrix}, \quad D = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad K = \begin{pmatrix} 1.027e-07 & 1.668e-07 \\ -4.766e-05 & -2.638e-05 \end{pmatrix}$$

Note that the state $\underline{x}$ does not have a physical meaning.

To validate the identified model, a different dataset (input signal parameters in Table 5.1) was used. Choosing the validation frequencies of the input signals within the bandwidth of the original input signal is important, because extrapolation of the model does not give reliable results. The input signals and discrete event output are shown in Figure 5.4.

The comparison between the state space model simulated with the new dataset and the discrete event output is shown in Figure 5.5. Again, the blue line (state space model) almost completely matches the black line (discrete event output). This validation shows that system identification can be useful in modelling discrete event systems.

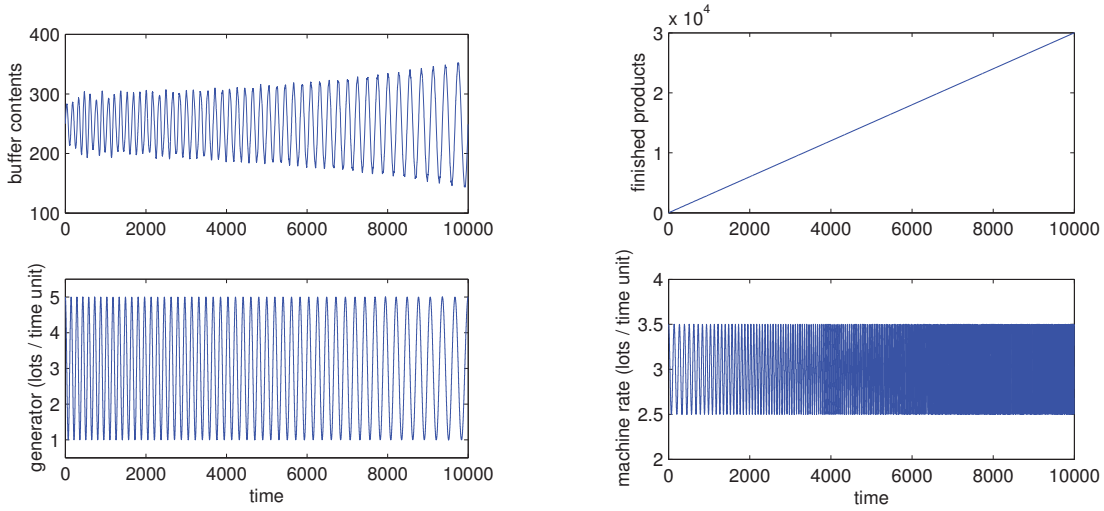


Figure 5.4: Input signals and measured output data used for validation.

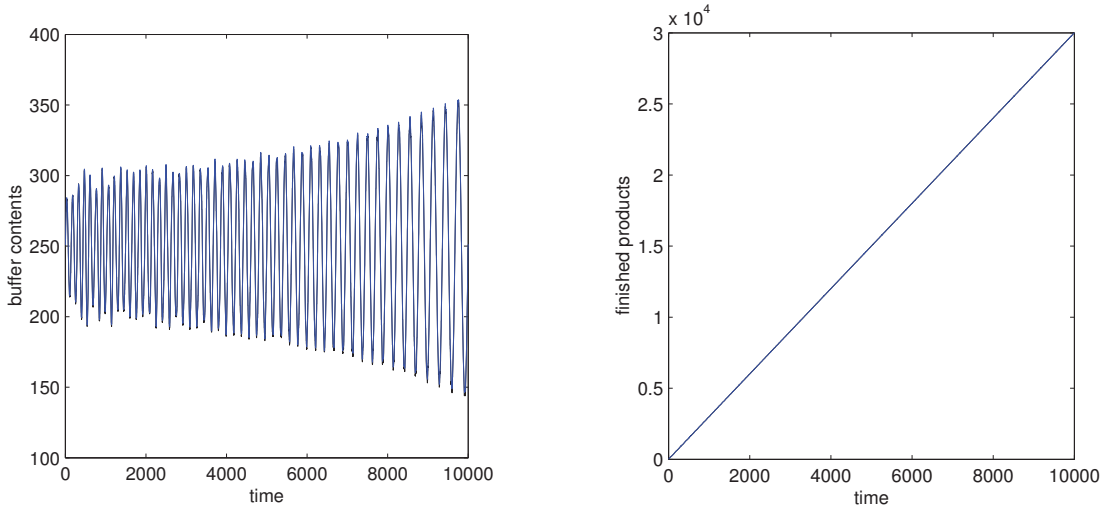


Figure 5.5: Second order state space model (blue) compared with validation measurement data (black).

A possible risk is the unconscious use of non-tolerable input signals. For example, using constant input signals is tempting. This is not allowed, because the frequency of constant input signals is zero. This frequency lies outside the bandwidth of the identified model. Figure 5.6 shows the model output of the same second order model applied to constant input signals ($u = 3$ and $m = 3$). It is clear that the buffer level prediction is completely useless, since it increases monotonously while the real buffer contents remains constant.

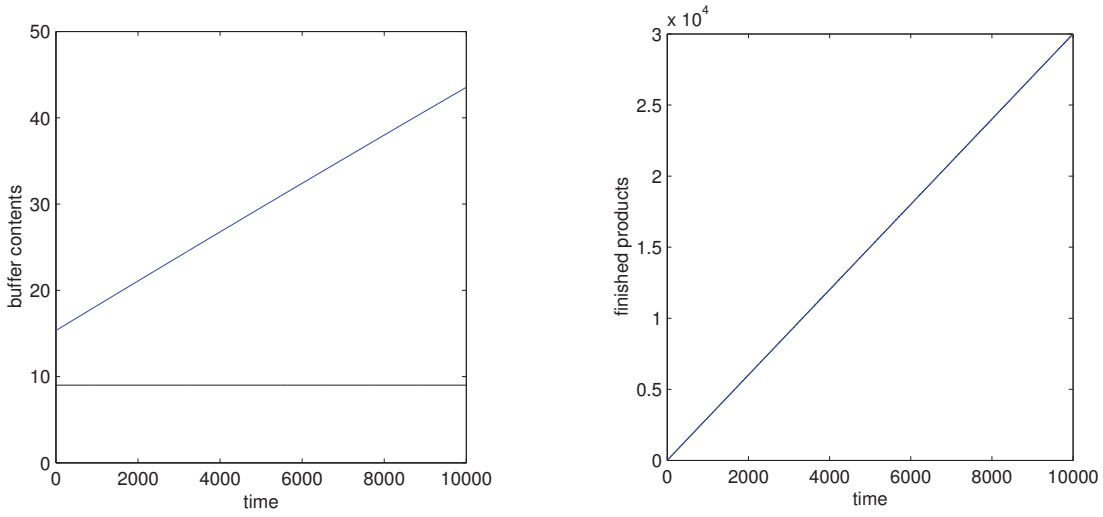


Figure 5.6: Second order state space model (blue) compared with measurement data based on constant input signals (black).

5.3 Identification of a GBMBMBME flowline with one product type

A second exercise in system identification is the identification of a GBMBMBME flowline producing one product type (see Figure 5.7, χ-source available in Appendix D.3). Simulation data for both identification and validation are shown in Table 5.2. A fourth order, eighth order and twelfth order model have been fit. They all fit very well with the original dataset. A comparison between the fourth order model output and the original measurement data is shown in Figure 5.8. For all four outputs, the fit is 99%. The fitted models have been validated with different input signals (Table 5.2) and measurement data. The validation results for the fourth order model are shown in Figure 5.9. Again, the fitness is above 90%. The matrices of the fourth order state space model are shown in Appendix D.4.

When the model order is the same as the number of outputs of the system, the A-matrix resembles the unity matrix. In the discrete time state space situation, this means nothing but input accumulation. In other words: the individual states are independent from each other. When the model order is chosen bigger than the number of outputs, some additional dynamics is introduced in the A-matrix, not necessarily improving the model validity. Higher order models do not outperform lower order models, especially not in the validation process. As the model order is of great importance and must not be too large in consideration of computational power, the following rule of thumb is proposed:

> The number of states in the state space description equals the number of measured outputs of the discrete event system.

This means that for the re-entrant flowshop case study, the number of outputs equals 43 and so is the model order of the state space system. The identification techniques (like implemented in the System Identification Toolbox) have to be capable of handling this relatively large model order. One has to realize that this rule of thumb is a 'minimal rule'. The number of states can never be smaller, because all buffers are physically independent. But extending the rule of thumb will irrevocably lead to non-workably large models, if possible to determine at all.

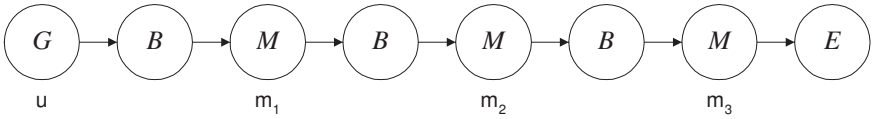


Figure 5.7: Iconic representation of the GBMBMBME flowline.

	f_0 (Hz)	f_1 (Hz)	mean	amplitude
Identification				
u	0.01	0.001	3.0	1.0
m_1	0.0001	0.05	3.0	1.5
m_2	0.03	0.0007	3.0	0.5
m_3	0.00001	0.01	3.0	2.0
Validation				
u	0.0005	0.008	3.0	1.5
m_1	0.0003	0.04	3.0	1.0
m_2	0.0009	0.02	3.0	2.5
m_3	0.009	0.0001	3.0	1.5

Table 5.2: Parameters of input signals for identification and validation.

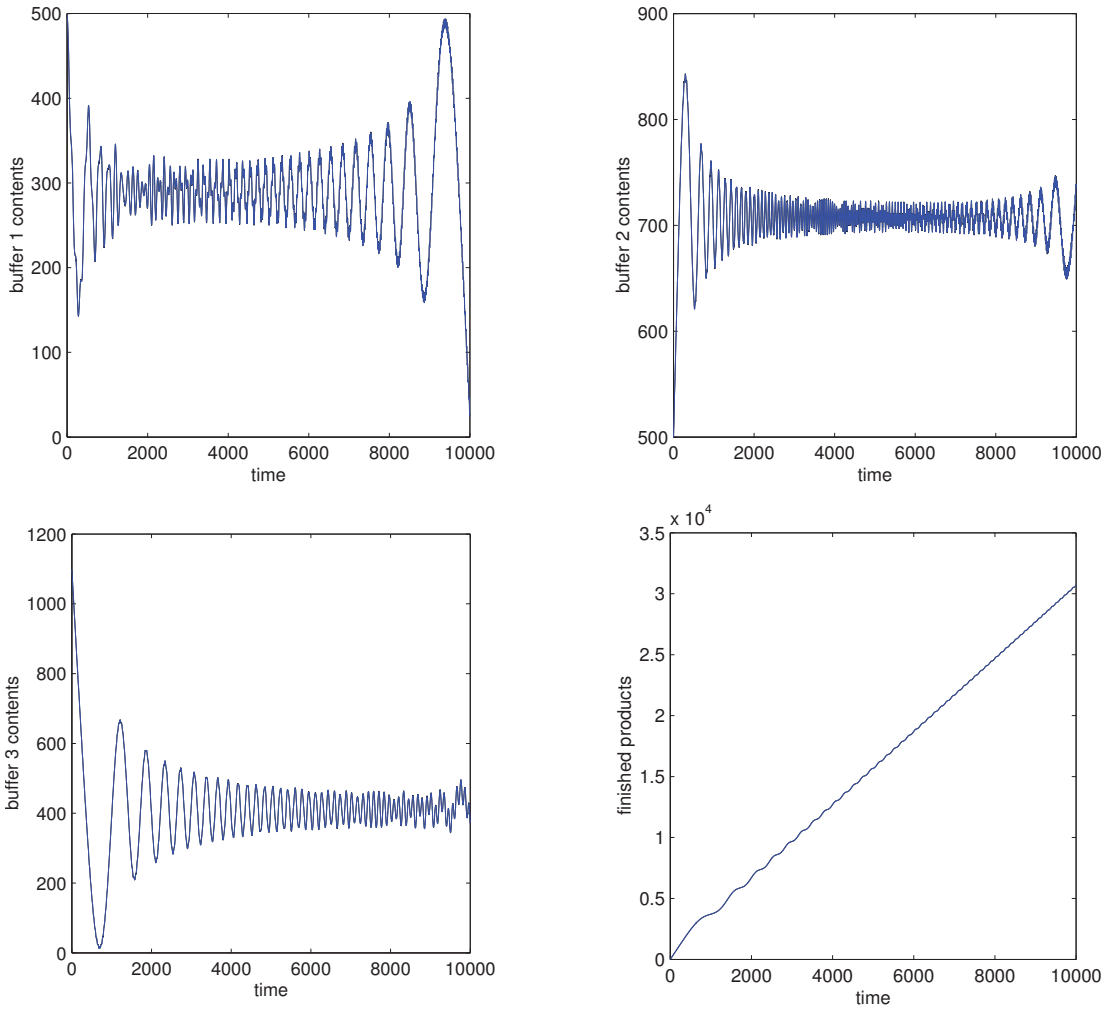


Figure 5.8: Fourth order state space model (blue) compared with original measurement data (black).

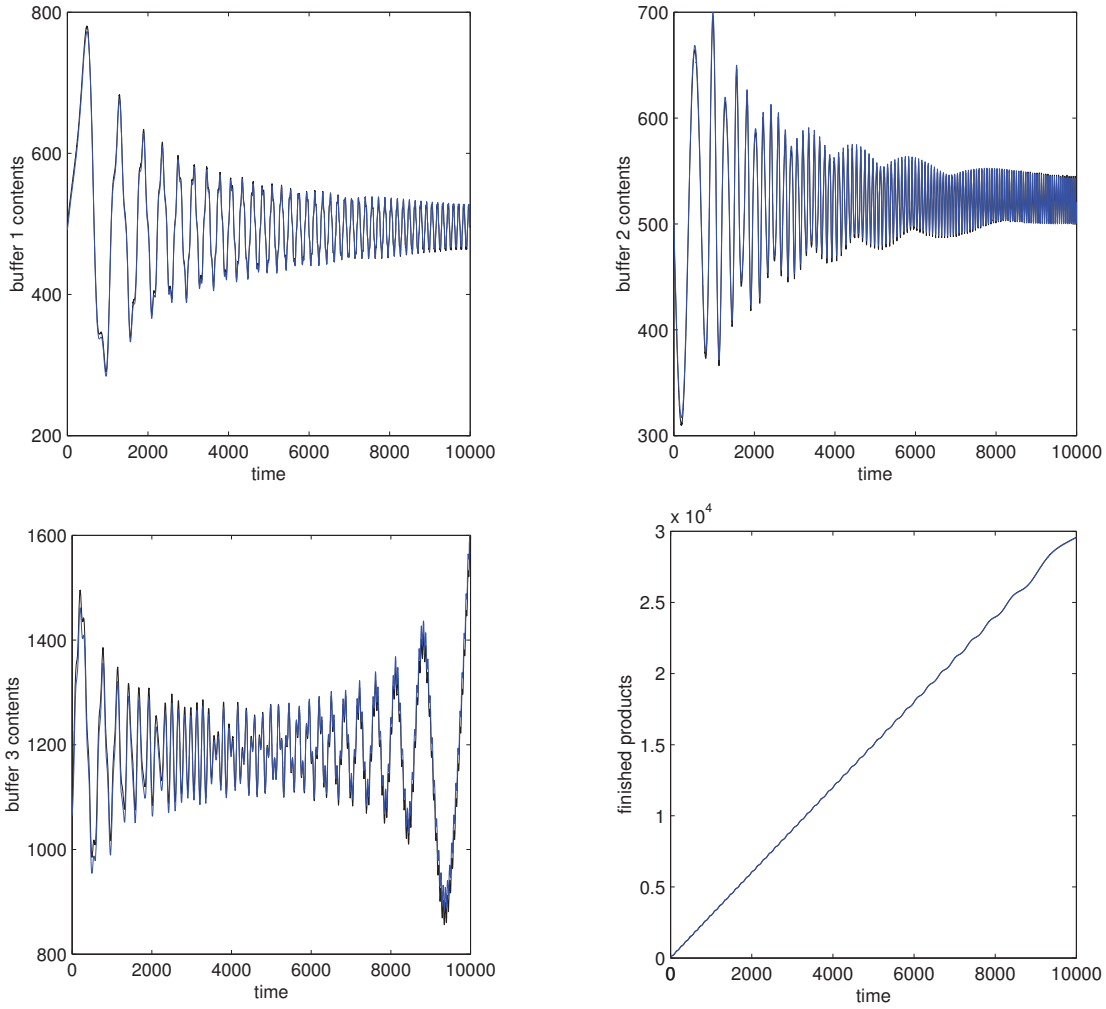


Figure 5.9: Fourth order state space model (blue) compared with validation measurement data (black).

5.4 Identification of a GBMBME re-entrant flowline with one product type

Two phenomena introduce the need of scheduling rules in the simulations. First of all re-entrant behavior. A system with one machine and one product type entering twice is identified and discussed in this section. The second phenomenon is producing more than one product type in a manufacturing system. This is discussed in the next section.

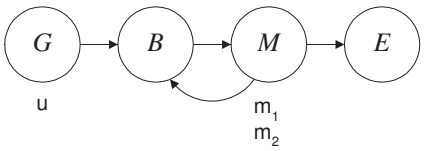


Figure 5.10: Iconic representation of the GBMBME re-entrant flowline.

A simple re-entrant flowline (Figure 5.10) is considered. Products have to enter the machine twice before completion. The buffer B consists of two subbuffers, from which the machine M can take products freely. The machine has to decide which of the two different product stages will be processed. It is assumed that neither setup times nor transport times exist. The decision is based on the same rules as described in Section 3.4. A desired production rate is integrated by a controller and compared to the realized production. The largest relative difference between desired production and realized production determines the choice of the machine. The χ-model of the system is given in Appendix D.5. As can be seen, exactly the same function `decision` as in Section 3.4 has been used.

This scheduling rule introduces two more inputs to the system. Not only the maximum machine production rate can vary (as in the two previous sections), but also the desired production rate may fluctuate. Therefore, the system of Figure 5.10 has 5 input signals (u, m_1, m_2, $m_{1,\text{desired}}$ and $m_{2,\text{desired}}$) and 3 output signals (two buffer levels and a stock level of finished products). Fluctuation of the maximum production rates m_1 and m_2 can be interpreted as a measurable process time deviation. The desired production rates do not have to equal those maximum rates, so new system inputs are involved. The parameters of the chirp signals used for system identification and validation are given in Table 5.3. Parameters for the maximum production rates have been given a smaller amplitude than the desired production rates parameters, since the latter are more likely to fluctuate heavier than the first. Moreover, the sum of the desired production rates (2.0) is below the maximum production rate of a single product (3.0), to keep the utilization of machine M below one.

Realizing that the system identification process also identifies the scheduling rule of the discrete event system is very important. A different choice for the scheduling rule will most likely result in a different state space model. This implies that applying a different scheduling rule to the system makes a new identification necessary. It is imaginable that simply trying a new scheduling rule therefore does not lie within the possibilities of normal plant management. This behavior is a disadvantage of the system identification method to obtain a continuous approximation model. In Chapter 6 a different modelling

method is explained, in which the scheduling rule is not included.

Straightforward system identification did not give good results in the validation process (graphs not included). Therefore, some additional knowledge has been put in the identification process: The A-matrix is not identified anymore, but manually set to the identity matrix, based on the experienced obtained in the previous sections. This state space model approach is only valid in case the model order equals the number of outputs of the system. System dynamics, such as time delays, are hereby excluded from the model. This approach is thus only applicable in cases where occurring time delays are not measurable. In the stationary situation, without buffers becoming empty and the sum of the process times smaller than the sample time, this assumption is justifiable. The moment a new lot enters a system, a finished product leaves the system. The time delay (sum of the effective process times) does not show in these simulations, since individual lots are not depicted.

Results of the identification process are shown in Figure 5.11. A third order model has been fitted on the data (matrices in Appendix D.6). The rule of thumb seems to hold here. Using different input signals, a validation test has been carried out. Results of this experiment are shown in Figure 5.12. Although still quite comparable, the results are not as good as in the previous models. The global phenomena all occur, but when studied in a more detailed way, the differences definitely become greater than in the previous models.

	f_0 **(Hz)**	f_1 **(Hz)**	**mean**	**amplitude**
Identification				
u	0.01	0.0001	1.5	1.0
m_1	0.00005	0.02	3.0	0.25
m_2	0.0003	0.07	3.0	0.5
$m_{1,\text{des}}$	0.01	0.0001	1.0	1.0
$m_{2,\text{des}}$	0.0005	0.05	1.0	1.5
Validation				
u	0.0002	0.02	1.5	0.5
m_1	0.0005	0.01	3.0	0.5
m_2	0.0005	0.05	3.0	0.25
$m_{1,\text{des}}$	0.009	0.0003	1.0	1.5
$m_{2,\text{des}}$	0.03	0.0007	1.0	1.0

Table 5.3: Parameters of input signals for identification and validation.

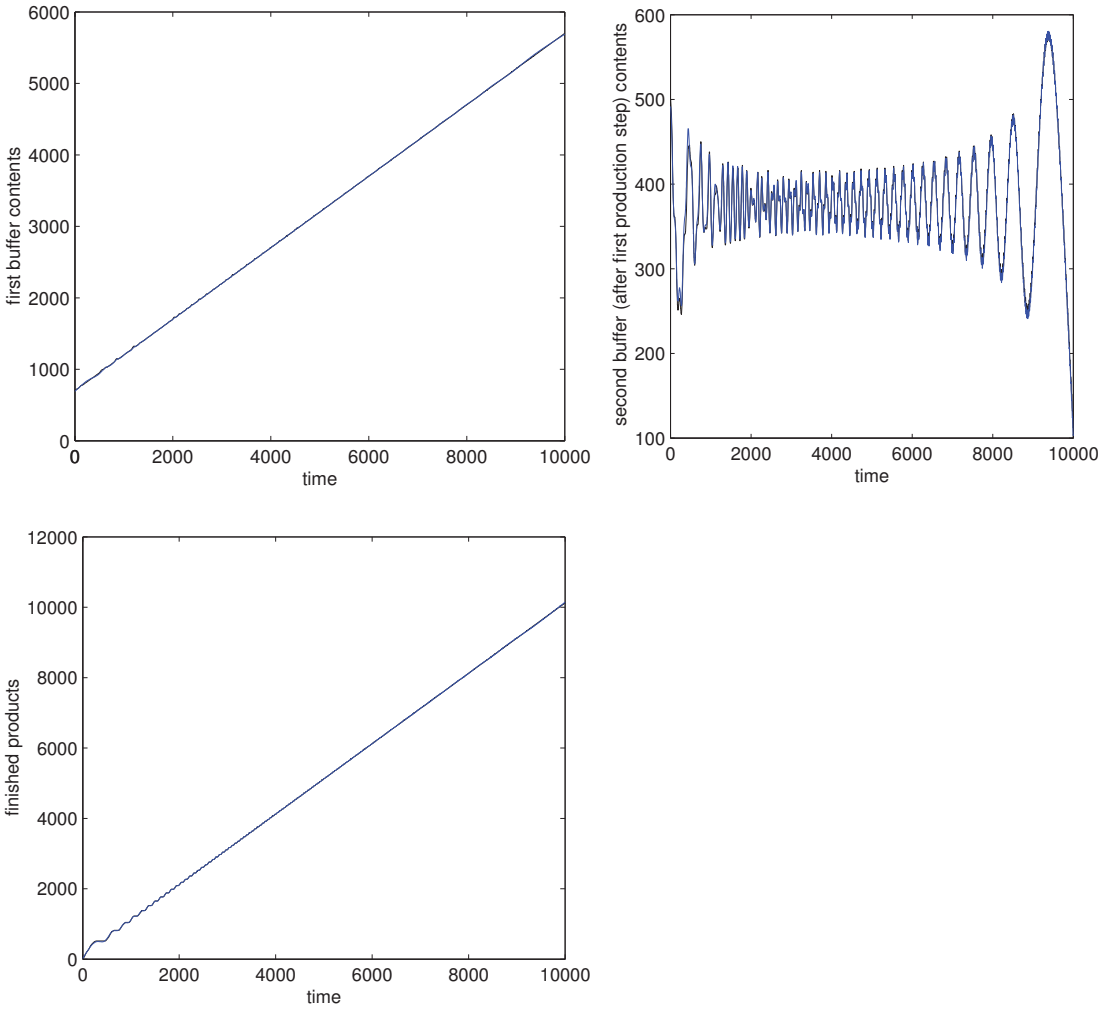


Figure 5.11: Third order state space model (blue) compared with original measurement data (black).

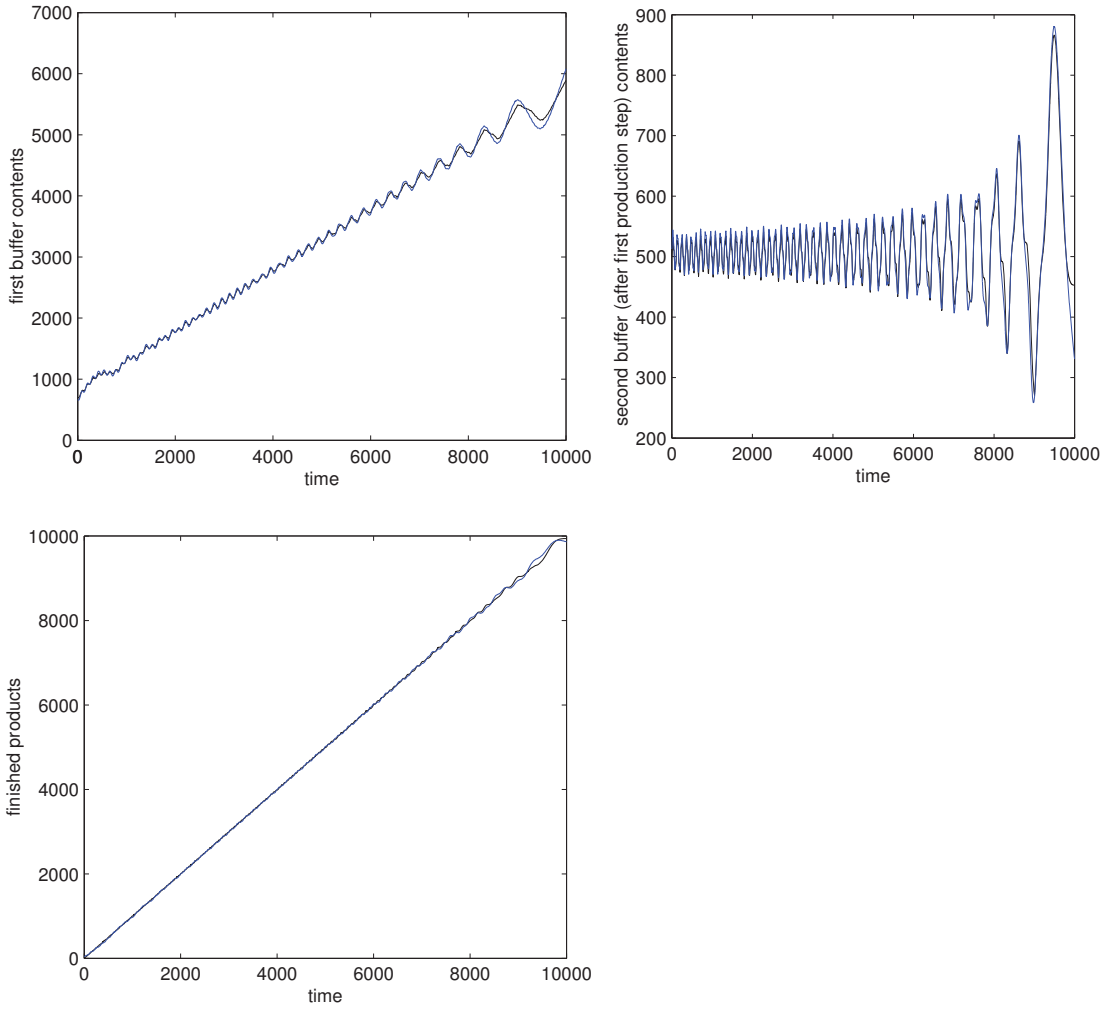


Figure 5.12: Third order state space model (blue) compared with validation measurement data (black).

5.5 Identification of a GBMBME re-entrant flowline with two product types

The second phenomenon requiring scheduling in the discrete event model is processing more than one product in a manufacturing line. A machine has to decide which product it has to take from the buffers. Based on the desired production rates, the choice has to be made. Again the function `decision`, used many times before, can be used for this.

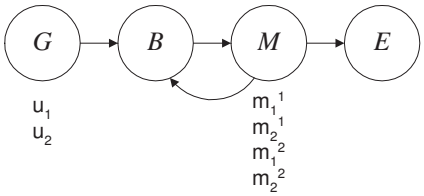


Figure 5.13: Iconic representation of the GBMBME re-entrant flowline with two product types.

Consider the re-entrant flowline of Figure 5.13. Two different products are made, each entering the machine twice. The buffer B thus consists of 4 subbuffers. The maximum production rates are defined as m_{11}, m_{12}, m_{21} and m_{22}. Together with the desired production rates and incoming product rate, they are quantized in Table 5.4. The discrete event simulation model is included in Appendix D.7.

The number of outputs of the model is 6 (4 buffer levels and 2 finished products stocks). Therefore, a sixth order state space model has been fit. The results of the identification process are shown in Figure 5.14. The sixth order model (matrices in Appendix D.8) fits quite well with the original measurement data, but greater differences appear in comparison with the previous identifications. A validation dataset has been obtained from a new experiment (parameters in Table 5.4). The sixth order model has been validated with this set. The results are shown in Figure 5.15. It is obvious that the state space model is by no means a good representation for this manufacturing line.

Although the validation data input set was within the bandwidth of the original input set, the results of the validation are far from desired. It seems that the system identification toolbox does not give reliable state space models to predict the system's behavior within the given assumptions and the rule of thumb derived in the previous sections.

In the next chapter, this rule of thumb is investigated further with a completely different modelling method. On an analytical basis, state space models with process time included are derived. These new models can be used to validate the rule of thumb used in this chapter.

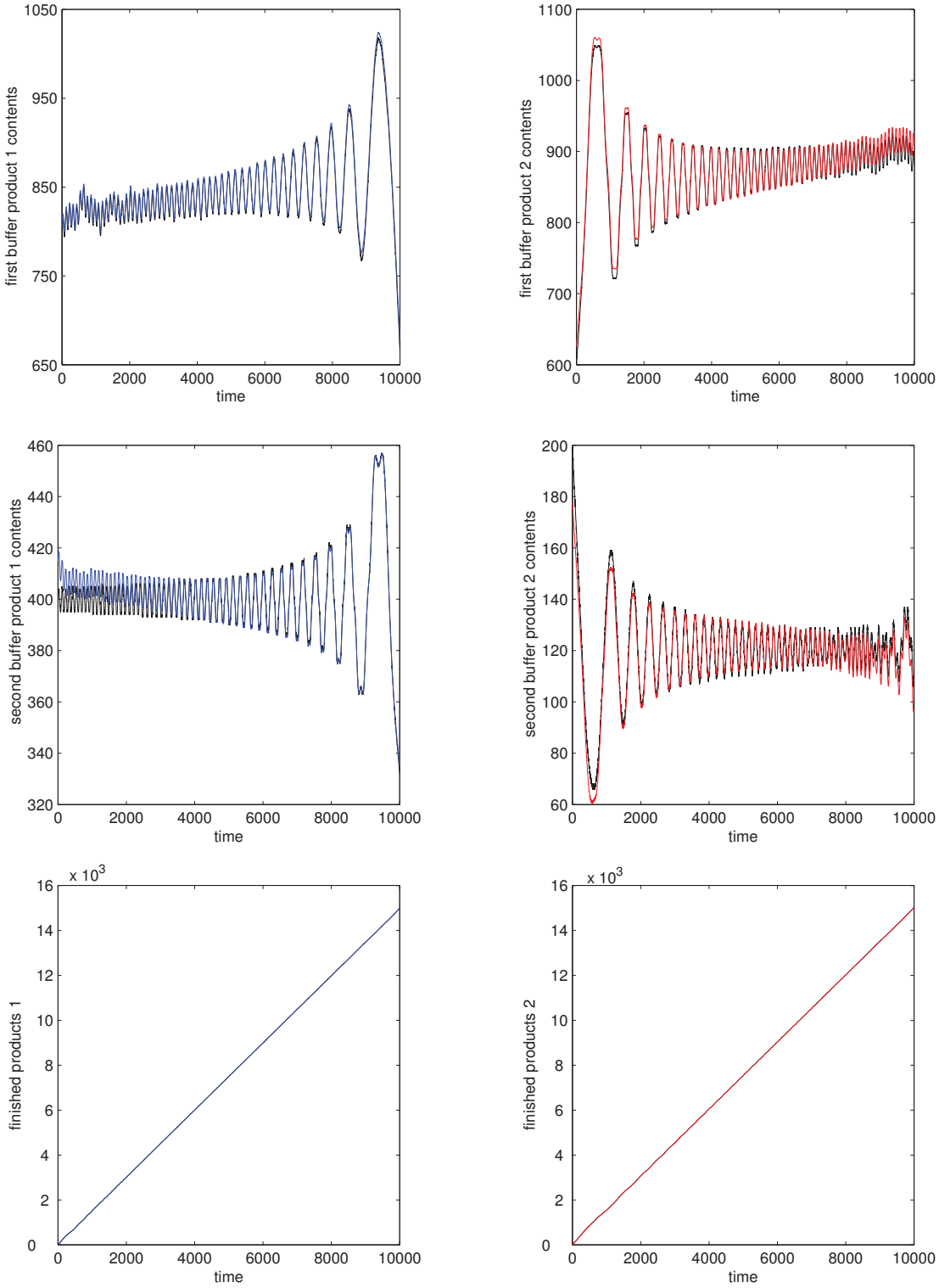


Figure 5.14: Sixth order state space model (blue and red) compared with original measurement data (black).

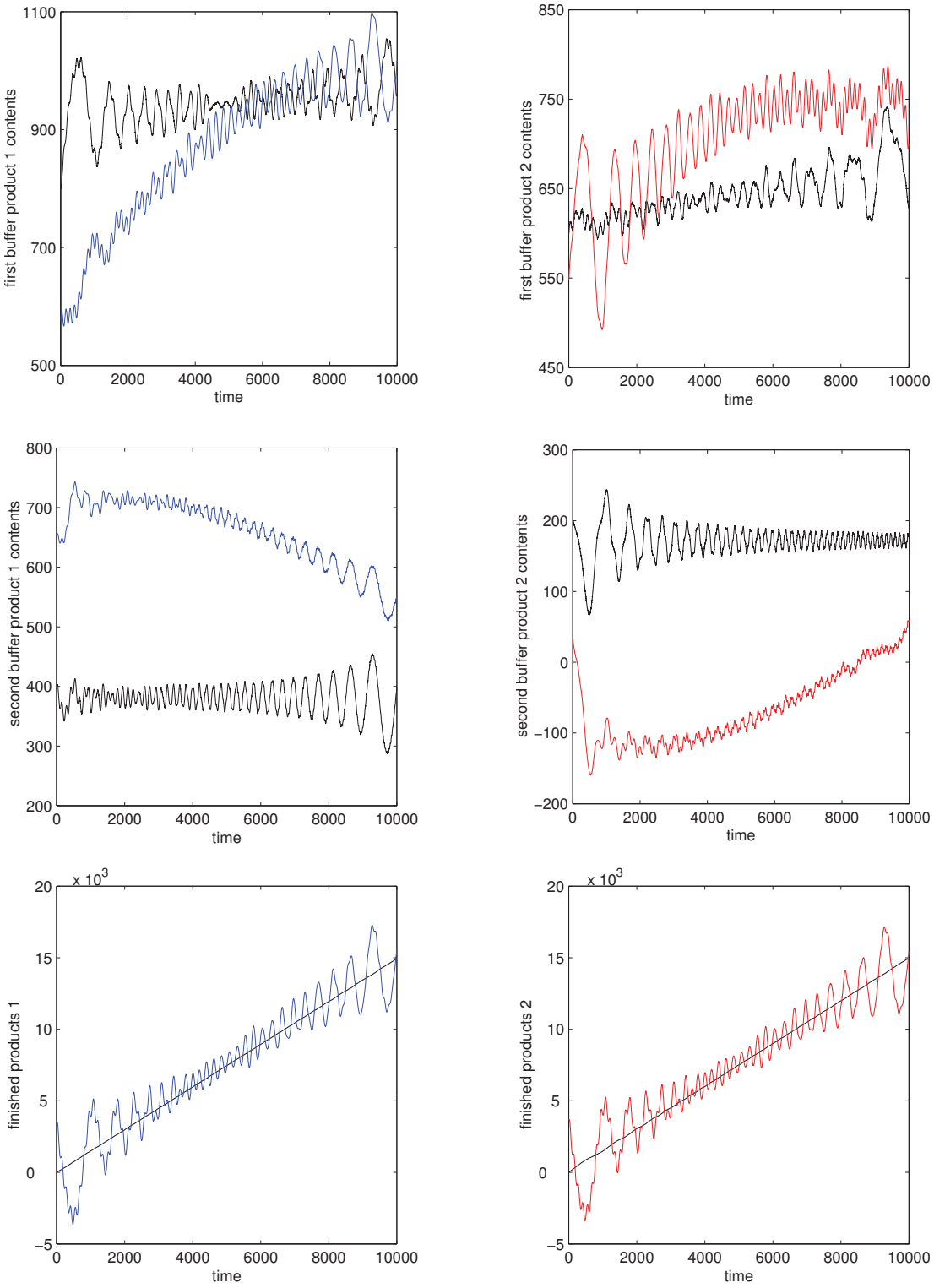


Figure 5.15: Sixth order state space model (blue and red) compared with validation data (black).

	f_0 **(Hz)**	f_1 **(Hz)**	**mean**	**amplitude**
Identification				
u_1	0.01	0.0001	1.5	1.0
u_2	0.0001	0.01	1.5	1.0
m_{11}	0.01	0.0001	6.0	0.5
m_{12}	0.0001	0.01	6.0	0.25
m_{21}	0.04	0.0004	6.0	0.75
m_{22}	0.0003	0.03	6.0	1.0
$m_{11,\text{des}}$	0.01	0.0001	5.0	1.0
$m_{12,\text{des}}$	0.0005	0.05	5.0	1.5
$m_{21,\text{des}}$	0.05	0.0005	5.0	0.5
$m_{22,\text{des}}$	0.0001	0.01	5.0	1.0
Validation				
u_1	0.0002	0.009	1.5	0.5
u_2	0.006	0.0003	1.5	0.25
m_{11}	0.0002	0.009	6.0	0.25
m_{12}	0.0003	0.008	6.0	0.75
m_{21}	0.03	0.0006	6.0	1.0
m_{22}	0.01	0.001	6.0	0.5
$m_{11,\text{des}}$	0.008	0.0008	5.0	2.0
$m_{12,\text{des}}$	0.04	0.0006	5.0	0.5
$m_{21,\text{des}}$	0.0007	0.03	5.0	1.0
$m_{22,\text{des}}$	0.0004	0.008	5.0	1.5

Table 5.4: Parameters of input signals for identification and validation.

5.6 Reflections on system identification

System identification is a technique mainly used to identify mechanical or electrical analogous systems. Generally, the number of inputs and outputs in these systems is limited or at least manageable. System identification has not been used very much in identifying manufacturing systems. This study is one of the first attempts for the Systems Engineering Group.

The System Identification Toolbox of Matlab is a convenient tool in identifying small scale systems. The results of identification of very small manufacturing systems look promising. But the method comes with some disadvantages. First of all, the state space models are only valid in continuous situations. Buffers are not allowed to become empty and machines can not be overloaded. Especially the buffer problem is a great disadvantage. Another problem is the bandwidth of the state space models. The state space models obtained by identification are only valid within the bandwidth they were identified with. Constant input signal are therefore out of reach. This is a major disadvantage of the identification method. Besides this, the user should choose a model order of the state space system to be identified. This model order is very important. A too small model order does not lead to a good system's description and a too large model order will overdetermine the system. The model validity will decrease then. Another disadvantage is computational effort. For relatively small systems, the computational effort is already substantial. For larger systems, the computational effort increases more than proportionally. The scheduling rule in the discrete event system is included in the identification process. One can imagine that this is not always desired, for simple changes in the scheduling rule cannot be tested without completely re-identifying the system.

A rule of thumb has been proposed to avoid too large model orders. The rule of thumb stated that the model order of the state space model equals the number of outputs of the discrete event model. For small systems, this rule seemed to be good, but for slightly larger systems with re-entrant behavior and multiple products, the rule of thumb failed. The reason for this is investigated in the next chapter. In Chapter 6, a new method for deriving state space models is developed, to overcome the disadvantages of the system identification method.

Chapter 6

Deriving state space models

6.1 General

In Chapter 5 the proposed method for using System Identification techniques to obtain state space models did not yield satisfactory results. So far, it is unclear why the technique works fine for simple models and gives bad results for more complex (but still very basic) models. In this chapter, the rule of thumb used in system identification is investigated and eventually reconsidered. A different technique is used: deriving a state space model with time delay approximations. These time delay approximations represent the process times of individual machines. Time delays are approximated using the Padé method, which is described in Section 6.2.2.

First, the technique of deriving (workable) state space models is explained in Section 6.2. Different aspects are treated, such as time delay, transfer functions, state space realizations and model reduction. Then, a closer look at the discrete event models is taken in Section 6.3. Finally, the technique is applied to a GBME flowline (Section 6.4) and a GBMBMBME flowline (Section 6.5) and reflections are made in Section 6.6.

6.2 Technique of deriving workable state space models

6.2.1 Goal

The rule of thumb stated in the previous chapter was:

> The number of states in the state space description equals the number of measured outputs of the discrete event system.

It seemed that system identification based on this rule of thumb was not always an appropriate way to obtain state space descriptions of the manufacturing systems. One of

the goals of this research is to model the time delay involved with production. Products arrive at a machine at a given rate and leave the machine with the same rate, but shifted in time. This time shift is the process time of the machine. We try to capture this time delay in a workable state space model, i.e. a state space model with a workable number of states.

6.2.2 Time delay

Consider a continuous product flow on a machine M at time t (Figure 6.1). Products arrive at machine M with speed $m(t)$. Products are processed with process time T and then leave the machine. Taking mass conservation into account, this departure rate must be the same as the arrival rate. So at time t, products leave the machine with rate $m(t-T)$. This is similar to compressible fluid flow (with constant flow rate v and time-varying incoming density r) through a straight tube with length L. The constant flow rate in combination with the tube length resemble the process time of the machine, while the density at the end of the tube equals the incoming density r shifted L/v time units (Figure 6.2).

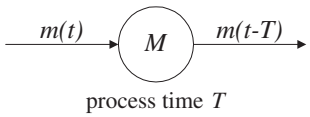


Figure 6.1: Time shift of arriving products: process time.

The Laplace transform of a time delay is an exponential function:

$$\mathcal{L}\{m(t-T)\} = \mathrm{e}^{-Ts} \cdot M(s). \tag{6.1}$$

This exponential function can not be written into workable state space notation: the order would become infinite. Therefore, an important approximation procedure is used: Padé approximation of an exponential function. The Padé approximation method writes the exponential function as a polynomial fraction. The order of the approximation can be chosen freely. The higher the order, the more accurate the approximation is. For

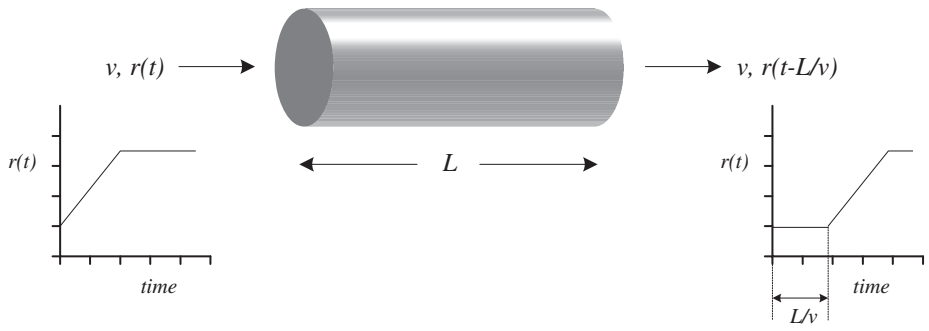


Figure 6.2: Similarity of tube flow to a machine.

example, the second order Padé approximation of e^{-Ts} is:

$$e^{-Ts} \simeq \frac{(Ts)^2 - 6Ts + 12}{(Ts)^2 + 6Ts + 12}. \tag{6.2}$$

The polynomial fraction can be embedded in state space notation. For SISO systems (Single Input, Single Output), this is a standard classroom exercise. For MIMO systems however (Multi Input, Multi Output), this is not a trivial procedure, but several (software) tools exist. State space realizations of transfer function matrices is discussed in the next subsection.

6.2.3 State space realization

A very common way to derive a state space model is to make a realization of a transfer function and in particular, a *minimal realization*.

Definition ([ZCB01]):
A realization (A, B, C) of a given transfer function is minimal if there exists no zero-state equivalent $(\tilde{A}, \tilde{B}, \tilde{C})$ whose dimension $\dim\{\tilde{A}\}$ is smaller than $\dim\{A\}$.

In practice, this means that a minimal realization is both observable and controllable. Analytical methods to compute a minimal realization exist, but is not discussed here. In this research, minimal realizations of a transfer function matrix are made with the `ss` command in Matlab.

For example, the Matlab minimal realization of the second order Padé approximation of unit delay is:

$$\text{Transfer function} = \frac{s^2 - 6s + 12}{s^2 + 6s + 12} \tag{6.3}$$

$$A = \begin{pmatrix} -6 & -3 \\ 4 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 4 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} -3 & 0 \end{pmatrix}, \quad D = 1. \tag{6.4}$$

6.2.4 Model order reduction

Not all modes in a state space model are equally important. Some of the states are stronger coupled to the inputs and outputs than others. To investigate this, the Hankel singular values can be computed. The procedure for this is:

Consider the continuous time state space model

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{6.5}$$

with controllability and observability grammians W_c and W_o. The change of coordinates $\overline{x} = Tx$ produces the equivalent model

$$\begin{aligned} \dot{\overline{x}} &= TAT^{-1}\overline{x} + TBu \\ y &= CT^{-1}\overline{x} + Du \end{aligned} \tag{6.6}$$

and transforms the grammians to $\overline{W}_c = TW_cT^T$ and $\overline{W}_o = T^{-T}W_oT^{-1}$. The transformation matrix T is chosen such that $W_o = W_c = \mathrm{diag}(g)$. The transformed state space model is a balanced realization. The vector g contains the Hankel singular values (HSV's).

The larger the Hankel singular value, the more influence the corresponding mode has. If some of the HSV's are big and others are small, one could think of model reduction: reducing the order of the state space model, keeping only the most important dynamics in the new state space model.

A useful tool in model reduction is the `SLICOT` toolbox for Matlab [BMS+99]. This toolbox is capable of reducing MIMO (multi-input, multi-output) state space models to a user-specified number of states.

After having realized a state space model of a transfer function matrix, the rule of thumb of the previous chapter can be validated by reducing the model order to the number of outputs of the discrete event system. The reduced state space model can be compared to the non-reduced minimal order state space realization.

6.3 Discrete event models

The discrete event models (in χ) of the manufacturing lines are simple but not trivial. Inputs for the model are the generator-speed (the rate at which the generator fills the first buffer in line) and the desired machine production rates. Particularly the latter can be interpreted in some different ways.

As in the discrete event model single products enter and leave machines and buffers (events), it is hard to define a rate at which this happens, except for the constant rate or statistical computations afterwards. Intuitively the constant rate is the reciprocal of the constant inter arrival time. But what if these inter arrival times are not constant? The 'rate' can not be defined ambiguously then. One could think of an instantaneous rate (the rate at *this* moment), but what if a machine is busy at the very moment? Another option is to calculate the next event based on the current desired rate: $t_{\text{next}} = t + 1/\text{rate}$. But the problem is that on time mark t_{next} the desired rate can be completely different. In this way, production tends to lag behind the desired production.

A different way to view the desired production rate is to accumulate (or integrate) this signal and only look at this total desired production. Intuitively this has been done in previous chapters, but this is also not trivial. The basic question is: When does

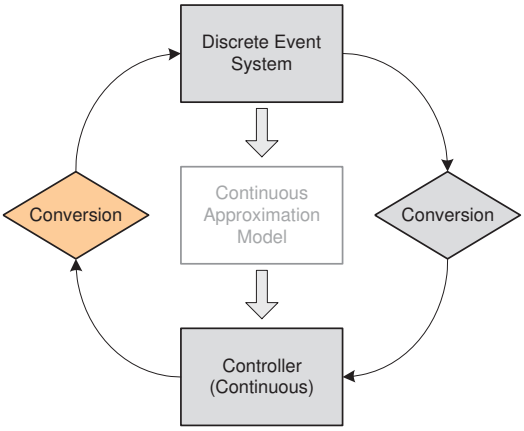
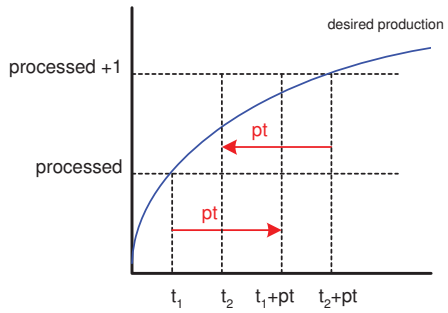


Figure 6.3: The decisions to be made by the machine are part of the left conversion block.

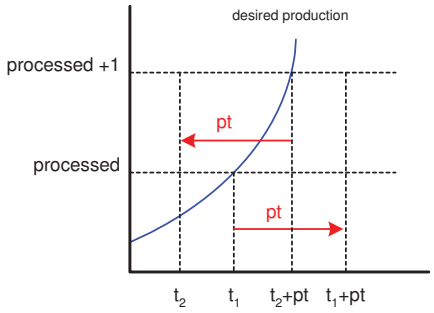
a machine have to start processing a lot? Numerous methods can be used; a few are discussed here and are shown in Figure 6.4. The decisions to be made are part of the left conversion block in the system-model-controller outline (Figure 6.3), since it translates the controller output (desired production rates) into single events.

Starting point in each situation is that the current number of finished products is called *processed*. The blue line is the integrated desired production rate, i.e. the total desired production. The process time of a lot is indicated as pt. In each subfigure, two extreme situations are shown and indicated with time marks t_1, t_2, $t_1 + pt$ and $t_2 + pt$.
The intuitive strategy (strategy 1) is shown in Figure 6.4(a) and is marked with t_1 and $t_1 + pt$. At time t_1, the integrated desired production rate becomes greater than the actual number of processed lots. The machine takes a lot from the preceding buffer and finishes it at time mark $t_1 + pt$. The number of processed lots is then $processed + 1$. The course of the blue line causes the desired total production to be below the actual production at that time mark $t_1 + pt$. A new lot is taken from the buffer as soon as the blue line crosses the $processed + 1$ line.
The second extreme strategy (strategy 2) is a JIT-like strategy (Just-In-Time). If the course of the blue line is known in advance, the start time of processing the lot can be computed backward. If it is known that at time mark $t_2 + pt$ the next lot must be finished on the machine, the machine should start at time mark t_2 to complete the lot in time. This strategy is only possible with knowledge of future desired production rates and predictable process times.
Figure 6.4(b) shows the same two strategies but shows that the course of the blue line may cause that strategy 2 takes the lot earlier from the buffer than strategy 1. Choosing strategy 1 in this situation implies that actual production always lags behind desired production.

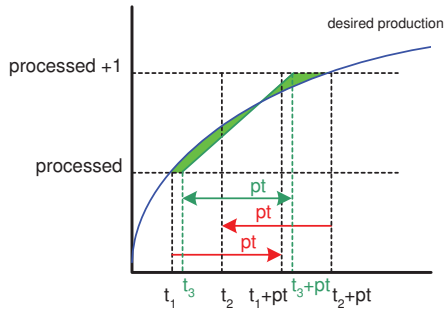
Complete different strategies are alternative strategy 1 (Figure 6.4(c)) and alternative



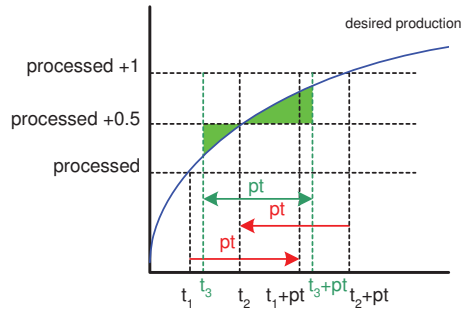
(a) Two extreme situations.



(b) The same two extreme strategies with different integrated signal.



(c) Alternative strategy 1.



(d) Alternative strategy 2.

Figure 6.4: Different strategies to decide when a machine has to start processing a lot. (All horizontal axes: time; all vertical axes: number of products.)

strategy 2 (Figure 6.4(d)). In both figures, intuitive strategy 1 and complementary extreme strategy 2 have been drawn for clarity.
Alternative strategy 1 computes the time mark t_3 at which the green shaded part above the diagonal line starting at t_3 equals the green shaded part below this diagonal line. The diagonal line can be interpreted as a linear progress indicator of the lot on the machine. If the two shaded parts are equal, the actual production has lagged behind the integrated desired production rate as much as it has run in front of it.
Alternative strategy 2 starts from the virtual line $processed + 0.5$. It then computes the new start time mark t_3 such that the green shaded part below this line and above the integrated desired production line equals the green shaded part below this integrated signal and above the $processed + 0.5$ line. The $processed + 0.5$ line can be shifted on one's own discretion.
Both alternative strategies require knowledge of the future desired production rate signal

for at least the period of one process time. This requirement can not always be fulfilled in practice. Moreover, a good prediction of the process time must be available, especially in the non-deterministic case.

It is clear that numerous strategies to choose the start time of processing a lot on a machine exist. In the next sections, both extreme situations (Figures 6.4(a) and 6.4(b)) are used.

6.4 GBME flowline

6.4.1 Characteristics

Reconsider the GBME flowline with only one product type (Figure 6.5). The buffer contents x_1 and the stock level x_2 are the only outputs of interest. The generator sends lots to the buffer B with generation rate $g(t)$. This input rate is a sinus: $g(t) = 1.5 + \sin(0.5t)$ and $g(t) \leq g_{\max} \quad \forall t$. The lot release policy for the discrete event simulation is based upon the integration strategy: as soon as the integral of the generator speed is greater than or equal to the actual number of released lots plus one, a lot is released. Note that the integral $G(t)$ of this sinus function ($G(t) = 1.5t - 2\cos(0.5t) + 2$) contains integration constant 2 to obey the initial condition $G(0) = 0$.
The machine M takes products from buffer B with rate $m(t)$. The rate $m(t)$ equals the desired rate $m_{\text{des}}(t)$ if $m_{\text{des}}(t)$ is smaller than the maximum rate $m_{\max}$ and without machine breakdowns. The rate $m(t)$ is also a sinusoidal signal: $m(t) = 1 + 0.5\sin(0.2t)$ and $m(t) \leq m_{\max} \quad \forall t$. The integral of the desired production rate (i.e. the total desired production) is $M(t) = t - 2.5\cos(0.2t) + 2.5$ with $M(0) = 0$ The process time of the machine ($1/m_{\max}$) equals 2/3 time unit. Note that the machine is never over-utilized. Two different simulations have been made: with both integration strategy 1 and strategy 2 (see Section 6.3).

6.4.2 Discrete event model

The discrete event model (χ version 0.7) is shown in Appendix E.1.1. The different extreme lot releasing policies are indicated in this model. Straightforward simulation

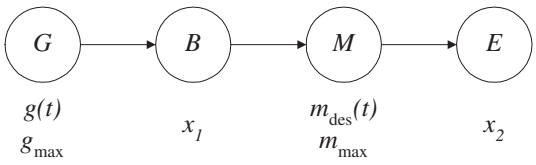


Figure 6.5: GBME flowline (iconic representation).

has been carried out. The results are be compared with a continuous approximation model, which is discussed in the next subsection.

6.4.3 Continuous approximation

The continuous approximation is based on the following assumption: The change in the buffer contents equals the incoming rate minus the outgoing rate and the products leave the machine M with the same rate as they entered the machine, but time shifted. The mathematical notation of this assumption is:

$$\begin{aligned} \dot{x}_1(t) &= g(t) - m_{\text{des}}(t) \\ \dot{x}_2(t) &= m_{\text{des}}(t-\tau) \\ \underline{y} &= \underline{x}. \end{aligned} \tag{6.7}$$

The time constant τ equals the process time of the machine, 2/3 time unit. The Laplace transforms of these differential equations are:

$$\begin{aligned} \mathcal{L}\{\dot{x}_1(t)\} &= sX_1(s) = G(s) - M_{\text{des}}(s) \\ \mathcal{L}\{\dot{x}_2(t)\} &= sX_2(s) = \mathrm{e}^{-\tau s} \cdot M_{\text{des}}(s). \end{aligned} \tag{6.8}$$

Now define $\underline{Y} = [X_1(s) \quad X_2(s)]^T$ and $\underline{U} = [G(s) \quad M(s)]^T$. The transfer function matrix $H(s)$ is then:

$$\begin{aligned} Y(s) &= H(s)U(s) \\ H(s) &= \begin{pmatrix} \frac{1}{s} & -\frac{1}{s} \\ 0 & \frac{1}{s}\mathrm{e}^{-\tau s} \end{pmatrix}. \end{aligned} \tag{6.9}$$

The exponential function has been approximated using second order Padé method. With the standard Matlab routine `ss`, a minimal realization of this transfer function matrix has been made:

$$\begin{aligned} A &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -4.5 & 5.196 & -1.5 \\ 0 & -1.299 & -4.5 & 1.299 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 4 \end{pmatrix}, \\ C &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0.25 \end{pmatrix}, \quad D = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}. \end{aligned} \tag{6.10}$$

The matrices A, B, C, D form a both controllable and observable system. With a second order time delay approximation, the minimal model order is 4. The obtained state space model A, B, C, D has been reduced with the `SLICOT` toolbox for Matlab, using the `SYSRED` command. To validate the rule of thumb stated at the beginning of

this chapter, the model order has been reduced to 2. The reduced state space model A_{red}, B_{red}, C_{red}, D_{red} is then:

$$\begin{aligned} A_{\text{red}} &= \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad B_{\text{red}} = \begin{pmatrix} 1 & -1 \\ 0 & -4.1633 \end{pmatrix}, \\ C_{\text{red}} &= \begin{pmatrix} 1 & 0 \\ 0 & -0.2402 \end{pmatrix}, \quad D_{\text{red}} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}. \end{aligned} \tag{6.11}$$

Note that these state space descriptions are continuous time state space descriptions. Using state transformation $\overline{x} = Tx$ with $T = C_{\text{red}}$ the transformed state space model $\overline{A}_{\text{red}}$, $\overline{B}_{\text{red}}$, $\overline{C}_{\text{red}}$, $\overline{D}_{\text{red}}$ is obtained:

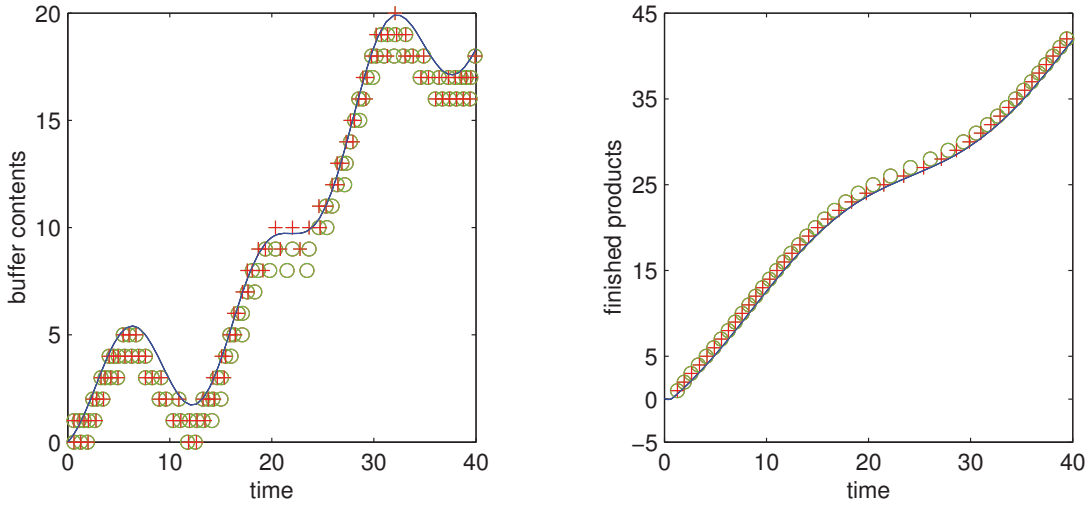
$$\begin{aligned} \overline{A}_{\text{red}} &= \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \overline{B}_{\text{red}} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}, \\ \overline{C}_{\text{red}} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \overline{D}_{\text{red}} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}. \end{aligned} \tag{6.12}$$

Note that the A_{red}-matrix contains zeros. The model reducing algorithm has removed the time delay approximated with Padé. It is clear that in Equation 6.12 the model description of Equations 6.8 without time delay has been found.

6.4.4 Simulation

The discrete event system and both state space models (original minimal and reduced) have been simulated. The discrete event system was simulated with χ (source code in Appendix E.1.1. The state space models were simulated with Simulink (model shown in Appendix E.1.2. A total run of 100 time units has been made, with a timestep (for the discrete event model) of 0.001 time unit. This timestep prevents the model from looping without time progression. It is expected that the discrete output resembles the continuous output. The results are shown in Figure 6.6. For clarity, only the first 40 time units have been plotted. At first sight, the continuous approximation gives a good indication of the buffer contents and finished products level. Both figures are studied more in detail.

Figure 6.6(a) (the buffer contents) shows two different kinds of events for the discrete event situation: filling the buffer with lots and taking lots from the buffer. For both machine decision policies, the filling of the buffer takes place at the same time: the green circle and red cross coincide, or take at least place at the same time. Around time=20, this seems not to be the case. In fact, the machine takes a lot from the buffer (strategy 1); then the generator sends a lot to the buffer and finally the machine takes



(a) Buffer contents.

(b) Finished products.

Figure 6.6: Simulation results of the GBME flowline. Lot releasing strategy 1 (green), strategy 2 (red) and continuous state space approximation (blue).

a lot from the buffer (strategy 2). That is why filling the buffer seems not to coincide for the two policies.

Figure 6.6(b) needs more explanation. First of all, noticing that the red-crossed simulation results are not better than the green-circled results, for the simple reason that the green crosses lie nearer to the blue line, is very important. To the contrary, the blue line is an approximation of both simulations, since the continuous approximation does not contain information about the different decision strategies. The scheduling rule of the machine (the decision strategy) is not part of the state space model anymore, as it was in system identification. The two machine strategies are clearly visible in this figure. Strategy 1 (green circles) processes lots earlier than strategy 2 (red crosses). Apparently, the situation of Figure 6.4(a) applies to this situation. Between time mark 25 and 35, the blue line looks more like Figure 6.4(b), but still strategy 1 produces earlier than strategy 2. The reason for this is that the desired production rate is always smaller than or equal to the maximum production rate. The system can be forced into strategy 2 by (temporally) demanding more than the maximum production rate, but then the continuous approximation model gives bad results since it is not based on these production limits.

Another phenomenon is striking: it looks like the discrete event simulation runs in front of the continuous approximation. For decision strategy 1 this is acceptable: products are indeed finished earlier than they should be. But strategy 2 also runs ahead of the blue approximation line. Figure 6.6(b) has been zoomed in and is shown in Figure 6.7. Two

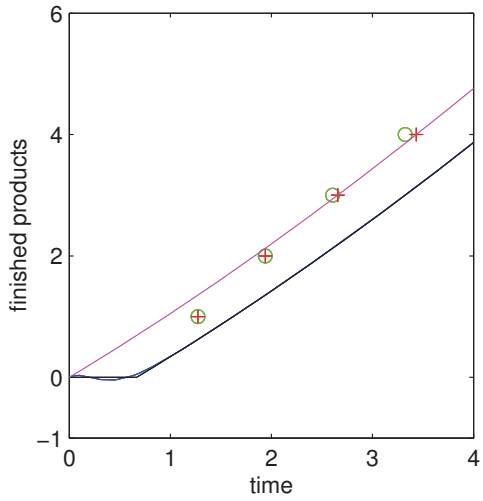


Figure 6.7: Closer look at the simulation results of Figure 6.6.

extra lines have been added. The purple line is the function $M(t) = t - 2.5\cos(0.2t) + 2.5$, the target production function of the machine. Since it is computed from $t = 0$, the purple line runs ahead of the blue approximations. The discrete event system lags behind this schedule during the first approximately 2.5 time units. From then, production becomes stable and the products made with decision strategy 2 are finished exactly on time.

An important conclusion is to be drawn here: in fact, the discrete event model is a closed loop model, since lots are taken from the buffer based on measurements (the number of actual processed lots). In the open loop situation, the discrete event output would match the continuous approximations, but this is difficult to realize for time-variant functions for production rates.

In Figure 6.7 the Padé approximation can also be viewed very well. The blue line is the second order Padé approximation of the black line (pure time delay). Note that the approximation becomes negative for a short time.

6.5 GBMBMBME flowline

6.5.1 Characteristics

Reconsider the GBMBMBME flowline with only one product type (Figure 6.8). Outputs of interest are the buffer levels x_1, x_2, x_3 and the number of finished products x_4. The generator G sends lots to the first buffer with generation rate $g(t)$. Again, this signal is $g(t) = 1.5 + \sin(0.5t)$ and $g(t) \leq g_{\max}\ \forall t$. The lot releasing policy is based upon the two

extreme strategies as explained before. The machines M take the products from the preceding buffers at a desired rate. These rates are sine-functions in time and stated in equation set 6.13.

$$\begin{aligned} m_{1,\mathrm{des}} &= 1.0 + 0.5 * \sin(0.2t) \text{ with } m_{1,\max} = 1.5 \text{ products/time unit} \\ m_{2,\mathrm{des}} &= 0.7 + 0.3 * \sin(0.2t + 1) \text{ with } m_{2,\max} = 1.0 \text{ products/time unit} \\ m_{3,\mathrm{des}} &= 0.3 + 0.2 * \sin(0.2t + 2) \text{ with } m_{3,\max} = 0.5 \text{ products/time unit} \end{aligned} \tag{6.13}$$

Note that the machines M are never over-utilized.

6.5.2 Discrete event model

The discrete event model (χ version 0.7) is shown in Appendix E.2.1. Again, the different extreme lot releasing policies are indicated in the model. Straightforward simulation has been carried out. The results are going to be compared with a continuous approximation model, which is developed in the next subsection.

6.5.3 Continuous approximation

Analogous to the model of the GBME flowline of the previous section, the continuous model of the GBMBMBME flowline is based on the assumption that the change of the buffer level equals the incoming rate minus the outgoing rate. Products leave a machine M with the same rate as they entered, but shifted in time, due to the process time of the machine. The mathematical notation of the continuous model is:

$$\begin{aligned} \dot{x}_1(t) &= g(t) - m_{1,\mathrm{des}}(t) \\ \dot{x}_2(t) &= m_{1,\mathrm{des}}(t - \tau_1) - m_{2,\mathrm{des}}(t) \\ \dot{x}_3(t) &= m_{2,\mathrm{des}}(t - \tau_2) - m_{3,\mathrm{des}}(t) \\ \dot{x}_4(t) &= m_{3,\mathrm{des}}(t - \tau_3) \\ \underline{y} &= \underline{x}. \end{aligned} \tag{6.14}$$

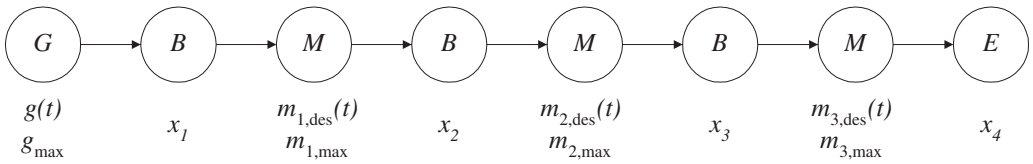


Figure 6.8: GBME flowline (iconic representation).

The time constants τ_n equal the process time of machine n. The Laplace transforms of these differential equations are:

$$\begin{aligned}
\mathcal{L}\dot{x}_1(t) &= sX_1 = G(s) - M_{1,\mathrm{des}}(s) \\
\mathcal{L}\dot{x}_2(t) &= sX_1 = M_{1,\mathrm{des}}(s)\mathrm{e}^{-\tau_1 s} - M_{2,\mathrm{des}}(s) \\
\mathcal{L}\dot{x}_3(t) &= sX_1 = M_{2,\mathrm{des}}(s)\mathrm{e}^{-\tau_2 s} - M_{3,\mathrm{des}}(s) \\
\mathcal{L}\dot{x}_4(t) &= sX_1 = M_{3,\mathrm{des}}(s)\mathrm{e}^{-\tau_3 s}.
\end{aligned} \tag{6.15}$$

Define $\underline{Y} = [X_1(s) \ldots X_4(s)]^T$ and $\underline{U} = [G(s)\ M_{1,\mathrm{des}}(s)\ M_{2,\mathrm{des}}(s)\ M_{3,\mathrm{des}}(s)]^T$. The transfer function matrix $H(s) = \underline{Y}(s)/\underline{U}(s)$ equals:

$$H(s) = \begin{pmatrix} \frac{1}{s} & -\frac{1}{s} & 0 & 0 \\ 0 & \frac{1}{s}\mathrm{e}^{-\tau_1 s} & -\frac{1}{s} & 0 \\ 0 & 0 & \frac{1}{s}\mathrm{e}^{-\tau_2 s} & -\frac{1}{s} \\ 0 & 0 & 0 & \frac{1}{s}\mathrm{e}^{-\tau_3 s} \end{pmatrix}. \tag{6.16}$$

Again, the exponential function has been approximated using second order Padé method. The Matlab routine `ss` computed a minimal realization of this transfer function matrix. The matrices form a controllable and observable system:

$$\begin{aligned}
A &= \begin{pmatrix}
-4.5 & 0 & 0 & 0 & -0.22 & -0.41 & 0.01 & 1.26 & 1.84 & -1.02 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -4.05 & 1.30 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -3.16 & -1.95 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.24 & 0 & -0.33 & 0.28 & -0.44 & -0.84 & 0 & 0.11 & 0.17 & -0.09 \\
0.44 & 0 & 0.18 & -0.15 & -0.55 & -1.13 & -0.38 & 0.26 & 0.39 & -0.21 \\
-0.01 & 0 & -0.04 & 0.04 & 1.15 & 1.78 & -1.53 & 0.20 & 0.29 & -0.16 \\
-1.90 & 0 & 0 & 0 & 0.17 & 0.32 & -0.01 & -1.00 & -1.46 & 0.80 \\
-1.63 & 0 & 0 & 0 & 0.31 & 0.58 & -0.02 & -1.78 & -2.60 & 1.43 \\
1.10 & 0 & 0 & 0 & 0.14 & 0.30 & 0.15 & 0.99 & 1.45 & -0.80
\end{pmatrix} \\
B &= \begin{pmatrix}
0 & -1.11 & 0 & 0 \\
-1 & 1 & 0 & 0 \\
0 & 0 & -1.23 & 0 \\
0 & 0 & 0.55 & 0 \\
0 & 0 & 1.87 & 0.39 \\
0 & 0 & -0.99 & 0.73 \\
0 & 0 & 0.25 & -0.02 \\
0 & 1.59 & 0 & 0.56 \\
0 & -1.08 & 0 & 0.83 \\
0 & 0 & 0 & 1.82
\end{pmatrix} \\
C &= \begin{pmatrix}
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.07 & 0 & 0.06 & -0.03 & 0.15 & 1.18 & -0.08 & -2.63 & -4.70 & 2.44 \\
0 & 0 & -1.07 & -2.21 & 0.30 & -0.34 & 0.05 & -0.11 & -0.17 & -0.36 \\
0 & 0 & 0 & 0 & 0.37 & -0.12 & -3.27 & 0.11 & 0.17 & 0.36
\end{pmatrix} \\
D &= \begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}.
\end{aligned} \tag{6.17}$$

The obtained state space model A, B, C, D has been reduced with the SLOCOT toolbox, using the SYSRED command. The model order of the reduced model has been set to 4, to validate the rule of thumb used in system identification. The reduced model A_{red}, B_{red}, C_{red}, D_{red} is then:

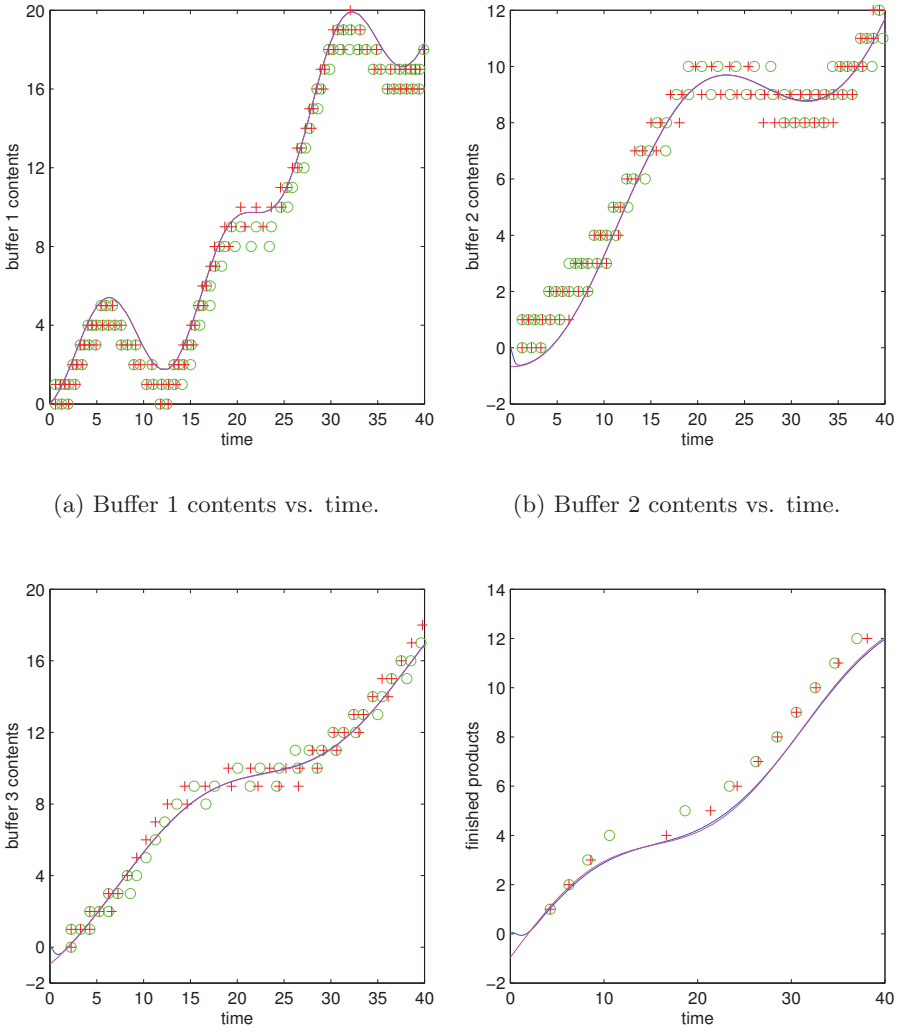
$$
\begin{aligned}
A &= 1.0 \cdot 10^{-15} \times \begin{pmatrix} -0.82 & 0 & 0 & -0.11 \\ 0 & 0.46 & -0.08 & -0.12 \\ 0 & 0 & -0.37 & -0.02 \\ 0 & 0 & 0 & 0.04 \end{pmatrix} \\
B &= \begin{pmatrix} 0.11 & 1.94 & -0.15 & 0.32 \\ 0.79 & -1.11 & 0.42 & 1.36 \\ -0.26 & 0.43 & 2.24 & 0.21 \\ -0.54 & 0.40 & -0.51 & 1.96 \end{pmatrix} \\
C &= \begin{pmatrix} 0.11 & 0.79 & -0.26 & -0.54 \\ 0.50 & -0.15 & -0.37 & 0.06 \\ -0.08 & -0.16 & 0.37 & -0.43 \\ 0.06 & 0.23 & 0.04 & 0.34 \end{pmatrix} \\
D &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -0.67 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -2 \end{pmatrix}.
\end{aligned}
\tag{6.18}
$$

The matrix A_{red} is almost zero. The time delay is not completely excluded from the model, but explicitly shows up in the D_{red} matrix. The diagonal elements equal the process times of the generator (0, no delay), machine M_1 (0.67), machine M_2 (1) and machine M_3 (0.50), with a minus sign. These diagonal elements are important, as can be seen in the simulation results in the next subsection.

6.5.4 Simulation

The discrete event model has been simulated with both lot releasing policies. The results are compared with the results from simulating the continuous approximation models. For clarity, only the first 40 time units have been plotted in Figure 6.9.

The figure shows that the continuous approximation models run behind the discrete event model. The reason for this is the same as in the previous section: the discrete event model seems to be internally controlled. Therefore, it is good to think about the fairness of comparing an open loop model to an internally controlled model.



(a) Buffer 1 contents vs. time.

(b) Buffer 2 contents vs. time.

(c) Buffer 3 contents vs. time.

(d) Buffer 4 contents vs. time.

Figure 6.9: Comparison of discrete event simulation to continuous approximation models. Green circles and red crosses are discrete event output with extreme strategy 1 and 2 respectively. Blue line is the minimal order model. Magenta line is the reduced order model.

6.5.5 Internal model control in continuous approximation

The continuous approximation seems not to fit well with the discrete event simulation output. In section 6.4.4 it had already been noticed that the discrete event model in fact is a closed loop model. In this section, the model structure is reconsidered. The continuous approximation model is extended to an internally controlled model.

Control goals

In future research, the continuous approximation models will be controlled with an external controller (Figure 1.2) that is still undetermined at the moment. But without realizing a controller, it is important to think about the control goals. Several possibilities exist. Two of them are:

- Smooth production at the last machine of a flowline and minimal buffers levels in all other buffers.
- All machines have to produce the same number of products.

The discrete event model uses the second option as the control goal: it compares the accumulated desired production with the actually production for all machines individually. If the first option is chosen as the control goal: one could imagine that problems may rise in case of machine breakdown: the downstream buffer becomes empty over time and once the machine has been repaired, the downstream buffer goal has been reached. The machine will not produce then. If the second control goal is used, the machine will try to catch up with the desired production after machine breakdown. This approach looks more promising than the first strategy. The continuous model is extended to a closed loop model.

Internal model control

To achieve this, the discrete event model is given a closer look. It is tried to copy the discrete event behavior as much as possible to the continuous model. The machine M in the discrete event model integrates the desired production rate, compares it to the actual production and then decides whether or not to produce items. This behavior can be copied to the continuous model, as schematically indicated in Figure 6.11. The determined state space model A, B, C, D is indicated as 'State Space Model'. The original inputs of the state space model, the generator rate and the machine production rates, are not directly connected to this state space model anymore. These input signals are shown at the left of the figure. They are integrated and compared to the actual machine production. Since the actual machine production was not an output of the state space model, 4 extra outputs are added to the state space model. This is explained further below. The difference between desired production and actual production is a

kind of new input. But this input cannot be connected to the state space model directly, since the state space model expects production rates (lots/time unit) and the difference between desired production and actual production is a real number of products. This is where the internal controller has to evolve. The deficit number of products must be translated into a workable production rate for the state space model. Since this controller is unknown yet, it is indicated as a question mark in Figure 6.11.

The (internal!) controller blocks indicated with a question mark are not trivial to determine. Again, the discrete event model is observed. The discrete event model uses an internal binary controller: produce or not produce. The decision is completely based on the difference between actual production and desired production (and possibly a decision strategy as explained before). Once the decision has been made to produce an item, it is completed at maximum speed. This binary control structure is shown in Figure 6.10 with the blue line. In a Simulink model, it is not difficult to implement this binary control structure. But implementing this non-linear element in the continuous approximation model is not desired. Reconsider the global framework of this research (Figure 6.12). To be able to use a wide variety of control techniques, a simple approximation model structure has to be chosen. A state space model is perfectly suitable for this. Therefore it is necessary to implement a linear internal controller in the continuous model. Only then, the complete internally controlled approximation model can be rewritten into a new state space model. As a first try, a linear proportional controller (P-control) is implemented (the red line in Figure 6.10). After an evaluation of the results, a PID-controller (proportional, integrative, derivative) is also implemented.

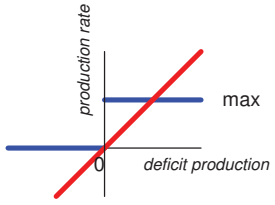


Figure 6.10: Binary control strategy in the discrete event model (blue) and linear approximation (red).

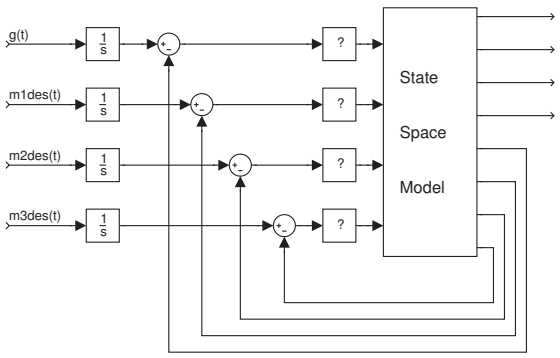


Figure 6.11: Schematic representation of the internally controlled continuous approximation model.

Four extra differential equations are added to the model dynamics. They represent the actual production of a machine. In fact, this is the amount of products a buffer is filled with. The 4 extra lines resemble the first 4 lines, only the negative terms are omitted. And because the last buffer with finished products is only filled, two lines in the transfer function matrix are identical, so one of them is omitted. This results in the following transfer function matrix:

$$H(s) = \begin{pmatrix} \frac{1}{s} & -\frac{1}{s} & 0 & 0 \\ 0 & \frac{1}{s}\mathrm{e}^{-\tau_1 s} & -\frac{1}{s} & 0 \\ 0 & 0 & \frac{1}{s}\mathrm{e}^{-\tau_2 s} & -\frac{1}{s} \\ 0 & 0 & 0 & \frac{1}{s}\mathrm{e}^{-\tau_3 s} \\ \frac{1}{s} & 0 & 0 & 0 \\ 0 & \frac{1}{s}\mathrm{e}^{-\tau_1 s} & 0 & 0 \\ 0 & 0 & \frac{1}{s}\mathrm{e}^{-\tau_2 s} & 0 \end{pmatrix}. \tag{6.19}$$

Making a minimal realization of this transfer function matrix yields a new A, B, C, D state space model. In fact, no extra dynamics were introduced while adding the 3 differential equations. The A matrix of the new state space model therefore has the same order as the old A-matrix and is a 10×10 matrix. The B matrix remains a 10×4 matrix. The C matrix is extended to give the 3 new outputs and is a 7×10 matrix now. The new D matrix is a 7×4 matrix containing zeros. The complete matrices are included in Appendix E.3.

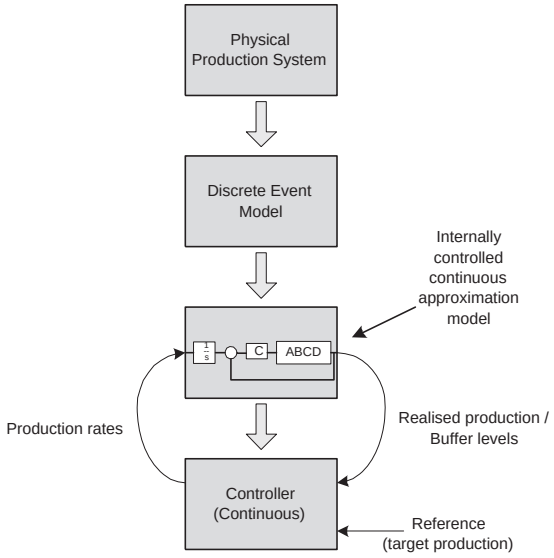


Figure 6.12: Research framework with internally controlled continuous approximation model.

Experiment with P-control

The 'extended state space model' has been simulated with Simulink and compared to the open loop results as simulated before. The Simulink model is shown in Figure 6.14. The same input signals as in section 6.5.1 have been used. The results of the discrete event model have been reused. The value of the proportional controller has been set to 0.7. The results of this simulation are only given for the number of finished products and given in Figure 6.13. The two extreme strategies in the lot processing policies are given again. Note that extreme strategy two matches the blue line, the target production, exactly. The closed loop internally proportionally controlled model (black) does not coincide with the target production. An oscillation can be observed, especially in the first 30 time steps. This looks like a gradually attenuated system. The P-value of the controller determines the damping ratio. A too large value of P causes the system to become unstable.

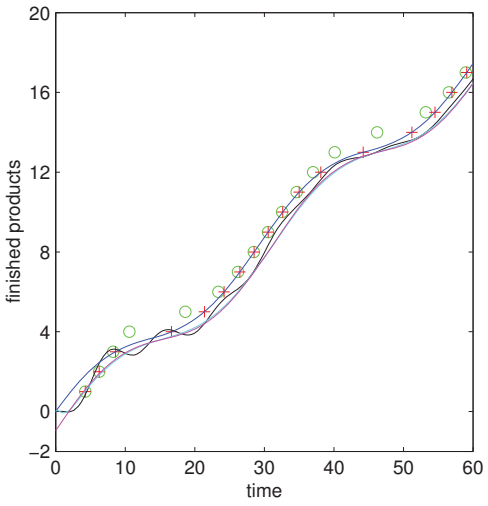


Figure 6.13: Simulation results of the model with P-control. Visible are: target production (blue), minimal realization continuous approximation model (cyan), reduced order model (magenta), closed loop P-controlled model (black) and discrete event simulation results (green and red).

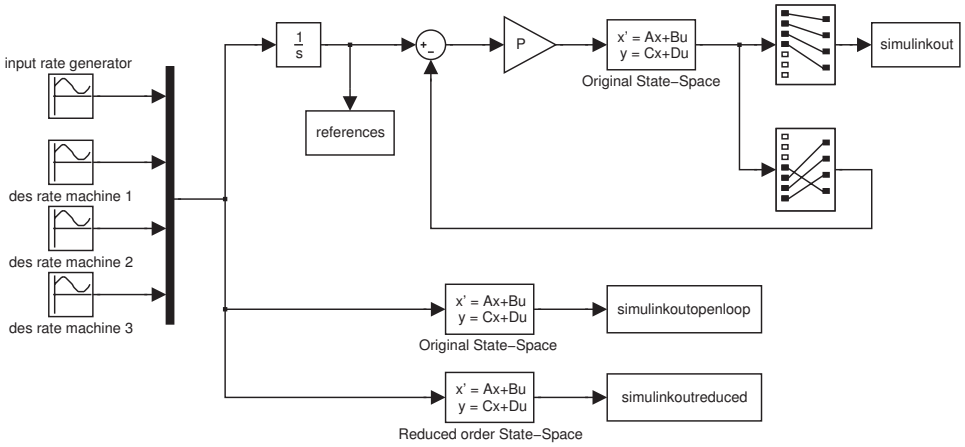


Figure 6.14: Simulink model of internally P-controlled state space model.

Experiment with PID-control

In the search for a better control structure, the controller block is extended to a PID-controller. This type of controller has a proportional part, integral part and derivative part and is a linear controller. This is important, because it leaves the possibility open to construct a state space model of the total continuous model block, as indicated in Figure 6.12. The Simulink model of this PID-controlled continuous model is shown in Figure 6.16. The controller block is shown in more detail in Figure 6.15. The four PID-blocks are standard Simulink blocks.

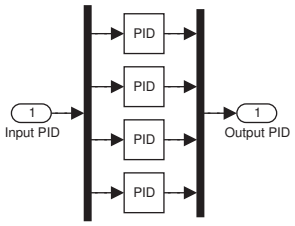


Figure 6.15: Controller block of the PID-controlled model.

The results of an experiment using the PID-controller ($P = 0.7$, $I = 0.1$, $D = 0.8$) are shown in Figure 6.17. The same input signals have been used as in the earlier experiments. Again, only the fourth output is shown, the other three are similar. The results are much better than in P-control. The controlled model output (black line) follows the target production line (blue) quite well. Again, a transient response is observable: the oscillation in the beginning of the simulation. A major advantage of this control method is the absence of a steady-state error: the controlled model practically follows the target production. In systems with more than 3 machines this is very important, since the approximation error between the open loop models and closed loop models will be significant then.

Tuning of the PID-parameters is important to achieve a good response of the model. Tuning has not been done extensively in this study, but by means of trial and error. A

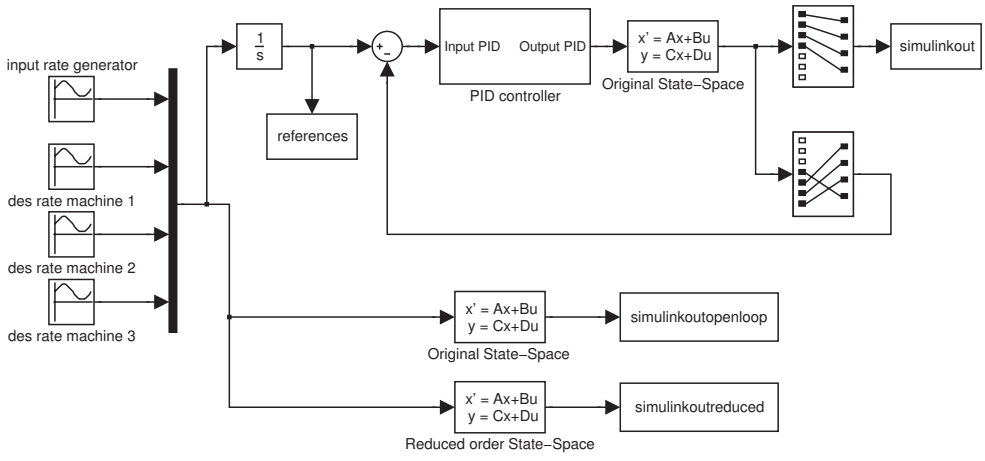


Figure 6.16: Simulink model of internally PID-controlled state space model.

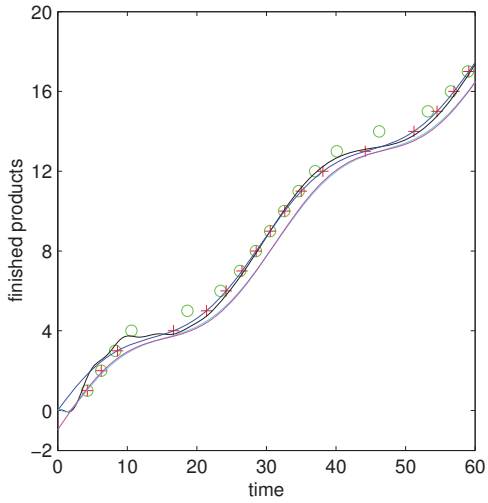


Figure 6.17: Simulation results of the model with PID-control. Visible are: target production (blue), minimal realization continuous approximation model (cyan), reduced order model (magenta), closed loop PID-controlled model (black) and discrete event simulation results (green and red).

proved method of tuning PID-controllers is the Ziegler-Nichols method. This could be used here. One should realize that different tuning of the PID-parameters does not give a better fitness to the discrete event extreme strategies output (the green and red circles and crosses in Figure 6.17. PID-control minimizes the steady state error. As the green circles are always in front of the target production, the different extreme strategies of product processing can not be approximated by different tuning of the PID-parameters.

If another linear controller is implemented, the extreme strategies of product processing by machines could be identified using system identification. Only the order of the linear control block can be chosen. One should realize that most disadvantages of the system identification toolbox hold here too.

The complete internally controlled continuous approximation model

All blocks in Figure 6.16 are linear. This means that an overall state space model of the complete internally controlled state space model can be constructed. This may not be very useful in the Simulink models, but an overall state space model can be used outside Simulink and is suitable in the controller design process of various control techniques. The resulting state space model can then be treated as a black box, with inputs U and outputs Y, as schematically presented in Figure 6.18. To construct the global state space model, the transfer functions matrices of the different blocks have been formulated. The transfer function matrix of the state space block A, B, C, D (Equation 6.19) has been split into two parts according to the different outputs (buffer levels and produced quantity). These parts are defined as H_1 and H_2. The transfer function of the PID-control block takes this form:

$$\mathrm{PID}(s) = \begin{pmatrix} P_1 + \frac{I_1}{s} + D_1 s & 0 & 0 & 0 \\ 0 & P_2 + \frac{I_2}{s} + D_2 s & 0 & 0 \\ 0 & 0 & P_3 + \frac{I_3}{s} + D_3 s & 0 \\ 0 & 0 & 0 & P_4 + \frac{I_4}{s} + D_4 s \end{pmatrix}. \tag{6.20}$$

The integration transfer function matrix can be written as:

$$\mathrm{Int}(s) = \begin{pmatrix} \frac{1}{s} & 0 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & 0 \\ 0 & 0 & \frac{1}{s} & 0 \\ 0 & 0 & 0 & \frac{1}{s} \end{pmatrix}. \tag{6.21}$$

The global transfer from inputs U to outputs Y is then stated as:

$$Y(s) = H_1(s)\mathrm{PID}(s)\left(I + H_2(s)\mathrm{PID}(s)\right)^{-1}\mathrm{Int}(s)U(s). \tag{6.22}$$

Since the transfer function matrices contain the Laplace variable s, straightforward computation of this total transfer function matrix is tricky. Pole-zero cancellation must be avoided. In the academic, deterministic and undisturbed case, these pole-zero cancellations will not cause damage, but when disturbances take place, these (bounded)

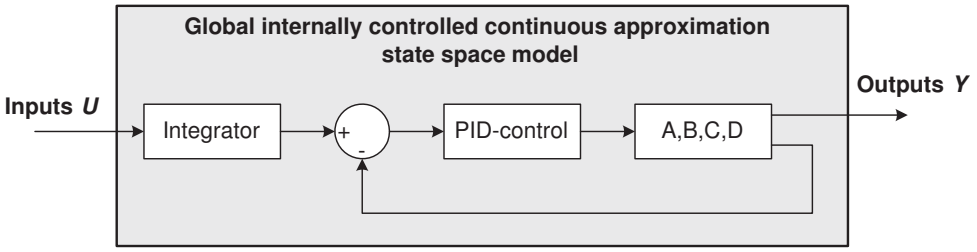


Figure 6.18: Global internally controlled state space model.

disturbances may cause the output Y to grow without bound. The complete transfer function matrix is too big to include in this report, but the structure can be given:

$$Y(s) = \begin{pmatrix} a & \sim b & 0 & 0 \\ 0 & b & \sim c & 0 \\ 0 & 0 & c & \sim d \\ 0 & 0 & 0 & d \end{pmatrix} U(s). \tag{6.23}$$

The notation $\sim x$ means that the matrix element is very similar to element x, but differs in plus/minus signs. If the extra outputs are desired (produced quantity per machine), the global transfer function matrix can be extended by:

$$Y_{\text{ext}}(s) = H_2(s)\text{PID}(s)\left(I + H_2(s)\text{PID}(s)\right)^{-1}\text{Int}(s)U(s). \tag{6.24}$$

Then, the transfer function matrix is a 7×4 matrix with this structure:

$$Y(s) = \begin{pmatrix} a & \sim b & 0 & 0 \\ 0 & b & \sim c & 0 \\ 0 & 0 & c & \sim d \\ 0 & 0 & 0 & d \\ a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \end{pmatrix} U(s). \tag{6.25}$$

Note that the virtual 8th row of the transfer function matrix (finished products) equals the 4th row and has thus been omitted. Converting the (non-extended) transfer function matrix into a minimal state space realization results in a 23th order state space model. The A, B, C, D matrices have not been included, but can be re-obtained by using the `ss` command in Matlab and the transfer function matrices as described in this section. The 23th order state space model contrasts sharply with the rule of thumb used in system identification, which would result in a 4th order model for this manufacturing line.

The new state space model eventually is an open loop model. The closed loop model developed in the previous sections has completely been included in the dynamics of the newly derived state space model. This open loop model has been simulated and compared to the closed loop simulation results. Results are not included in this report, because the open loop internally controlled model output complete matches the closed loop controlled model output.

6.6 Reflections on deriving state space models

The research in this chapter has been done to validate the rule of thumb stated in Chapter 5 that the number of states in a state space model describing a manufacturing flowline equals the number of outputs of the flowline. Outputs of interest always were buffer levels, so the number of buffers determined the state space model order. For

larger models with re-entrant behavior, this rule of thumb dit not hold. Therefore, extra research has been done.

A complete different modelling approach has been used in this chapter than in the previous ones. Using analytical equations and relations, transfer functions of continuous manufacturing systems have been formulated, taking process times of machines into account. Combining these transfer functions resulted in transfer function matrices, that have been translated into state space models using Matlab. The time delay due to the process times has been approximated with the Padé method.

The main conclusion reached in this chapter is that the discrete event models used in this research are in fact closed loop models. The discrete event model (reality) compares its actual production with a reference value, the desired production. Based on this comparison, an internal control algorithm decides whether or not to produce new items. This is a binary control algorithm in this research. The internal controller in the discrete event model can use different decision strategies. The goal of this project is to develop continuous approximation models which model output fit well with the discrete event model output. Therefore, it is absolutely necessary to develop internally controlled state space models. The results in this chapter look very promising with respect to this internally controlled continuous approximation models.

Both the internal model control and the process times of the machines introduce dynamics. These dynamics are included in the state space models. The order of the state space models will therefore always be greater than the number of outputs of the discrete event model. Model order reducing algorithms have been used to investigate the effect of choosing a smaller model order. For very small systems (like a GBME flowline) model order reduction leads to a model without time delay. For larger models (like a GBMBMBME flowline) the reduced order model introduces a non-zero D-matrix containing the process times reciprocals. The reduced order model output does not match the full order model output completely, but the overall trend is followed quite well.

The internal controller and process times are not the only reason why the model approach in system identification did not give good results. In Chapter 5 the machine production rate has been varied and fed into the state space model as an input signal. The process time of machines became a model input. This means that the time delay (dynamics!) due to the process time became dependant on the input signal. A basic principle of linear systems (like state space models) is that the system's dynamics are not dependant on the magnitude of the input signals. The only dynamics of a linear system are gain and phase as a function of the frequency of the input signal, not the magnitude of the input signal. The rule of thumb stated with system identification could not hold because wrong assumptions have been made.

The models developed in this chapter are only suitable in linear, continuous situations. The consequence for the discrete event model is that buffers must not become empty and machines are not to be over-utilized. An external supervisory process has to account for this. The question is to what extent the problems are passed to an external controller

in this linear model approach. Control techniques based on state space models that take bounded signals into account exist, like Model Predictive Control (MPC). This MPC technique is capable of bounding states, input signals and output signals. The limitations due to the linearity of the state space model might be taken care of with this type of controller.

Another question that arises is the need for an external controller to follow a reference production. With the open loop internally controlled state space approximation model, it is already possible to follow a reference target production for each machine. But maybe this can be achieved faster of more efficient with an external controller? One could also think of an open loop structure as shown in Figure 6.19. Customer orders are received and translated into a target production reference signal. This reference signal (desired production rates) can be feeded directly to the continuous approximation model. The translation block can take several forms, which is worth a whole new study.

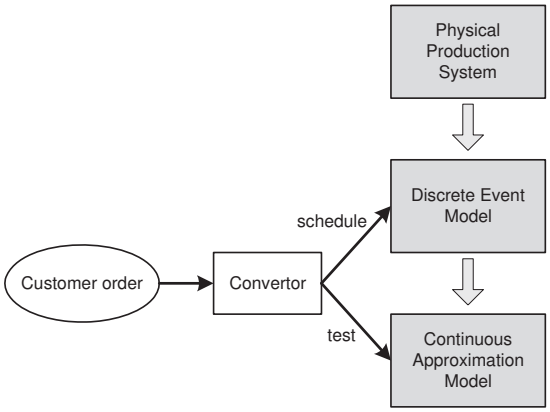


Figure 6.19: Does the need for an external controller still exist?

Chapter 7

Conclusions and recommendations for further research

7.1 Conclusions

The main goal of this research was to develop continuous state approximation models for discrete event manufacturing systems that incorporate machine process times. Three different modelling techniques have been used to develop these models. The major conclusion is that it is indeed possible to approximate a certain class of discrete event models by means of a continuous state model.

The three models that have been developed are:

- A hybrid model that uses a collection of state space models to approximate the discrete event system. Based on the state and boundary conditions, the appropriate state space model is constructed.
- A state space model obtained by means of system identification techniques. The state space matrices are obtained using a global 'black box' method.
- A state space model derived on an analytical basis. Two sub-models emerged: one the one hand a model based on transfer functions and on the other hand a state space model with the time delay due to process times explicitly included in the model's dynamics by means of a Padé approximation.

The three model types are evaluated here on compatibility with classical control techniques (the control technique must be capable of dealing with the model structure), computational effort, flexibility with respect to changes of the manufacturing system, handling boundary conditions and physical limitations.

Compatibility with classical control techniques

For MIMO systems (multi-input, multi-output), the far most convenient model structure to develop a controller is a state space model. All three model structures result in state space models. However, the hybrid model delivers a set of state space models. Hybrid models limit the number number of control techniques that can be used. Different hybrid control techniques exist, but are not elaborated here. The quickly growing number of possible state space models due to expanding the discrete event model could be a restriction in convenient use of this hybrid model.
The state space models obtained by both system identification and the analytical method are far more suitable for classical control, since it exists of only one state space model. It is very important to realize that the models obtained by system identification can only be used within a predefined bandwidth, which is a great disadvantage.

Computational effort

An important aspect in deriving continuous approximation models is computational effort. The hybrid model is a generic model. Once the number of products, number of machines, product recipes and system's limitations have been entered, the hybrid model is ready to be used. But controller determination can be more difficult. The complete set of all possible state space models does not have to be predetermined, because the hybrid model algorithm constructs the matrices 'on the fly'. In contrast with this, controller computation can most probably not be done in a real-time environment, since it is too time-consuming. The controllers will most likely have to be determined beforehand. This makes it necessary to build the set of possible state space models beforehand, which will be very time consuming, if possible at all.
System identification is also very time consuming. In order to be able to use a considerable bandwidth, a lot of frequencies must be included into the input signals, leading to huge data-files and very long computations. The resulting state space model is not always a valid model. That is the reason why another modelling method has been elaborated.
The analytical method requires very little computational power. Formulating the transfer function matrices is the most time consuming part. But if used more widely, this aspect can easily be automated.
Almost all simulation models in this research used a timestep for simulations. The magnitude of the timestep has been chosen freely and has not been varied. It was chosen in a way that it was much smaller than the other parameters. However, the smaller the timestep, the longer a simulation takes and the more accurate the results will be. The computational effort is therefore partially influenced by the timestep that is chosen.

Flexibility with respect to changes of the manufacturing system

In case the manufacturing system is modified, the continuous models have to change too. New equipment, new products or changes in the product recipes cause the necessity to develop new continuous approximation models. The hybrid model is perfectly suited for model changes: all global constants are predefined and the algorithm deals with the actual modelling process. System identification is highly inflexible. Every change in the manufacturing system requires a complete new identification. This is a disadvantage of the 'black box' approach of system identification. Even new low-level scheduling rules of a machine may cause the need for a new complete identification, since those scheduling rules are included in the state space models. The analytical models using transfer functions are not very flexible at the moment. But since the computation order is always the same, this can be automated, yielding a flexible model structure. Changes in low-level scheduling rules do not require a new model, because these scheduling rules are not implemented in the model.

Boundary conditions and physical limitations

Machines have a finite capacity and buffers can not become negative in the real manufacturing system. Those limitations and boundary conditions must be accounted for in either the continuous approximation model or the controller that is to be developed. The hybrid model scores best at this point: buffers can not become empty, machines have finite capacity and if a controller asks more than the machines can produce, the desired production rates are scaled down to workable production rates. In future, this model could be extended to buffer upper limits and batch production. The price to be paid is the enormous collection of state space models that makes this behavior possible. The model obtained by system identification has most limitations. First of all, the restricted bandwidth is a big problem. Constant input signals are not allowed, since good behavior is not guaranteed then. It does neither deal with finite machine capacity nor with buffer lower limits. The analytical model itself does not deal with finite machine capacity or buffer levels. A future controller will have to deal with this. But a great advantage is that constant input signals are allowed, since no limited bandwidth exists. The state limits may only apply to the analytical model based on transfer functions. The Padé approximations cause the state to become negative for a short while before the state rises positive (see Figure 6.7). Padé approximations thus might be a difficulty in using state limitations.

The models obtained by system identification did not all give good results. The rule of thumb used to choose the model order was: "The number of states in the state space description equals the number of measured outputs of the discrete event system." This proved to be somewhat optimistic. The actual needed number of states is larger, to be able to include the time delay approximations (Padé method) in the

model dynamics. However, a larger model order is not workable due to the enormous data-files needed and large computation time.
The remaining two models (hybrid and analytical based on transfer functions) look promising with respect to the results obtained in this study. Both models can be used in further research. This is elaborated in the next section.

Another very important conclusion in this research is that the discrete event models that are to be controlled are in fact internally controlled models. In the global framework (Figure 1.2), the left conversion block that translates desired production rates into events contains the internal controller. Based on the information from the controller and the actual machine production, it has to decide whether or not to produce a next item. This is closed loop control. A good approximation model also contains this internal closed loop to get maximum similarity in the comparison between continuous model and discrete event model. The hybrid model does not contain this internal closed loop. The latent internal controller had only been discovered during the development of the analytical model based on transfer functions.

What was new in this study? Most important new thing in this research has been the inclusion of machine process times in the dynamic models. The Kimemia and Gershwin model [KG83] has been extended to a model that deals with time delays due to process times.

The framework presented in the introductory chapter has not been fully covered in this study. Consider again the framework in Figure 7.1. The light orange blocks have been investigated. The physical manufacturing system had been assumed to equal the discrete event system. The controller block has not been elaborated, although during the model development process, a easy connection (compatibility) between continuous model and controller has been in the back of mind.

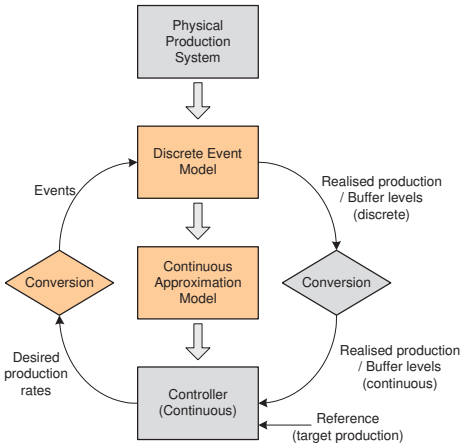


Figure 7.1: Global internally controlled state space model.

7.2 Recommendations and reflections for further research

Although the results of this research are quite satisfactory, the study is not yet complete. Three modelling techniques have been explored, but not elaborated very thoroughly. Two methods look promising one the one hand, on the other the system identification method seems not useful in modelling discrete event manufacturing systems. The obtained hybrid model and analytical model must be put to the test extensively. Possibly, benchmark problems can be tackled with the new modelling techniques. Moreover, extensively testing could reveal new interesting modelling issues. Both the hybrid model and the analytical model based on transfer functions are worth to be elaborated deeply. For both modelling methods, some recommendations are given here.

Hybrid model

The power of the hybrid model is the ability to deal with physical limitations and boundary conditions. Most important recommendation is to investigate whether or not the internal control loop can be included in the model. Only while developing the last modelling method, this internal closed loop has been discovered. Most discrete event models used for the hybrid model were in fact closed loop models. It is a challenge to include this internal controller in the hybrid model. Another recommendation is elaborating the ability to deal with physical limitations. At the moment, the model knows that buffers can not become empty and that machines can not be over-asked. Another limitation could be a buffer ceiling: a machine can not produce if the downstream buffer has reached its maximum capacity. In addition to this, batch processing can be explored. In the current model, a machine is only allowed to produce if the preceding buffer contains at least one product. One can imagine that a kind of batch processing is modelled when this threshold level is raised.
In addition to this, manufacturing lines with assembly or disassembly can be investigated. Is it possible to use the hybrid model for this? Adding some logic rules to the algorithm could be a good option for this. These logic rules are based on the product's recipe. This kind of behavior will be new in the hybrid model algorithm.
Only having a good model is not enough. The main goal is to obtain models that can be used for control. The hybrid model needs special attention when it comes to control. The number of $\dot{x} = Bu$ models grows rapidly when extending the manufacturing system. Control theorists might assist in choosing and developing a controller for the hybrid model.

Analytical model based on transfer functions

The analytical model based on transfer functions is a ready to use model for simple small manufacturing lines. However, this only holds for the continuous case. If discontinuities occur, the model predicts differently (the continuous model simulation

output no longer looks like the discrete event model simulation output). The main shortcoming of the model is dealing with these discontinuities. Several options exist. Because the linear state space model can not deal with limitations, a controller needs to be modelled. This can be either an open loop controller or a closed loop controller. An open loop controller only has to translate customer orders into workable machine production rates. The internal closed loop controller takes care of the production simulations. The continuous approximation model has completely been decoupled from the manufacturing system then. But this would also be the case if closed loop control would be applied to the discrete event system. Another possibility is using a control technique that is capable of handling state limits and input signal limits. An example is Model Predictive Control (MPC). This control technique predicts the model's behavior subject to computed input signals. If state limits are exceeded, the input signal will be adjusted. The input signal can also be bounded. The MPC technique works with state space models, so that will be no problem.
Introduction of batch processing can be difficult in this model type, but maybe with a new analytical basis, this can be added to the model. Assembly and disassembly in the manufacturing line is another research topic for this model type.

A recommendation for both models is the introduction of stochastic behavior. Machine process times are not constant in a real production facility. The real process time is not always known beforehand. How does this affect the model structures? In a state space model, this varying process time can not be introduced as an extra input, since the model dynamics can not depend on the magnitude of an input signal. One of the questions that arises is whether the model should be based on the mean process time or another value for safety reasons? Introduction of stochastics will be a considerable research issue.

A topic that is not covered in this study is the applicability of the models. The initial assumption was that the models would only be valid in bulk production, with large number of products. The product stream looks like a flow then, like cigarettes, candy-bars or even wafers. The example models in this study were however small scale models. The obtained state space models might be a representation of small scale manufacturing lines too. The applicability of the model is a very different but important aspect of the global research. Exploring the borders of the research field is an interesting and valuable new topic.

As indicated in the conclusions at the beginning of this chapter, the timestep that is chosen for simulations has an influence on the accuracy and the computational effort. In a future research, the magnitude of this influence could be investigated. This investigation could lead to more justified choices for the timesteps.

As a final recommendation, the global framework of this research could be completed with the development of a controller and the right conversion block of the framework (Figure 1.2). First of all, investigation of the need for those blocks is necessary. The

internally controlled continuous approximation models perform very well in tracking a target production. A well determined target production function will result in good behavior of the discrete event model. An open loop controller that converts customer orders into a workable target production might do well. This must be investigated. If it is necessary to reconsider the global research framework, then that has to be done. On the other hand, if a closed loop controller seems to be necessary, completion of the modelling and controlling part of the framework will be a considerable amount of work. But without a controller (either open loop or closed loop), the research will not have been completed.

Bibliography

[BD97] J. Banks and J. Dai. Simulation studies of multiclass queueing networks. *IIE Transactions*, 29:213–219, 1997.

[BMS+99] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. Slicot — a subroutine library in systems and control theory. *Applied and Computational Control, Signal and Circuits*, 1(10):499–539, 1999.

[BT97] C. Bispo and S. Tayur. *Managing simple re-entrant flow lines: I. A theoretical framework*. GSIA Working paper, 1997.

[Cam01] E.J.J. van Campen. *Design of a multi-process multi-product wafer fab*. PhD Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, April 2001.

[Dia96] B. Diamond. Concepts of modelling and simulation. Internal paper of Imagine That, Inc. www.imaginethatinc.com, 1996.

[DW96] J. Dai and G. Weiss. Stability and instability of fluid models for re-entrant lines. *Mathematics of Operations Research*, 21:115–134, 1996.

[DW99] J. Dai and G.E. Weiss. A fluid heuristic for minimizing makespan in jobshops, 1999. Preprint, pages 1–35.

[KG83] J. Kimemia and S.B. Gershwin. An algorithm for the computer control of a flexible manufacturing system. *IIE Transactions*, 15(4):353–362, December 1983.

[Kum93] P.R. Kumar. Re-entrant lines. *Queueing Systems*, 13:87–110, 1993.

[LK89] S. Lou and P. Kager. A robust production control policy for VLSI wafer fabrication. *IEEE Transactions on Semiconductor Manufacturing*, 2(4):159–164, 1989.

[LMG+96] H. Lieu, H. Mahmassani, N.H. Gartner, C. Messer, and A.K. Rathi. *Traffic flow theory — A state of the art report*. Turner-Fairbank Highway Research Center, 1976, revised in 1996. http://www.tfhrc.gov/its/tft/tft.htm.

[Lon98] R.G. Longoria. Runge-Kutta 4 fixed step ode-solver, 1998. Department of Mechanical Engineering, University of Texas, Austin. http://www.me.utexas.edu/ lotario/me244L/labs/simulation/simover.html.

[LW55] M.H. Lighthill and G.B. Whitham. On kinematic waves — II. A theory of traffic flow on long crowded roads. In *Proceedings Royal Society London*, volume A229, pages 317–345, 1955.

[LYS+91] S. Lou, H. Yan, S. Sethi, A. Gardel, and P. Deostahli. Using simulation to test the robustness of various existing production control policies. In *Proceedings of the 1991 Winter Simulation Conference*, pages 261–269, 1991.

[Nij00] H. Nijmeijer. Observer design: A partial survey and recent developments, 2000. 50th Canadian Chemical Engineering conference, Montreal.

[Roo95] J.E. Rooda. *The modelling of industrial systems.* Technische Universiteit Eindhoven, 1995. Lecture notes on the course 4C530: Modelling of industrial systems.

[ULMV92] R. Uzsoy, C.Y. Lee, and L.A. Martin-Vega. A review of production planning and scheduling models in the semiconductor industry, part I. *IIE Transactions*, 24(4):47–60, 1992.

[ZCB01] G. Zames, C.D. Charalambous, and B. Boulet. *Linear Systems.* McGill University, Montreal, Canada, 2001. Lecture notes on the course ECSE-501A: Linear Systems.

Appendix A

Hybrid flowline models

A.1 GBMBME flowline model

```
proc G (a: !nat, u: real) =
|[ x: nat
 | x:=1
 ; *[ time < 5 -> delta 1/u; a!x; x:= x + 1 ]
 ; terminate
]|

proc B (a: ?nat, b: !nat) =
|[ x: nat, xs: nat*
 | xs := []
 ; *[ true;        a?x      -> xs := xs++[x]
    | len(xs) > 0; b!hd(xs) -> xs := tl(xs)
    ]
]|

proc M (a: ?nat, b: !nat, m: real) =
|[  x: nat
 | *[ true -> a?x; delta 1/m; b!x ]
]|

proc E (a: ?nat) =
|[ x: nat
 |*[ true -> a?x ]
]|

syst Model() =
|[ a, b, c, d, e: -nat
 | G( a, 5.0 ) || B( a, b ) || M( b, c, 3.0 ) || B( c, d ) || M( d, e, 4.0 ) || E( e )
]|

xper =
|[ Model() ]|
```

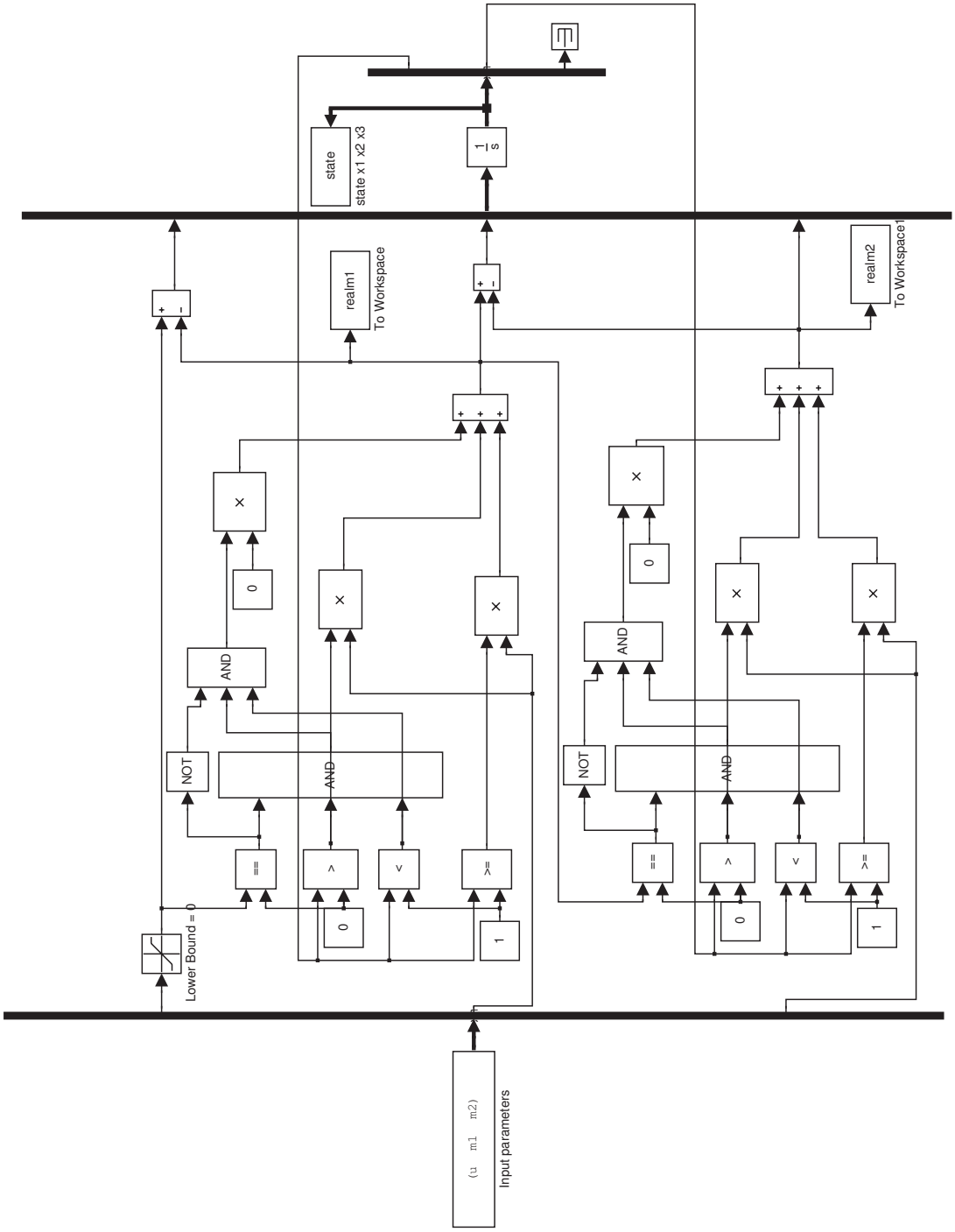


Figure A.1: Simulink model of the GBMBME-flowline.

A.2 GBMBME flowline model for 2 product types

```
type loggen = real#nat#nat

proc G (a: !nat, u: real) =
|[  x: nat
 |  x:=1
 ;  *[ true -> delta 1/u; a!x; x:= x+1 ]
]|

proc B (a: ?nat, b: !nat, cont1, cont2: !nat, id: nat, l: !loggen) =
|[ x: nat, xs: nat*
 | xs := []
 ; *[ true;        a?x           -> xs := xs++[x]; l!<time, id,len(xs)>
    | len(xs) > 0; b!hd(xs)      -> xs := tl(xs);  l!<time, id,len(xs)>
    | true;        cont1!len(xs) -> skip
    | true;        cont2!len(xs) -> skip
    ]
]|

proc M (in1,in2: ?nat, out1,out2: !nat, buf1,buf2: ?nat,
                       cumul1,cumul2: ?real, m1,m2,timestep:real) =
|[ c1, c2, prod1, prod2, diff1, diff2: real, x, choice: nat
 , cont1, cont2: nat, diffs: real*, conts: nat*
 | prod1:=0.0; prod2:=0.0
 ; *[ true -> cumul1?c1
             ; cumul2?c2
             ; buf1?cont1
             ; buf2?cont2
             ; diff1:=(c1-prod1)/c1
             ; diff2:=(c2-prod2)/c2
             ; diffs:=[diff1] ++ [diff2]
             ; conts:=[cont1] ++ [cont2]
             ; choice:=decision(diffs, conts)
             ; [ choice = 1 -> in1?x; delta 1/m1; prod1:=prod1+1; out1!x
               | choice = 2 -> in2?x; delta 1/m2; prod2:=prod2+1; out2!x
               | choice = 0 -> delta timestep
               ]
    ]
]|

func decision(diffs: real*, conts: nat*) -> nat =
|[ i, number, index: nat, record: real, c: bool
 | i:=1; number:=len(diffs); record:=0.0; index:=0
 ; *[ i <= number -> c:= hd(diffs) > 0 and hd(conts) > 0 and hd(diffs) > record
                   ; [ c     -> index:=i
                                ; record:=hd(diffs)
                     | not c -> skip
                     ]
                   ; diffs:=tl(diffs)
                   ; conts:=tl(conts)
                   ; i:=i+1
    ]
```

```
 ; ret index
]|

proc MC (m1, m2: ?real, m1c, m2c: !real, timestep: real)=
|[ m1r, m2r, c1, c2: real
 | c1:=0.0000001; c2:=0.0000001  // prevent from dividing by zero
 ; *[ m1?m1r -> c1:=c1+m1r*timestep
    | m2?m2r -> c2:=c2+m2r*timestep
    | m1c!c1 -> skip
    | m2c!c2 -> skip
    ]
]|

func calc(mr: real, b: nat, mm: real) -> real =
|[ [ mr /= 0 and b <  1 -> ret 0.0
   | mr /= 0 and b >= 1 -> ret mm
   | mr =  0 and b >  0 -> ret mm
   | mr =  0 and b =  0 -> ret 0.0
   ]
]|

proc C(bi11, bi12, bi21, bi22: ?nat, m11, m12, m21, m22: !real,
       m11max, m12max, m21max, m22max, m11d, m12d, m21d, m22d, u1, u2, timestep: real)=
|[ egdm11m, egdm12m, egdm21m, egdm22m, m11m, m12m, m21m, m22m, m11r, m12r, m21r, m22r: real,
   b11,b12,b21,b22: nat, m11f, m12f, m21f, m22f: nat
 | *[ true -> bi11?b11 ; bi12?b12 ; bi21?b21 ; bi22?b22
            ; m21m:=calc(m11r, b21, m21max)
            ; m22m:=calc(m12r, b22, m22max)
            ; m11m:=calc(u1, b11, m11max)
            ; m12m:=calc(u2, b12, m12max)
            ; [ m11m =  0 -> egdm11m:=0.0; m11f:=0
              | m11m /= 0 -> egdm11m:=1/m11m; m11f:=1
              ]
            ; [ m12m =  0 -> egdm12m:=0.0; m12f:=0
              | m12m /= 0 -> egdm12m:=1/m12m; m12f:=1
              ]
            ; [ m21m =  0 -> egdm21m:=0.0; m21f:=0
              | m21m /= 0 -> egdm21m:=1/m21m; m21f:=1
              ]
            ; [ m22m =  0 -> egdm22m:=0.0; m22f:=0
              | m22m /= 0 -> egdm22m:=1/m22m; m22f:=1
              ]
            ; m11r:=m11d/( max(1.0,(m11d*egdm11m+m12d*egdm12m)))*m11f
            ; m12r:=m12d/( max(1.0,(m11d*egdm11m+m12d*egdm12m)))*m12f
            ; m21r:=m21d/( max(1.0,(m21d*egdm21m+m22d*egdm22m)))*m21f
            ; m22r:=m22d/( max(1.0,(m21d*egdm21m+m22d*egdm22m)))*m22f
            ; m11!m11r ; m12!m12r ; m21!m21r ; m22!m22r
            ; delta timestep
    ]
]|
```

```
proc E (a: ?nat, id: nat, l: !loggen) =
|[ x: nat
 | *[ true -> a?x; l!<time,id,x> ]
]|

proc Log(l: (?loggen)^6, output: !file) =
|[ tend: real
 , x: loggen
 | tend:= 10.0                          // end of simulation and logging
 ; *[ time < tend
      -> [ j: nat <- 0..6: true; l.j?x ]
       ; output! x.0, tab(), x.1, tab(), x.2, nl()
    ]
 ; terminate
]|

syst Model (u1, u2, m11max, m12max, m21max, m22max,
            m11des, m12des, m21des, m22des, timestep: real)=
|[  l: (-loggen)^6
 ,  bc11, bc12, bc21, bc22, g1, g2, b1m1, bi11, b2m1, bi12
 ,  m1b1, m1b2, b1m2, bi21, b2m2, bi22, e1, e2: -nat
 ,  cmc11, cmc12, cmc21, cmc22, mcm11, mcm12, mcm21, mcm22: -real
 |  G(g1, u1) || G(g2, u2)
 || B(g1, b1m1, bi11, bc11, 1, l.0) || B(g2, b2m1, bi12, bc12, 2, l.1)
 || M(b1m1, b2m1, m1b1, m1b2, bi11, bi12, mcm11, mcm12, m11max, m12max, timestep)
 || B(m1b1, b1m2, bi21, bc21, 3, l.2) || B(m1b2, b2m2, bi22, bc22, 4, l.3)
 || M(b1m2, b2m2, e1, e2, bi21, bi22, mcm21, mcm22, m21max, m22max, timestep)
 || E(e1, 5, l.4) || E(e2, 6, l.5)
 || MC(cmc11, cmc12, mcm11, mcm12, timestep) || MC(cmc21, cmc22, mcm21, mcm22, timestep)
 || C(bc11, bc12, bc21, bc22, cmc11, cmc12, cmc21, cmc22, m11max, m12max, m21max, m22max,
       m11des, m12des, m21des, m22des, u1, u2, timestep)
 || Log(l, fileout("uitvoer2.txt"))
]|

xper =
|[ Model( 5.0, 5.0, 4.0,   6.0,   7.0,   3.0,   8.0,   5.0,   2.0,   5.0,   0.01 ) ]|
//        u1,  u2,  m11max,m12max,m21max,m22max,m11des,m12des,m21des,m22des,timestep
```

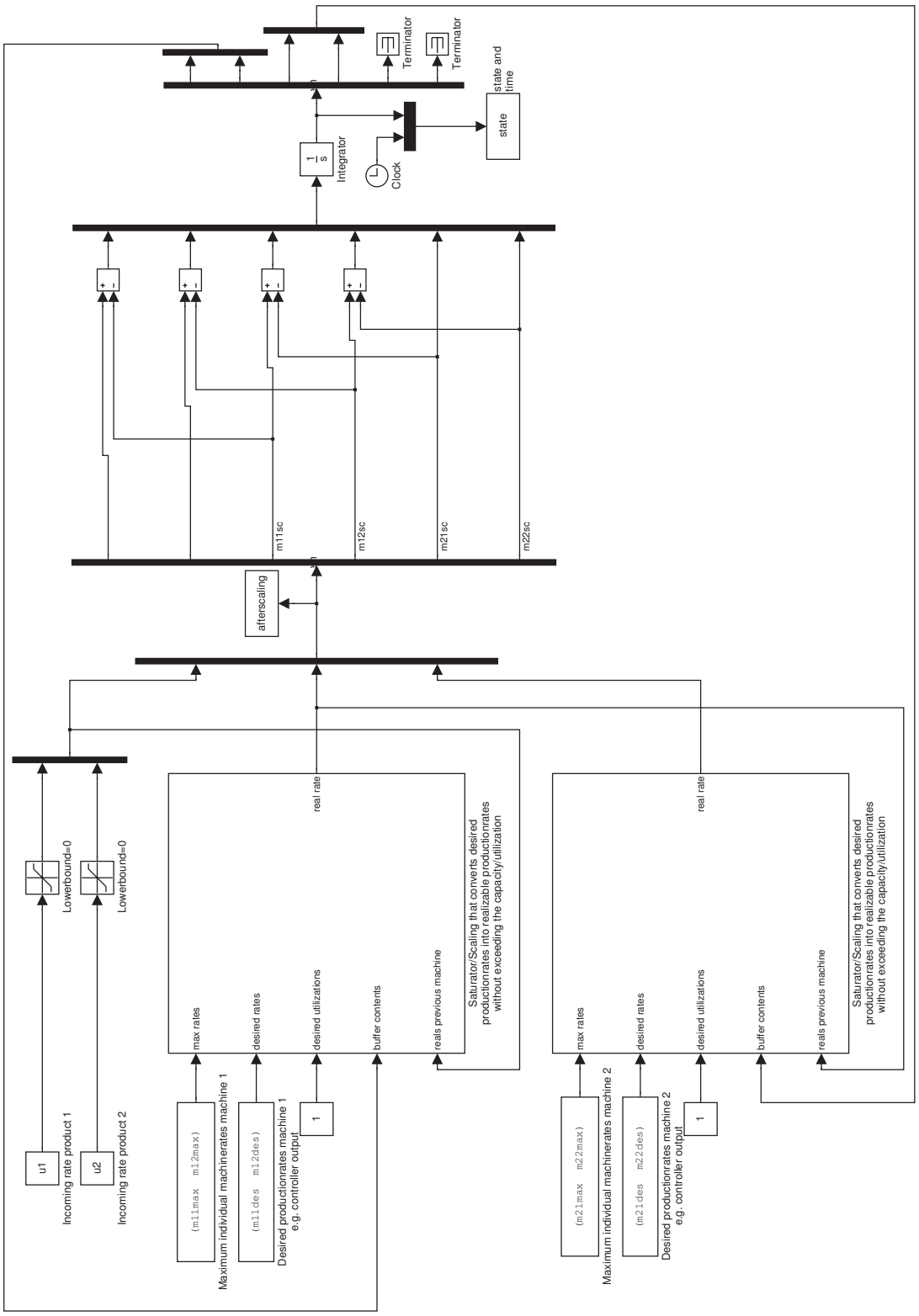


Figure A.2: Simulink model of the GBMBME-flowline with 2 different product types.

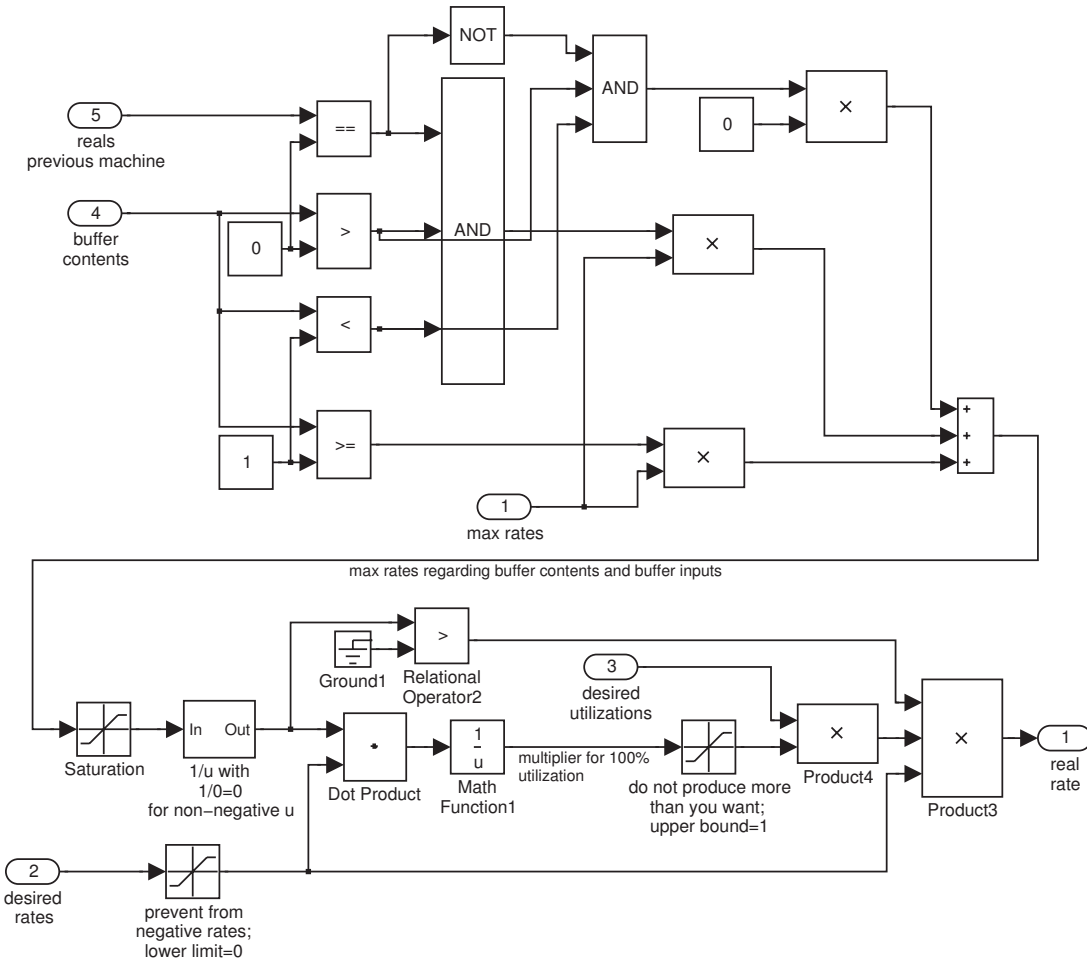


Figure A.3: Simulink model of the saturator/scaling process that computes real production rates for a machine without exceeding the maximum machine capacity and regarding the buffer contents and buffer inputs.

Appendix B

Matlab hybrid model

```
clear
clc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             Plant configuration               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numberofproducts=5;
numberofmachines=5;
machineorder=[1 2 3 4 5 0 0 0 0 0;
              5 4 3 2 1 0 0 0 0 0;
              1 2 3 2 3 4 3 4 5 0;
              1 4 3 4 3 2 3 2 5 0;
              3 4 3 4 1 2 1 2 3 5];
% Computation of several matrices and rows based on the plant configuration
[numberofstates, numberofproductionsteps, numberpermachine, ...
cumulnumberpermachine, numberofstepsperproduct, matrix, previous]=...
computeplant(numberofproducts, numberofmachines, machineorder);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulationtime, interval and initial conditions %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Tspan=[0 20];   % Simulation start- and endtime
N=5000;         % Number of simulation steps
X0=zeros(43,1); % Initial condition state
memory=5;       % Number of timesteps a machine should not produce before
                % the next machine may produce

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     START OF MAXIMUM AND DESIRED RATES DEFINITIONS                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Incoming product rates
u=[5; 5; 5; 5; 5];

% Syntax: machine A product 1 step 1, machine A product 2 step 5, machine A product 3 step 1 etc.
```

```
% Idem for machine B, C, D and E
mamax=[5 5 5 5 5 5];
mbmax=[5 5 5 5 5 5 5 5];
mcmax=[5 5 5 5 5 5 5 5 5 5 5];
mdmax=[5 5 5 5 5 5 5 5];
memax=[5 5 5 5 5];

% Syntax: machine A product 1 step 1, machine A product 2 step 5, machine A product 3 step 1 etc.
% Idem for machine B, C, D and E
mades=[3 3 3 3 3 3];
mbdes=[3 3 3 3 3 3 3 3];
mcdes=[3 3 3 3 3 3 3 3 3 3 3];
mddes=[3 3 3 3 3 3 3 3];
medes=[3 3 3 3 3];

reala=zeros(6,1);
realb=zeros(8,1);
realc=zeros(11,1);
reald=zeros(8,1);
reale=zeros(5,1);
real=[u; reala; realb; realc; reald; reale];

% Desired utilizations
utila=1;
utilb=1;
utilc=1;
utild=1;
utile=1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         END OF MAXIMUM AND DESIRED RATES DEFINITIONS         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Start of the Computation of the Runge Kutta Method      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h = (Tspan(2)-Tspan(1))/N;
halfh = 0.5*h;
neqs=size(X0);
X=zeros(neqs(1),N+1);
T=zeros(1,N+1);
X(:,1)=X0;
T(1)=Tspan(1);
Td = Tspan(1);
Xd = X0;
Reals=[];
Realprevs=[];
wacht = waitbar(0,'Progress of the computation');
for i=2:N+1
  waitbar(i/(N+1),wacht);
  % DETERMINE BUFFER CONTENTS
  [bca, bcb, bcc, bcd, bce]=determinebc(Xd);
  % TRANSPOSE REALS TO CREATE REALPREVS
  realprev=previous*real;
```

```
  Realprevs=[Realprevs realprev];
  [realpreva, realprevb, realprevc, realprevd, realpreve]=determineprev(realprev);
  [boola, boolb, boolc, boold, boole]=determinebools(Realprevs, memory);
  % START OF REAL RATES COMPUTATION
  reala=computereal(mamax, mades, bca, utila, realpreva, boola)';
  realb=computereal(mbmax, mbdes, bcb, utilb, realprevb, boolb)';
  realc=computereal(mcmax, mcdes, bcc, utilc, realprevc, boolc)';
  reald=computereal(mdmax, mddes, bcd, utild, realprevd, boold)';
  reale=computereal(memax, medes, bce, utile, realpreve, boole)';
  real=[u; reala; realb; realc; reald; reale];
  % END OF REAL RATES COMPUTATION
  RK=matrix*real;
  X(:,i)=Xd+RK*h;
  T(i)=Td+h;
  Xd = X(:,i);
  Reals=[Reals real];
  Td = T(i);
end
close(wacht);
X=X';T=T';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [numberofstates, numberofproductionsteps, numberpermachine, ...
         cumulnumberpermachine, numberofstepsperproduct, matrix, previous]=...
         computeplant(numberofproducts, numberofmachines, machineorder)

     [i,j]=find(machineorder~=0);
numberofstates=numberofproducts+size(i,1);
numberofproductionsteps=size(i,1);
for i=1:numberofmachines
    numberpermachine(i)=size(find(machineorder==i),1);
end
cumulnumberpermachine(1)=0;
for i=1:numberofmachines-1
    cumulnumberpermachine(i+1)=cumulnumberpermachine(i)+numberpermachine(i);
end
for i=1:numberofproducts
    numberofstepsperproduct(i)=size(find(machineorder(i,:)),2);
end
matrix=[zeros(numberofproductionsteps,numberofproducts) -1*eye(numberofproductionsteps);
        zeros(numberofproducts, numberofproducts) zeros(numberofproducts, numberofproductionsteps)];
previous=matrix; % for now!
rij=1;
for i=1:numberofmachines
    for j=1:numberofproducts
        for k=1:numberofstepsperproduct(j)
            if machineorder(j,k)==i
                if k==1
                    element=j;
                    matrix(rij,element)=1;
                    rij=rij+1;
                else
                    vorige=machineorder(j,k-1);
```

```
                    teller=0;
                    for m=1:j
                        for n=1:numberofstepsperproduct(m)
                            if machineorder(m,n)==vorige & ~(j==m & n>=k)
                                teller=teller+1;
                            end
                        end
                    end
                    element=numberofproducts+cumulnumberpermachine(vorige)+teller;
                    matrix(rij,element)=1;
                    rij=rij+1;
                end
            end
            if k==numberofstepsperproduct(j)
                laatste=machineorder(j,k);
                teller=0;
                for m=1:j
                        for n=1:numberofstepsperproduct(m)
                            if machineorder(m,n)==laatste & ~(j==m & n>=k)
                                teller=teller+1;
                            end
                        end
                    end
                element=numberofproducts+cumulnumberpermachine(laatste)+teller+1;
                matrix(numberofproductionsteps+j,element)=1;
            end
        end
    end
end
previous=matrix-previous; % for ever!


function real=computereal(mmax, mdes, bc, util, realprev, bool)
temp1=mmax'.*(bc>=1)+mmax'.*(realprev==0 & bc>0 & bc<1 & bool);
temp5=max(0, temp1);
temp4=max(0, 1./(temp5-(temp5==0)))';
temp2=util*min(1, 1./(temp4*max(0,mdes)'-((temp4*max(0,mdes)')==0)));
real=  max(0,mdes) .* temp2 .* (temp4>0);


function [bca, bcb, bcc, bcd, bce]=determinebc(Xd)
bca=Xd(1:6);
bcb=Xd(7:14);
bcc=Xd(15:25);
bcd=Xd(26:33);
bce=Xd(34:38);


function [boola, boolb, boolc, boold, boole]=determinebools(Realprevs, memory)
aantalrijen=size(Realprevs,1);
aantalkolommen=size(Realprevs,2);
if memory>aantalkolommen
    memory=aantalkolommen;
end
```

```
for i=1:aantalrijen
    sumsquared(i)=sum(Realprevs(i,aantalkolommen-memory+1:aantalkolommen).^2);
end
boola=(sumsquared(1:6)==0)';
boolb=(sumsquared(7:14)==0)';
boolc=(sumsquared(15:25)==0)';
boold=(sumsquared(26:33)==0)';
boole=(sumsquared(34:38)==0)';


function [realpreva, realprevb, realprevc, realprevd, realpreve]=determineprev(realprev)
realpreva=realprev(1:6);
realprevb=realprev(7:14);
realprevc=realprev(15:25);
realprevd=realprev(26:33);
realpreve=realprev(34:38);
```

Appendix C

Sample χ-output of the Matlab GUI code generator

```
// CONSTANTS BELOW ARE GENERATED BY THE MATLAB-PROGRAM REENTRANT.M
// THIS CHI 0.7 PROGRAM WAS ENTIRELY GENERATED BY MATLAB ON 26-Mar-2002 AT 10:34
// DO NOT EDIT WITHOUT KNOWLEDGE OF THE CHI-FORMALISM.

const simulationtime          :real = 25.0
,     timestep                :real = 0.010000
,     numberofstates          :nat  = 43
,     numberofproducts        :nat  = 5
,     numberofmachines        :nat  = 5
,     numberofproductionsteps :nat  = 38
,     cumulagesperproduct     :nat^ 5 = <0,6,12,22,32>
,     numberpermachine        :nat^ 5 = <6,8,11,8,5>
,     cumulnumberpermachine   :nat^ 5 = <0,6,14,25,33>
,     cumulnumberpermachine2  :nat^ 5 = <6,14,25,33,38>
,     prodlist                :nat^ 43 = <0,6,14,25,33,38,34,26,15,7,1,39,2,8,16,9,
                                          17,27,18,28,35,40,3,29,19,30,20,10,21,11,
                                          36,41,22,31,23,32,4,12,5,13,24,37,42>
,     previdx                 :nat^ 38 = <38,7,40,41,32,12,0,15,2,16,20,21,4,5,6,26,
                                          8,9,27,29,30,10,42,31,13,14,34,17,18,3,19,
                                          22,23,25,39,28,11,24>
,     firststep               :nat^ 5 = <0,34,2,3,22>
,     rowmachineorder         :nat^ 50 = <1,2,3,4,5,0,0,0,0,0,5,4,3,2,1,0,0,0,0,0,
                                          1,2,3,2,3,4,3,4,5,0,1,4,3,4,3,2,3,2,5,0,
                                          3,4,3,4,1,2,1,2,3,5>
,     inputrates              :real^ 5 = <5.000,5.000,5.000,5.000,5.000>

// DO NOT EDIT BELOW THIS LINE. THE CONFIGURATION OF THE RE-ENTRANT FLOW-SHOP
// IS ENTIRELY CONTROLLED BY THE PARAMETERS ABOVE.

type loggen = real#nat#nat
,    id = nat
,    timestamp = real
,    prodtype = nat
```

```
,    age = nat
,    lot= (id#timestamp#prodtype#age)
,    rate= real
,    extlot=lot#real

proc G (reqlengthgen: ?(nat^numberofstates), inputgen: !lot) =
|[ ident: nat^numberofproducts, bc: nat^numberofstates, i: nat, nextinput: real^numberofproducts
 | i:=0
 ; *[ i < numberofproducts -> ident.i:=1
                           ; [ inputrates.i > 0.0 -> nextinput.i:=1/inputrates.i
                             | inputrates.i = 0.0 -> nextinput.i:=2*simulationtime
                             ]
                           ; i:=i+1
    ]
 ; reqlengthgen?bc  // ontvang nulwaarden voor de bc-s als beginconditie van proces buffercontents
 ; *[                                reqlengthgen?bc        -> skip
    | j: nat <- 0..numberofproducts: delta nextinput.j-time -> inputgen!<ident.j, time, j, 0>
                                                               ; ident.j:=ident.j+1
                                                               ; nextinput.j:=time+1/inputrates.j
                                                               ; reqlengthgen?bc
    ]
]|

proc B (machineupdate: !(bool^numberofmachines), getinfo: ?(nat#int),
        mrates: !(rate^numberofproductionsteps), inputgen: ?lot,
        inputmac: (?lot)^numberofmachines, outputmac: (!extlot)^numberofmachines,
        updatelength: !(nat^2)) =
|[ x: lot, xs: (lot*)^numberofstates, t,m,list: nat, machinebusy: bool^numberofmachines,
   processtime: real, product: int, productnat: nat,
   maxrates: rate^numberofproductionsteps
 | t:=0
 ; *[ t < numberofmachines -> machinebusy.t:=false; t:=t+1 ]
 ; t:=0
 ; *[ t < numberofstates -> xs.t:=[]; t:=t+1 ]
 ; maxrates:= <5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000,
               5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000,
               5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000,
               5.000,5.000,5.000,5.000,5.000,5.000,5.000,5.000>
 ; mrates!maxrates
 ; machineupdate!machinebusy  // start sim: send status to the controller
 ; *[                                inputgen?x   -> list:=prodlist.((cumulagesperproduct.(x.2)) + x.3)
                                                    ; xs.list := xs.list++[x]
                                                    ; updatelength!<list, len(xs.list)>
    | i: nat <- 0..numberofmachines: inputmac.i?x -> list:=prodlist.((cumulagesperproduct.(x.2)) + x.3)
                                                    ; machinebusy.i:=false
                                                    ; machineupdate!machinebusy
                                                    ; xs.list := xs.list++[x]
                                                    ; updatelength!<list,len(xs.list)>
    |                         getinfo?<m, product> -> productnat:=i2n(product)
                                                    ; processtime:=1/(maxrates.productnat)
                                                    ; outputmac.m!<hd(xs.productnat), processtime>
                                                    ; machinebusy.m:=true
                                                    ; machineupdate!machinebusy
                                                    ; xs.productnat:=tl(xs.productnat)
```

```
                                                    ; updatelength!<productnat, len(xs.productnat)>
    ]
]|

proc buffercontents(updatelength: ?(nat^2), requestlengths
 , reqlengthgen: !(nat^numberofstates), output: !file)=
|[ t: nat, bc: nat^numberofstates, info: nat^2
 // info has 2 elements: 0: buffernumber and 1: bufferlength
 | t:=0
 ; output!numberofproducts," ",numberofmachines," ",rowmachineorder,nl()
 ; *[ t < numberofstates -> bc.t:=0; t:=t+1 ]
 ; output!time," "; t:=0; *[ t< numberofstates -> output!bc.t," "; t:=t+1]; output!nl()
 ; reqlengthgen!bc
 ; *[ true -> updatelength?info
            ; bc.(info.0):=info.1
            ; output!time," "
            ; t:=0; *[t < numberofstates -> output!bc.t," "; t:=t+1]
            ; output!nl()
            ; reqlengthgen!bc
            ; requestlengths!bc
    ]
]|

proc C(machineupdate: ?(bool^numberofmachines), mrates: ?(rate^numberofproductionsteps)
 , cdesrates: ?rate^numberofstates, getinfo: !(nat#int)
 , requestlengths: ?(nat^numberofstates))=
|[ maxrates: rate^numberofproductionsteps, desrates: rate^numberofstates, t:nat
 , bc, released: nat^numberofstates, cumul, diffs: real^numberofstates
 , scaledrates: rate^numberofstates, machinebusy: bool^numberofmachines
 , bestchoice: int^numberofmachines
 | t:=0
 ; *[ t < numberofmachines -> bestchoice.t:=-1; t:=t+1 ]
 ; t:=0
 ; *[ t < numberofstates -> cumul.t:=0.0; released.t:=0; scaledrates.t:=0.0; t:=t+1 ]
 ; mrates?maxrates             // start sim: import maxrates from proces B
 ; machineupdate?machinebusy   // start sim: import machine status from proces B
 ; *[ machineupdate?machinebusy   -> diffs:=computereldiffs(cumul, released)
                                  ; bestchoice:=computebestchoice(diffs, bc)
    | j: nat <- 0..numberofmachines: machinebusy.j = false and bestchoice.j /= -1
      ; getinfo!<j, bestchoice.j> -> released.(i2n(bestchoice.j)):=released.(i2n(bestchoice.j))+1
                                  ; diffs:=computereldiffs(cumul, released)
                                  ; bestchoice:=computebestchoice(diffs, bc)
    | requestlengths?bc  -> bestchoice:=computebestchoice(diffs, bc)
    | cdesrates?desrates -> scaledrates:=computescaled(desrates, maxrates)
                          ; t:=0
                          ; *[ t<numberofstates -> cumul.t:= cumul.t + scaledrates.t * timestep
                                                 ; t:=t+1
                             ]
                          ; diffs:=computereldiffs(cumul, released)
                          ; bestchoice:=computebestchoice(diffs, bc)
    ]
]|
```

```
func computescaled(desrates: rate^numberofstates
 , maxrates: rate^numberofproductionsteps) -> rate^numberofstates =
|[ t, i: nat, egdmax: rate^numberofproductionsteps, newscaledrates: rate^numberofstates
 , inproduct: real^numberofmachines, inpr: real
 | t:=0
 ; *[ t < numberofmachines -> inproduct.t:=0.0; t:=t+1 ]
 ; t:=0
 ; *[ t<numberofproductionsteps -> egdmax.t:=1/maxrates.t; t:=t+1 ]
 ; t:=0
 ; *[ t<numberofmachines ->
           i:=0
           ; *[i < numberpermachine.t ->
                      inproduct.t := inproduct.t + desrates.(i+cumulnumberpermachine.t) * ...
                                    ... egdmax.(i+cumulnumberpermachine.t)
                      ; i:=i+1
              ]
           ; t:=t+1
   ]
 ; t:=0
 ; *[ t<numberofproductionsteps -> i:=numberofmachines-1
                                  ; *[ cumulnumberpermachine.i > t -> i:=i-1 ]
                                  ; inpr:=inproduct.i
                                  ; newscaledrates.t := desrates.t / max(1.0,inpr)
                                  ; t:=t+1
   ]
 ; t:=numberofproductionsteps
 ; *[ t<numberofstates -> newscaledrates.t:=desrates.t
                        ; t:=t+1
   ]
 ; ret newscaledrates
]|

func computereldiffs(cumul: real^numberofstates, released: nat^numberofstates) -> real^numberofstates =
|[ t: nat, diffs: real^numberofstates
 | t:=0
 ; *[ t < numberofstates -> [ cumul.t > 0 -> diffs.t:=(cumul.t-released.t)/cumul.t
                                           ; [ diffs.t < 0.00000000001 -> diffs.t:=0.0
                                             | diffs.t >=0.00000000001 -> skip
                                             ]
                            | cumul.t = 0 -> diffs.t:=0.0
                            ]
                          ; t:=t+1
   ]
 ; ret diffs
]|

func computebestchoice(diffs: real^numberofstates, bc: nat^numberofstates)
                       -> int^numberofmachines =
|[ t, i: nat, bestchoice: int^numberofmachines, biggestdiff: real^numberofmachines
 | t:=0
 ; *[ t < numberofmachines -> bestchoice.t:=-1; biggestdiff.t:=0.0; t:=t+1 ]
 ; i:=0; t:=0
 ; *[ i < numberofmachines ->
      *[ t < cumulnumberpermachine2.i ->
```

```
              [ diffs.t > 0 and bc.t > 0 and diffs.t > biggestdiff.i -> biggestdiff.i := diffs.t
                                                                     ; bestchoice.i := n2i(t)
              | diffs.t <=0 or  bc.t <=0 or  diffs.t <=biggestdiff.i -> skip
              ]
            ; t:=t+1
        ]
        ; i:=i+1
    ]
 ; ret bestchoice
]|

proc M (in1: ?extlot, out1: !lot) =
|[ lotwithtime: extlot, prod: lot
 | *[ true -> in1?lotwithtime; prod:=lotwithtime.0; delta lotwithtime.1
            ; prod.3:=prod.3+1; out1!prod
    ]
]|

proc iodata(cdesrates: !rate^numberofstates)=
|[ MLtimeOK: bool, newdesiredprodrates: rate^numberofstates
 | *[ true -> MLtimeOK:=true
    ; [ not MLtimeOK -> skip
      | MLtimeOK     -> newdesiredprodrates:= <3.000,3.000,3.000,3.000,3.000,3.000,
                                               3.000,3.000,3.000,3.000,3.000,3.000,
                                               3.000,3.000,3.000,3.000,3.000,3.000,
                                               3.000,3.000,3.000,3.000,3.000,3.000,
                                               3.000,3.000,3.000,3.000,3.000,3.000,
                                               3.000,3.000,3.000,3.000,3.000,3.000,
                                               3.000,3.000,1.000,1.000,1.000,1.000,
                                               1.000>
                       ; cdesrates!newdesiredprodrates
      ]
    ; [ time > simulationtime -> terminate
      | time <=simulationtime -> skip
      ]
    ; delta timestep
    ]
]|

syst REENTRANT()=
|[ inputgen: -lot, mrates: -rate^numberofproductionsteps, getinfo: -(nat#int),
   cdesrates: -rate^numberofstates, inputmac: (-lot)^numberofmachines,
   outputmac: (-extlot)^numberofmachines, updatelength: -nat^2, requestlengths,
   reqlengthgen: -nat^numberofstates, machineupdate: -bool^numberofmachines
 | G( reqlengthgen, inputgen )
|| i: nat <- 0..numberofmachines: M( outputmac.i, inputmac.i )
|| iodata( cdesrates )
|| B( machineupdate, getinfo, mrates, inputgen, inputmac, outputmac, updatelength )
|| buffercontents( updatelength, requestlengths, reqlengthgen, fileout("02032601.crf"))
|| C( machineupdate, mrates, cdesrates, getinfo, requestlengths )
]|

xper=|[ REENTRANT() ]|
```

Appendix D

System identification source models and matrices

D.1 GBME flowline with one product type

```
const pi            : real = 3.1415926536
,     simulationtime: real = 10000.0

func chirp(c, a, f0, f1, t: real) -> real =
|[ behta: real
 | behta:=(f1 - f0) / simulationtime
 ; ret c + a * cos(2 * pi * (behta / 2 * (t^2) + f0 * t ))
]|

proc G (a: ~void) =
|[ u:real
 | *[ time < simulationtime -> u:=chirp(3.0, 1.0, 0.01, 0.001, time)
                             ; delta 1/u
                             ; a~
    ]
 ; delta 100; terminate
]|

proc B (a, b: ~void, c: !file, bc: nat) =
|[ x: nat
 | x:=bc
 ; *[ true;  a~ -> x:=x+1
                 ; c!time,tab(),x,nl()
    | x > 0; b~ -> x:=x-1
                 ; c!time,tab(),x,nl()
    ]
]|
```

```
proc M (a, b: ~void) =
|[ m: real
 | *[ true -> a~
            ; m:=chirp(3.0, 1.5, 0.005, 0.05, time)
            ; delta 1/m
            ; b~
    ]
]|

proc E (a: ~void, b: !file) =
|[ x: nat
 | x:=0
 ; *[ true -> a~; x:=x + 1
            ; b!time,tab(),x,nl()
    ]
]|

syst Model() =
|[ a, b, c: -void
 |  G( a )
 || B( a, b, fileout("buffer.txt"), 250 )
 || M( b, c )
 || E( c, fileout("stock.txt") )
]|

xper =
|[ Model() ]|
```

D.2 Matrices of state space model GBME flowline with one product type

The identified state space model of this discrete event system is:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

$$B = 1,0 \cdot 10^{-3} \times \begin{pmatrix} 0.0005 & 0.0006 \\ -0.2326 & 0.2325 \end{pmatrix},$$

$$C = 1,0 \cdot 10^{5} \times \begin{pmatrix} 0.0034 & -0.0431 \\ 8.8122 & 0.0190 \end{pmatrix},$$

$$D = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix},$$

$$K = 1,0 \cdot 10^{-4} \times \begin{pmatrix} 0.0010 & 0.0017 \\ -0.4766 & -0.2638 \end{pmatrix}.$$

D.3 GBMBMBME flowline with one product type

```
const pi            : real = 3.1415926536
,     simulationtime: real = 10000.0

func chirp(c, a, f0, f1, t: real) -> real =
|[ behta: real
 | behta:=(f1-f0)/simulationtime
 ; ret c+a*cos(2*pi * ( behta/2*(t^2) + f0*t ))
]|

proc G (a: ~void) =
|[ u:real
 | *[ time < simulationtime -> u:=chirp(3.0, 1.0, 0.01, 0.0001, time)
                             ; delta 1/u; a~
    ]
 ; delta 50; terminate
]|

proc B (a, b: ~void, c: !file, bc: nat) =
|[ x: nat
 | x:=bc
 ; *[ true;  a~ -> x:=x+1; c!time,tab(),x,nl()
    | x > 0; b~ -> x:=x-1; c!time,tab(),x,nl()
    ]
]|

proc M (a, b: ~void, con, amp, f0, f1: real) =
|[ m: real
 | *[ true -> a~
            ; m:=chirp(con, amp, f0, f1, time)
            ; delta 1/m
            ; b~
    ]
]|

proc E (a: ~void, b: !file) =
|[ x: nat
 | x:=0
 ; *[ true -> a~; x:=x+1; b!time,tab(),x,nl() ]
]|

syst Model() =
|[ a, b, c, d, e, f, g: -void
 |  G( a )
 || B( a, b, fileout("buffer1.txt"), 500 )  || M( b, c, 3.0, 1.5, 0.0001, 0.05 )
 || B( c, d, fileout("buffer2.txt"), 500 )  || M( d, e, 3.0, 0.5, 0.03, 0.0007 )
 || B( e, f, fileout("buffer3.txt"), 1100 ) || M( f, g, 3.0, 2.0, 0.00001, 0.01 )
 || E( g, fileout("stock.txt") )
]|

xper =
|[ Model() ]|
```

D.4 Matrices of state space model GBMBMBME flowline with one product type

The identified state space model of this discrete event system is:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$B = 1,0 \cdot 10^{-3} \times \begin{pmatrix} 0.0002 & 0.0003 & 0.0004 & 0.0002 \\ -0.0340 & -0.0399 & -0.0174 & 0.0912 \\ -0.1668 & 0.0390 & 0.1612 & -0.0332 \\ -0.0966 & 0.2732 & -0.1968 & 0.0210 \end{pmatrix},$$

$$C = 1,0 \cdot 10^{5} \times \begin{pmatrix} 0.0073 & -0.0071 & -0.0399 & -0.0321 \\ 0.0052 & -0.0152 & -0.0184 & 0.0370 \\ 0.0096 & -0.0947 & 0.0297 & -0.0181 \\ 8.6429 & 0.0848 & -0.0071 & 0.0025 \end{pmatrix},$$

$$D = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$K = 1,0 \cdot 10^{-4} \times \begin{pmatrix} 0.0003 & 0.0005 & 0.0007 & 0.0010 \\ -0.0480 & -0.0717 & -0.0863 & 0.0288 \\ -0.2172 & -0.2132 & -0.0938 & -0.1228 \\ -0.1378 & 0.0206 & -0.1430 & -0.1129 \end{pmatrix}.$$

D.5 GBMBME re-entrant flowline with one product type

```
const pi            : real = 3.1415926536
,     simulationtime: real = 10000.0
,     timestep      : real = 0.01

type lot=bool

func chirp(c, a, f0, f1, t: real) -> real =
|[ behta: real
 | behta:=(f1-f0)/simulationtime
 ; ret c+a*cos(2*pi * ( behta/2*(t^2) + f0*t ))
]|

proc G (a: !lot) =
|[ u:real
 | *[ time < simulationtime -> u:=chirp(1.5, 1.0, 0.01, 0.0001, time)
                             ; delta 1/u
                             ; a!false
    ]
 ; delta 50; terminate
]|

proc B (a: (?lot)^2, b: !lot, c, d: !file, toc: !(nat^2)^2, fromc: ?nat, bc: nat^2) =
|[ n: nat^2, x: lot, best: nat, released: nat^2, next, go: bool
 | n:=bc
 ; go:=false
 ; released:=<0,0>
 ; *[ j: nat <- 0..2: true; a.j?x -> n.j:= n.j + 1
                                   ; [ j=0 -> c!time,tab(),n.0,nl()
                                     | j=1 -> d!time,tab(),n.1,nl()
                                     ]
    |       rel(n) and go; b!next -> released.best:=released.best+1
                                   ; n.best:= n.best - 1
                                   ; [ best=0 -> c!time,tab(),n.0,nl()
                                     | best=1 -> d!time,tab(),n.1,nl()
                                     ]
    |     true; toc!<n, released> -> fromc?best
                                   ; [ best = 0 -> next:=false; go:=false
                                     | best = 1 -> next:=false; best:=0; go:=true
                                     | best = 2 -> next:=true; best:=1; go:=true
                                     ]
    ]
]|

func rel(n: nat^2) -> bool =
|[ ret (n.0 > 0 or n.1 > 0) ]|

proc E (a: ?lot, b: !file) =
|[ x: lot, n: nat
 | n:=0
 ; *[ true -> a?x; n:=n+1; b!time,tab(),n,nl() ]
]|
```

```
proc C (fromb: ?(nat^2)^2, tob: !nat)=
|[ c: real^2, m: real^2, datab: (nat^2)^2, bc, released: nat^2
 , relative: real^2, best: nat
 | c:=<0.0,0.0>
 ; *[ true -> m.0:=chirp(2.5, 1.0, 0.01, 0.0001, time)
            ; m.1:=chirp(2.5, 1.5, 0.0005, 0.05, time)
            ; c.0:=c.0+m.0*timestep
            ; c.1:=c.1+m.1*timestep
            ; fromb?datab
            ; bc:=datab.0
            ; released:=datab.1
            ; relative.0:=(c.0-released.0)/c.0
            ; relative.1:=(c.1-released.1)/c.1
            ; best:=decision([relative.0, relative.1], [bc.0, bc.1])
            ; tob!best
            ; delta timestep
    ]
]|

func decision(diffs: real*, conts: nat*) -> nat =
|[ i, number, index: nat, record: real, c: bool
 | i:=1; number:=len(diffs); record:=0.0; index:=0
 ; *[ i <= number -> c:= hd(diffs) > 0 and hd(conts) > 0 and hd(diffs) > record
                  ; [ c     -> index:=i
                                ; record:=hd(diffs)
                    | not c -> skip
                    ]
                  ; diffs:=tl(diffs)
                  ; conts:=tl(conts)
                  ; i:=i+1
    ]
 ; ret index
]|

proc M (a: ?lot, b, c: !lot) =
|[ x: lot, m1, m2: real
 | *[ true -> a?x; [ not x -> m1:=chirp(3.0, 0.25, 0.00005, 0.02, time)
                            ; delta 1/m1
                            ; x:=true
                            ; b!x
                    | x     -> m2:=chirp(3.0, 0.5, 0.0003, 0.07, time)
                            ; delta 1/m2
                            ; c!x
                    ]
    ]
]|
```

```
syst Model() =
|[ a: (-lot)^2, b, c: -lot, bc: -(nat^2)^2, cb: -nat
 |  G( a.0 )
 || B( a, b, fileout("buffer1.txt"), fileout("buffer2.txt"), bc, cb, <700, 500> )
 || M( b, a.1, c )
 || C( bc, cb )
 || E( c, fileout("stock.txt") )
]|

xper =
|[ Model() ]|
```

D.6 Matrices of state space model GBMBME re-entrant flowline with one product type

The identified state space model of this discrete event system is:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$B = 1,0 \cdot 10^{-3} \times \begin{pmatrix} 0.0025 & 0.0003 & 0.0006 & 0.0001 & 0.0000 \\ 0.0354 & 0.1670 & -0.1857 & -0.0096 & -0.0021 \\ -1.9377 & 1.9038 & 0.2230 & 0.1423 & 0.1174 \end{pmatrix},$$

$$C = 1,0 \cdot 10^{5} \times \begin{pmatrix} 0.9842 & -0.0123 & -0.0065 \\ 0.0061 & 0.0504 & 0.0011 \\ 1.9677 & -0.0307 & 0.0043 \end{pmatrix},$$

$$D = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$K = 1,0 \cdot 10^{-3} \times \begin{pmatrix} 0.0003 & 0.0003 & 0.0008 \\ -0.0117 & 0.1248 & -0.0638 \\ -0.6415 & 0.2430 & 0.6465 \end{pmatrix}.$$

D.7 GBMBME re-entrant flowline with two product types

```
const pi            : real = 3.1415926536
,     simulationtime: real = 10000.0
,     timestep      : real = 0.01

type lot=nat#bool
// lot.0 is product type: 0 or 1
// lot.1 is production stage: first stage: false; second stage: true

func chirp(c, a, f0, f1, t: real) -> real =
|[ behta: real
 | behta:=(f1-f0)/simulationtime
 ; ret c+a*cos(2*pi * ( behta/2*(t^2) + f0*t ))
]|

proc G (a: !lot) =
|[ u, next: real^2
 | u:=<1.5, 1.5>
 ; next:=<2/3,2/3>
 ; *[ j: nat <- 0..2: time < simulationtime; delta next.j-time
                -> a!<j, false>
                 ; [ j=0 -> u.j:=chirp(1.5, 1.0, 0.01, 0.0001, time)
                   | j=1 -> u.j:=chirp(1.5, 1.0, 0.0001, 0.01, time)
                   ]
                 ; next.j:=time+1/u.j
   ]
 ; delta 50; terminate
]|

func rel(n: (nat^2)^2) -> bool =
|[ ret (n.0.0 > 0 or n.0.1 > 0 or n.1.0 > 0 or n.1.1 >0) ]|

func decision(diffs: real*, conts: nat*) -> nat =
|[ i, number, index: nat, record: real, c: bool
 | i:=1; number:=len(diffs); record:=0.0; index:=0
 ; *[ i <= number -> c:= hd(diffs) > 0 and hd(conts) > 0 and hd(diffs) > record
                  ; [ c     -> index:=i
                                ; record:=hd(diffs)
                    | not c -> skip
                    ]
                  ; diffs:=tl(diffs)
                  ; conts:=tl(conts)
                  ; i:=i+1
   ]
 ; ret index
]|
```

```
// n.0.i number of products in first stage for product 0 and 1 (i=0 and i=1)
// n.1.i number of products in second stage for product 0 and 1 (i=0 and i=1)
proc B (a: (?lot)^2, b: !lot, c,d,e,f: !file, toc: !((nat^2)^2)^2, fromc: ?nat, bc: (nat^2)^2) =
|[ x: lot, n: (nat^2)^2, go: bool, released: (nat^2)^2, next: lot, best: nat, bestg: nat^2
 | n:=bc
 ; go:=false
 ; released:=<<0,0>,<0,0>>
 ; *[ j: nat <- 0..2: true; a.j?x -> [ x.0 = 0 -> n.j.0 := n.j.0 + 1
                                          ; [ j = 0 -> c!time,tab(),n.0.0,nl()
                                            | j = 1 -> d!time,tab(),n.0.1,nl()
                                            ]
                                    | x.0 = 1 -> n.j.1 := n.j.1 + 1
                                          ; [ j = 0 -> e!time,tab(),n.1.0,nl()
                                            | j = 1 -> f!time,tab(),n.1.1,nl()
                                            ]
                                    ]
    | rel(n) and go;       b!next -> released.(bestg.0).(bestg.1):=released.(bestg.0).(bestg.1)+1
                                   ; n.(bestg.0).(bestg.1):=n.(bestg.0).(bestg.1)-1
                                   ; [ bestg = <0,0> -> c!time,tab(),n.0.0,nl()
                                     | bestg = <0,1> -> d!time,tab(),n.0.1,nl()
                                     | bestg = <1,0> -> e!time,tab(),n.1.0,nl()
                                     | bestg = <1,1> -> f!time,tab(),n.1.1,nl()
                                     ]
    | true;     toc!<n, released> -> fromc?best
                                   ; [ best = 0 -> next:=<0,false>; go:=false
                                     | best = 1 -> next:=<0,false>; bestg:=<0,0>; go:=true
                                     | best = 2 -> next:=<0,true> ; bestg:=<0,1>; go:=true
                                     | best = 3 -> next:=<1,false> ; bestg:=<1,0>; go:=true
                                     | best = 4 -> next:=<1,true> ; bestg:=<1,1>; go:=true
                                     ]
    ]
]|

proc M (a: ?lot, b, c: !lot) =
|[ x: lot, m00, m01, m10, m11: real
 | *[ true -> a?x; [ not x.1 -> [ x.0 = 0 -> m00:=chirp(6.0, 0.5, 0.01, 0.0001, time)
                                           ; delta 1/m00
                                           ; x.1:=true
                                           ; b!x
                                 | x.0 = 1 -> m01:=chirp(6.0, 0.25, 0.0001, 0.01, time)
                                           ; delta 1/m01
                                           ; x.1:=true
                                           ; b!x
                                 ]
                   | x.1     -> [ x.0 = 0 -> m10:=chirp(6.0, 0.75, 0.04, 0.0004, time)
                                           ; delta 1/m10
                                           ; c!x
                                 | x.0 = 1 -> m11:=chirp(6.0, 1.0, 0.0003, 0.03, time)
                                           ; delta 1/m11
                                           ; c!x
                                 ]
                   ]
    ]
]|
```

```
proc C (fromb: ?((nat^2)^2)^2, tob: !nat)=
|[ c: (real^2)^2, m: (real^2)^2, datab: ((nat^2)^2)^2, bc
 , released: (nat^2)^2, relative: (real^2)^2, best: nat
 | c:=<<0.0,0.0>,<0.0,0.0>>
 ; *[ true -> m.0.0:=chirp(5.0, 1.0, 0.01, 0.0001, time)
            ; m.0.1:=chirp(5.0, 1.5, 0.0005, 0.05, time)
            ; m.1.0:=chirp(5.0, 0.5, 0.05, 0.0005, time)
            ; m.1.1:=chirp(5.0, 1.0, 0.0001, 0.01, time)
            ; c.0.0:=c.0.0+m.0.0*timestep
            ; c.0.1:=c.0.1+m.0.1*timestep
            ; c.1.0:=c.1.0+m.1.0*timestep
            ; c.1.1:=c.1.1+m.1.1*timestep
            ; fromb?datab
            ; bc:=datab.0
            ; released:=datab.1
            ; relative.0.0:=(c.0.0-released.0.0)/c.0.0
            ; relative.0.1:=(c.0.1-released.0.1)/c.0.1
            ; relative.1.0:=(c.1.0-released.1.0)/c.1.0
            ; relative.1.1:=(c.1.1-released.1.1)/c.1.1
            ; best:=decision([relative.0.0, relative.0.1, relative.1.0, relative.1.1],
                            [bc.0.0, bc.0.1, bc.1.0, bc.1.1])
            ; tob!best
            ; delta timestep
    ]
]|

proc E (a: ?lot, b,c: !file) =
|[ x: lot, n: nat^2
 | n:=<0, 0>
 ; *[ true -> a?x; n.(x.0):=n.(x.0)+1; [ x.0 = 0 -> b!time,tab(),n.0,nl()
                                       | x.0 = 1 -> c!time,tab(),n.1,nl()
                                       ]
    ]
]|

syst Model() =
|[ a: (-lot)^2, b, c: -lot, bc: -((nat^2)^2)^2, cb: -nat
 |  G( a.0 )
 || B( a, b, fileout("buffer11.txt"), fileout("buffer12.txt")
     , fileout("buffer21.txt"), fileout("buffer22.txt"), bc, cb, <<800, 600>, <400, 200>> )
 || M( b, a.1, c )
 || C( bc, cb )
 || E( c, fileout("stock1.txt"), fileout("stock2.txt") )
]|

xper =
|[ Model() ]|
```

D.8 Matrices of state space model GBMBME re-entrant flowline with two product types

The identified state space model of this discrete event system is:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$B = 1,0 \cdot 10^{-3} \times \begin{pmatrix} -0.059 & 0.023 & 0.042 & 0.000 & 0.000 & -0.009 & 0.035 & -0.053 & 0.000 & 0.000 \\ 0.037 & 0.136 & 0.015 & -0.071 & -0.017 & 0.114 & -0.014 & -0.001 & -0.009 & -0.022 \\ -0.149 & -0.011 & -0.093 & -0.036 & 0.014 & 0.043 & 0.008 & 0.087 & 0.012 & 0.012 \\ 0.153 & 0.185 & -0.049 & 0.602 & -0.231 & -0.017 & -0.227 & -0.279 & -0.000 & 0.008 \\ 0.011 & 0.017 & 0.003 & 0.128 & 1.438 & 0.009 & -0.006 & -0.008 & -0.093 & -0.062 \\ 2.004 & 1.773 & -0.611 & -0.173 & 0.059 & -0.842 & -2.920 & -3.759 & -0.059 & -0.080 \end{pmatrix},$$

$$C = 1,0 \cdot 10^{5} \times \begin{pmatrix} 0.0135 & 0.0083 & -0.0295 & 0.0001 & 0.0007 & 0.0002 \\ 0.0086 & 0.0398 & 0.0062 & 0.0007 & 0.0005 & -0.0000 \\ 0.0012 & 0.0026 & -0.0119 & -0.0002 & -0.0020 & -0.0002 \\ 0.0005 & -0.0105 & -0.0021 & -0.0013 & 0.0020 & -0.0001 \\ 2.7145 & -0.0129 & 0.0050 & 0.0021 & -0.0002 & -0.0000 \\ 2.7142 & -0.0012 & 0.0075 & -0.0017 & -0.0007 & 0.0001 \end{pmatrix},$$

$$D = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$K = 1,0 \cdot 10^{-3} \times \begin{pmatrix} 0.0000 & 0.0000 & 0.0001 & 0.0001 & 0.0002 & 0.0002 \\ 0.0024 & 0.0104 & -0.0007 & -0.0034 & -0.0086 & -0.0066 \\ -0.0120 & -0.0005 & -0.0130 & -0.0028 & 0.0057 & 0.0071 \\ 0.0075 & 0.0052 & -0.0134 & -0.0333 & 0.0818 & -0.0534 \\ 0.0721 & 0.0479 & -0.1962 & 0.0449 & -0.0357 & -0.0761 \\ 0.4419 & -0.4778 & -1.6435 & -0.9335 & -0.7198 & -0.0886 \end{pmatrix}.$$

Appendix E

State space realization

E.1 GBME flowline

E.1.1 χ-model

The following source code has been used to simulate the GBME flowline with only one product type. The black colored model is based on integration strategy 1 (see Section 6.3). The black colored model with the red additions is the model based on integration strategy 2.

```
const simulationtime: real = 100.0
,     timestep      : real = 0.001


// g(time)=1.5+sin(0.5 * time)
// Integrated: G(time)=1.5*time-2*cos(0.5*time)+2    G(0)=0
proc G (a: ~void) =
|[ Gi:real, released: nat
 | released:=0
 ; *[ time < simulationtime -> Gi:=1.5*time-2*cos(0.5*time)+2
                             ; [ Gi >= released+1 -> a~; released:= released + 1
                               | Gi <  released+1 -> delta timestep
                               ]
    ]
 ; delta 10; terminate
]|

proc B (a, b: ~void, c: !file, bc: nat) =
|[ x: nat
 | x:=bc
 ; *[ true;  a~ -> x:=x+1; c!time,tab(),x,nl()
    | x > 0; b~ -> x:=x-1; c!time,tab(),x,nl()
    ]
]|
```

```
// maxrate=1.5, process time=1/1.5
// m(time)=1+0.5*sin(0.2*time)
// Integrated: M(time)=time-2.5*cos(0.2*time)+2.5   M(0)=0
proc M (a, b: ~void) =
|[ Mi: real, processed: nat
 | processed:=0
 ; *[ true -> Mi:=(time+1/1.5)-2.5*cos(0.2*(time+1/1.5))+2.5
            ; [ Mi >= processed + 1 -> a~; delta 1/1.5; processed:=processed + 1; b~
              | Mi <  processed + 1 -> delta timestep
              ]
    ]
]|

proc E (a: ~void, b: !file) =
|[ x: nat
 | x:=0
 ; *[ true -> a~; x:=x+1; b!time,tab(),x,nl() ]
]|

syst Model() =
|[ a, b, c: -void
  | G( a )
 || B( a, b, fileout("buffermplus.txt"), 0 )
 || M( b, c )
 || E( c, fileout("stockmplus.txt") )
]|

xper =
|[ Model() ]|
```

E.1.2 Simulink model

The following Simulink model has been used to simulate the state space models obtained by realization techniques and model reduction techniques. The third (most below) part of this model is a model using pure time delay functions of Simulink. In fact, this model simulates the behavior of the differential equations 6.8 without Padé approximations.

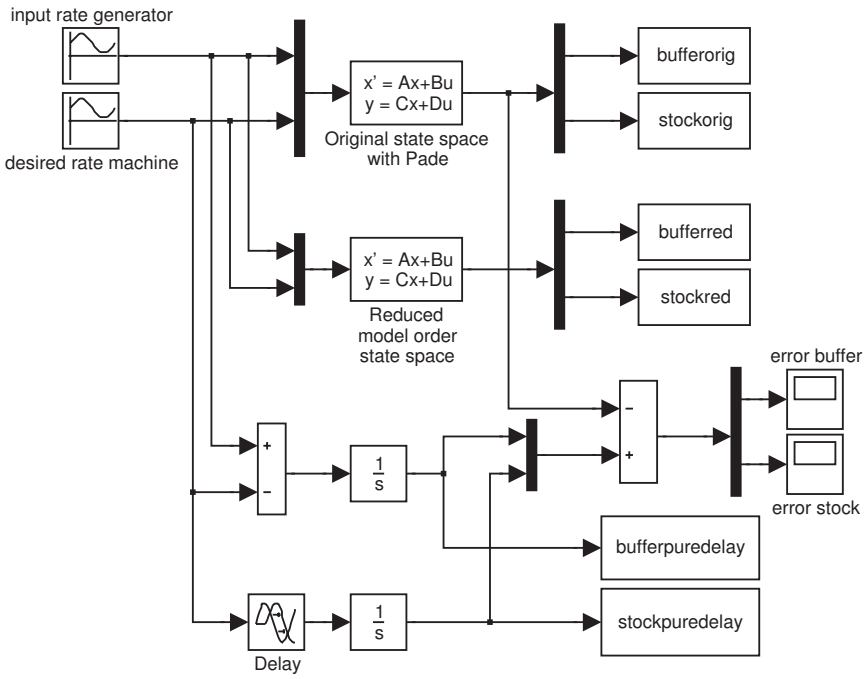


Figure E.1: Simulink model of the GBME flowline.

E.2 GBMBMBME flowline

E.2.1 χ-model

The following source code has been used to simulate the GBMBMBME flowline with only one product type. The black colored model is based on integration strategy 1 (see Section 6.3). The black colored model with the red additions is the model based on integration strategy 2.

```
const simulationtime: real = 100.0
,     timestep      : real = 0.001

//g(time) = 1.5 + sin (0.5 * time)
//Integrated: G(time) =  1.5*time - 2 * cos (0.5 * time) + 2    G(0)=0
proc G (a: ~void, bias, ampl, freq, phaselag: real) =
|[ Gi:real, released: nat
 | released:=0
 ; *[ time < simulationtime
     -> Gi:=bias*time-ampl/freq*cos(freq*time-phaselag)+ampl/freq*cos(-phaselag)
      ; [ Gi >= released+1 -> a~; released:=released+1
        | Gi <  released+1 -> delta timestep
        ]
    ]
 ; delta 10; terminate
]|

proc B (a, b: ~void, c: !file, bc: nat) =
|[ x: nat
 | x:=bc
 ; *[ true;  a~ -> x:=x+1; c!time,tab(),x,nl()
    | x > 0; b~ -> x:=x-1; c!time,tab(),x,nl()
    ]
]|

// m_des(time) = mbias+mampl*sin(mfreq*time+phaselag)
// Integrated: M_des(time)= mbias*time-mampl/mfreq*cos(mfreq*time+phaselag)+mampl/mfreq*cos(phaselag)
// Initial condition: M(0)=0
proc M (a, b: ~void, mmax, mbias, mampl, mfreq, phaselag: real, id: nat) =
|[ Mi: real, processed: nat
 | processed:=0
 ; *[ true
     -> Mi:=mbias*(time+1/mmax)-mampl/mfreq*cos(mfreq*(time+1/mmax)+phaselag)+(mampl/mfreq*cos(phaselag))
      ; [ Mi >= processed + 1 -> a~; delta 1/mmax; processed:=processed + 1; b~
        | Mi <  processed + 1 -> delta timestep
        ]
    ]
]|
```

```
proc E (a: ~void, b: !file) =
|[ x: nat
 | x:=0
 ; *[ true -> a~; x:=x+1; b!time,tab(),x,nl() ]
]|

syst Model() =
|[ a, b, c, d, e, f, g: -void
 |  G( a, 1.5, 1.0, 0.5, 0.0 )
 || B( a, b, fileout("buffer1plus.txt"), 0 ) || M( b, c, 1.5, 1.0, 0.5, 0.2, 0.0, 2 )
 || B( c, d, fileout("buffer2plus.txt"), 0 ) || M( d, e, 1.0, 0.7, 0.3, 0.2, 1.0, 3 )
 || B( e, f, fileout("buffer3plus.txt"), 0 ) || M( f, g, 0.5, 0.3, 0.2, 0.2, 2.0, 4 )
 || E( g, fileout("buffer4plus.txt") )
]|

xper =
|[ Model() ]|
```

E.3 Matrices of the 7 outputs state space model

$$
A = \begin{pmatrix}
-3.34 & -2.48 & -0.19 & 0.34 & -1.11 & 0.42 & 0.35 & 0 & -0.07 & -0.16 \\
1.86 & -2.75 & 1.04 & -1.04 & -0.20 & -0.88 & 1.22 & 0 & -0.25 & -0.55 \\
-2.01 & 0.76 & -0.67 & 1.72 & 0.30 & 0.68 & -0.28 & 0 & 0.06 & 0.13 \\
0.30 & -1.38 & 0.384 & -3.30 & -2.30 & -0.38 & -0.65 & 0 & 0.13 & 0.30 \\
-1.10 & -0.36 & -0.92 & 1.91 & -2.64 & 0.94 & -1.52 & 0 & 0.31 & 0.69 \\
1.65 & 0.49 & 0.53 & -1.94 & 0.54 & -0.62 & 0.21 & 0 & -0.04 & -0.10 \\
-1.98 & 1.16 & -1.11 & 1.98 & -1.17 & 1.06 & -1.35 & 0 & 0.28 & 0.62 \\
0 & 0 & 0 & 0 & 0 & 0 & -0.36 & -1.50 & -2.12 & 0.17 \\
0.41 & -0.24 & 0.23 & -0.41 & 0.24 & -0.22 & 0.03 & 0.34 & -1.51 & -0.01 \\
0.90 & -0.53 & 0.51 & -0.90 & 0.53 & -0.48 & 0.73 & -0.16 & 0.53 & -0.33
\end{pmatrix}
$$

$$
B = \begin{pmatrix}
0.03 & 1.10 & 0.06 & 0 \\
0.29 & -0.77 & 1.09 & 0 \\
-0.39 & -1.59 & 1.78 & 0 \\
-0.03 & -1.10 & -0.06 & 0 \\
-0.42 & 0.80 & 0.26 & 0 \\
0 & 1.68 & 2.05 & 0 \\
0.68 & -0.14 & 0.55 & 1 \\
0 & 0 & 0 & 0 \\
-0.14 & 0.03 & -0.11 & 0.83 \\
-0.31 & 0.07 & -0.25 & 1.82
\end{pmatrix}
$$

$$
C = \begin{pmatrix}
-0.09 & 0.38 & -0.21 & 0.09 & -0.52 & -0.18 & 0.70 & 0 & -0.14 & -0.32 \\
-0.02 & -2.37 & 0.58 & 0.02 & 2.31 & -0.77 & 2.25 & 0 & -0.47 & -1.06 \\
2.46 & 0.12 & 0.20 & 2.44 & 0.03 & 0.23 & -0.14 & 0 & -0.18 & -0.39 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.20 & 3.29 & 0.17 & 0.36 \\
0.03 & 0.29 & -0.39 & -0.03 & -0.43 & 0 & 0.68 & 0 & -0.14 & -0.31 \\
-0.01 & -2.25 & 0.78 & 0.01 & 2.33 & -0.54 & 2.31 & 0 & -0.48 & -1.05 \\
2.46 & 0.12 & 0.20 & 2.44 & 0.03 & 0.23 & 0.06 & 0 & -0.01 & -0.03
\end{pmatrix}
$$

$$
D = \begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}.
$$

(E.1)