On the Operability of Heat Exchanger Networks

by

Caleb Hattingh, B.Sc. (Chem Eng)

Submitted to the

University of Cape Town in partial fulfillment

of the requirements for the degree of

Master of Science in Engineering

Department of Chemical Engineering

University of Cape Town

Rondebosch

South Africa

October 17, 2000

Acknowledgements

I would like to thank Sasol for funding this research project, the Process Control department at UCT for providing a vibrant working atmosphere, and the staff and postgraduate students at the Chemical Engineering department at UCT for many fond memories. Many thanks go to Donald Knuth for producing the TEX typesetting package and to Leslie Lamport for the LATEX enhancements; they made technical writing a joy.

I would further like to thank Professor Chris Swartz for his evergreen personality, continuous encouragement, meticulous attention to detail (something to which I hope to aspire) and general all-round support.

To my parents, James and Yvonne, I give great thanks for their continuous support throughout this project and all the years leading up to it. To Fred, Jean-Pierre and Clifton: thanks for the music.

Lastly, I would like to thank Georgina, who has been my inspiration and my light.

Science is a history
of superseded theories
— Anon.

Synopsis

The dynamic operability of processes refers to the degree to which plants may be satisfactorily controlled. This report presents a study of the operability of heat exchanger networks (HENs).

The integration of heat exchange systems such as HENs typically results in significant steady-state cost savings which is the motivation for their implementation. However, such integration may lead to problems in the dynamic operation of the system if the operability of HENs is not considered.

Operability analysis techniques are presented that provide a quantitative measure of the operability of HENs that is related to the minimum integral setpoint error of a closed-loop HEN under a step disturbance. The different operability analysis techniques are specified by using different controller types which are optimally tuned in an optimization framework. The different controllers include PI (proportional-integral) control, MPC (model predictive control), optimal linear control (via Q-parametrization) and an optimal open-loop control strategy that represents the best possible closed-loop performance.

In addition, the operability measures produced by each of these optimal control strategies are compared with one another in order to infer the *controller-independent* operability characteristics of the HEN itself. Two alternative HEN designs from a literature HEN example are compared in this way.

Furthermore, the operability characteristics of flexible designs is assessed. The flexibility of HEN designed is ensured through an optimization procedure in which HEN design variables are found that allow a HEN to operate feasibly for a range of possible operating conditions based on potential uncertainties in design parameters and possible disturbances. The economic impact of flexibility design is addressed, and then the dynamic operability of alternative flexible designs is assessed.

A software package customized towards the dynamic operability assessment of heat exchanger networks was developed for the studies presented here, and the functionality provided by the software is presented and explained.

The results show that the comparison of different dynamic operability analysis techniques, each specific to different control strategies, can provide valuable information about the operability characteristics of HENs themselves. In particular, the relative improvement in dynamic performance one finds when an operability analysis technique using a more advanced control strategy is used, allows one to infer how much controller complexity is required to reject poor operability characteristics in a particular HEN.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the study of process control has recognized that the closed-loop performance of a plant depends not only on the quality and tuning of the control strategy implemented, but also on the design of the process itself. In addition, the tightening of environmental constraints, movement towards global economic competitiveness and the elevated computational ability to perform increasingly complex designs has led to increased integration among process units.

While increased process integration holds tremendous economic benefit, it also results in increasingly complex dynamic characteristics. It has been shown that traditional process control design methods may not be able to provide sufficiently adequate closed-loop performance for such plants. Such studies (such as Anderson, 1966) typically show that dynamic control problems result from the design of the plant, and not the design of the control system.

Consequently, methods are required for assessing the dynamic performance possible for a particular design before a controller implementation is considered. Such tools will allow the designer to gauge alternative process designs based on their inherent operability in addition to the usual economic evaluations.

An example of a tightly integrated process is a heat exchanger network (HEN).

HENs are the most common industrial applications of process integration, and yet there has not been much study into their dynamic characteristics. Therefore, the dynamic operability assessment of HENs is considered in this work, although the methods presented are applicable to a wider range of processes.

This report begins by discussing various methods of operability analysis ranging from simple open-loop measures to vastly complex mixed-integer nonlinear optimization strategies, and evaluates both the advantages and shortcomings of each. The methodology behind the modeling and simulation of heat exchangers and networks built from these is given, along with an evaluation of the different methods that may be used.

A procedure for the design of flexible plants is presented, both for the general (non-process specific) case as well as for the flexible HENs design case so that the reader may easily visualize the application of flexibility issues in other spheres of process design. The study of flexibility is concerned with the feasibility of steady-state operation, and thus requires a steady-state process design model, for which the derivation is given.

Once a steady-state design has been found, the corresponding nonlinear dynamic model is set up in SIMULINK© . This model includes factors such as bypasses, utility exchangers, pipe delays, and the approximation of countercurrent heat transfer behaviour by multiple well-mixed cells in series.

However, the methods of dynamic operability assessment require dynamic models from which dynamic output trajectories may be calculated quickly and easily. For this reason, linear HEN models are identified from the nonlinear dynamic models and two methods are investigated: the use of the MATLAB© System Identification Toolbox, and a hand-coded least squares optimization strategy.

The methods of dynamic operability assessment as developed and implemented in this study are then presented. This method involves the calculation of a dy-

namic performance measure of the closed-loop response of a HEN subject to a step disturbance as a function of a discrete set of control methods. The optimization strategy for each control method is described in rigorous mathematical detail and the implications of each method are discussed.

As a service to the continuing research effort in this field, the computational code implemented for this work was prepared for easy use as a MATLAB© software package that is simple to operate and easily extendible. This software package is then described with particular attention to the aim behind the structure of the numerical routines and the functionality of the programs.

These computational methods are then demonstrated on a number of HEN examples based on literature examples, with one example including flexibility considerations. The results found in each investigation are presented and the implications of each result are discussed.

Finally, conclusions regarding the applicability of the methods of dynamic operability assessment presented in this document are put forward, and recommendations for further study are given.

Chapter 2

Review of Operability Assessment Methods

A number of different methods of dynamic operability assessment have been proposed in the literature, ranging from simple scalar open-loop indicators to highly complex mixed-integer nonlinear optimization problem formulations. In all cases, the goal is to find a means by which the dynamic operability of chemical plants may be assessed, either qualitatively or quantitatively.

The traditional approach to plant design is to use cost optimality of plant operation at steady state as a performance criterion, and then to consider plant control afterwards. In order to make allowance for uncertainties and changes in process operation, plants are 'overdesigned' through heuristic means, with this procedure based on designer experience (Downs and Ogunnaike, 1995).

However, recent trends in design such as the increased integration of processes to save on energy costs and capital costs and tighter environmental regulations have made the traditional plant design methods inappropriate, and alternative strategies and analysis techniques have become necessary.

The segregation of design and control functions is still common in the workplace

for two reasons: Historically, the control engineer is brought in after the design has been made and this paradigm is difficult to change in the current industrial workplace. The common notion is that steady-state process design alone determines the process economics. However, this becomes a poor approximation if it becomes too difficult to maintain operation at the predicted economically optimal steady-state design point. Overdesign should also take into account expected transient operating conditions, and not just different steady-state operating points.

Product quality is being measured as product variability and is becoming a discriminator among chemical suppliers (Downs and Ogunnaike, 1995). In addition, plants with good operability characteristics require little operator intervention. Much of the current improvement work on existing processes is to reduce process and product variability. This new focus on product variability has intensified the need to have processes with greater operability characteristics, are easy to operate and result in low product variability.

It is now widely recognized that process control should be considered from the earliest stages of process design, so that expensive control difficulties may be ironed out at an early stage.

The study of dynamic operability is concerned with the degree to which a plant can successfully be controlled. Most of the methods of dynamic operability assessment are based on the ability of a system to meet some criterion based on the quality of control that can be achieved, and are referred to by terms such as dynamic operability, resiliency and controllability.

Morari (1983) states that with increased process integration, it has become important to compare the dynamic operability characteristics (dynamic resilience) of alternate designs. Furthermore, dynamic resilience is determined by characteristics inherent in the system and is independent of the imposed controller structure and type.

One of the earliest published accounts of an actual industrial controllability problem arising from poor design is presented by Anderson (1966). Regardless of the control system design, stable process operation could not be achieved until the process was redesigned.

As a result of the formal training methods used to educate engineers, many enter the industrial workforce with a strong background in design and evaluation and relatively weak backgrounds in plant operation (Downs and Ogunnaike, 1995). The key in process design is knowing where to draw the line between the benefits arising from tight process integration and the costs of diminished process operability that may result. Processes that are easy to operate not only show the expected earnings and costs, but are also usually more amenable to modifications and improvements later on.

2.1 Flexibility Assessment

Plant flexibility may be defined as the ability a of plant to operate at feasibly at steady state for a range of operating conditions. Feasibility as used in this context means the satisfaction of physical and performance constraints, including mass and energy balances, purity constraints, safety margins on equipment and other steady-state design specifications (Swaney and Grossmann, 1985a).

The concept of flexibility is not new. Overdesign techniques have been used as a method of increasing plant flexibility for a number of years. However, it has been shown that empirical overdesign techniques may *worsen* plant flexibility.

Grossmann and Morari (1984) provide an example of a heat exchanger network that was overdesigned (in terms of heat exchanger area) to accommodate for uncertainty in the values of the overall heat transfer coefficients of the network, and was found to be infeasible. It was found, however, that when one of the exchangers in the network was underdesigned, the network operated feasibly for the same range

of uncertainty. Naturally, the benefit of flexibility techniques lies primarily at the design stage since it may not be viable to modify existing plant designs.

The steady-state model of a plant with feasibility constraints may be represented by

$$\begin{aligned} h(d,z,x,\theta) &= 0 \\ g(d,z,x,\theta) &\leq 0 \end{aligned} \tag{2.1}$$

where h is the vector of mass and energy balances associated with the steady state operation of the plant and g is the vector of inequalities associated with operating limits and product specifications. The vector d represents the design variables of the system, z the control variables, x the state variables and θ the vector of uncertain parameters. In many cases of chemical plant design, it is possible to specify to a certain degree of accuracy a permissible range for parameters used in that design. An example of this could be the rate constant used in reactor design, which might have a potential deviation of $\pm 10\%$ from the nominal value. The uncertain parameters θ are typically defined by bounds such as in

$$T(\delta) = \left\{ \theta \,\middle|\, \theta^N - \delta\Delta\theta^- \leq \theta \leq \theta^N + \delta\Delta\theta^+ \right\} \tag{2.2}$$

where T is the set of these uncertain parameter ranges, parametrized by a nonnegative scalar δ. $\Delta\theta^+$ and $\Delta\theta^-$ are the vectors of the expected positive and negative deviations of the uncertain parameters θ. Since x can be eliminated from g through h, the set of inequality constraints may be expressed as

$$h(d,z,x,\theta) = 0 \Longrightarrow x = x(d,z,\theta) \tag{2.3}$$

$$g(d,z,x(d,z,\theta),\theta) = f(d,z,\theta) \leq 0 \tag{2.4}$$

An early formulation for design under uncertainty is given by Nishida et al. (1971)

as:

$$\begin{aligned} &\min_{d} \max_{\theta} C(d, x, \theta) \\ &\text{such that } f(d, z, \theta) \leq 0 \\ &\theta^L \leq \theta \leq \theta^U \end{aligned} \tag{2.5}$$

The introduction of the min-max problem formulation intends to minimize the cost function with respect to d, the design variables, but maximize it with respect to θ, which results in a 'worst case scenario'. This formulation does not allow for the effect of *control* variables, implying that no change in the operation of the process system is allowed after design stage.

A new mathematical formulation was therefore derived, which rigorously ensures feasibility for every case; however, it was assumed that the critical points were vertices of the uncertain parameter region. The design strategy can be considered to be composed of two stages, an operating stage and a design stage, although the computational algorithm for this problem would essentially consist of one large optimization problem, and therefore the 'two-stage' description is somewhat artificial.

The computational effort required to solve these problem formulations is rather large, even for relatively small problems. Halemane and Grossmann (1983) have shown how the application of a projection-restriction strategy can minimize this problem. They deal largely with the optimization of flexible chemical plants in terms of multipurpose plants, or multiperiod plant operation where a fixed set of design variables is found that allows optimum operation for a set of discrete operating states.

Pai and Hughes (1987) have expanded on the idea of formulating the uncertainty problem as a two-stage problem as was first presented by Halemane and Grossmann (1983). The problem formulation may be described as finding an optimum cost function based on fixed design variables at the nominal steady-state operating point (design stage), while keeping operational flexibility in consideration (operating stage).

An index of flexibility has been proposed (Swaney and Grossmann, 1985a; Swaney and Grossmann, 1985b) which may be used to compare similar designs. The flexibility problem is set up in the following way. The flexibility index may be defined as the largest value δ may take while the system remains feasible, with δ defined as in equation 2.2. Expressed mathematically, this corresponds to

$$F = \max \delta \tag{2.6}$$

$$\text{s.t.} \left\{ \forall \theta \in T(\delta) \quad | \exists z | \; f(d, z, \theta) \leq 0 \right\}.$$

This measure of flexibility enables the designer to identify any flexibility bottlenecks in a particular design, and should allow a comparison of different designs on the basis of flexibility, and particularly the effect of each process flexibility on the associated cost function. Swaney and Grossmann (1985a) have provided a means of accomplishing this.

New mathematical formulations have been presented (Grossmann and Floudas, 1987) which are based on mixed integer programming, which do not assume that critical points are the vertices of the uncertain parameter region. The theory is based on the fact that an analysis of flexibility can be made in the constraint space that could be potentially active in limiting design flexibility.

Flexibility studies generally result in a trade-off between economic considerations and the degree of flexibility for that design. This concept can be taken further: by specifying soft flexibility constraints, an optimization procedure will realize that certain regions of infeasibility based on the corresponding uncertain parameter values need not penalize the design cost function if a large economic optimum at other regions of feasibility outweighs this. Clearly, this decision will occur only if the uncertain parameters are defined by statistical probability distributions and not with fixed bounds (Pistikopoulos and Mazzuchi, 1990).

Pistikopoulos (1995) has shown how one might choose a more general approach

in that both types of uncertainty may be taken into account and the inclusion of flexibility and steady-state open loop controllability in the synthesis of HENs is considered by Papalexandri and Pistikopoulos (1994).

Mohideen et al. (1996a) present a unified process synthesis framework for solving flexibility problems. Mohideen did introduce an integrated framework for flexibility *and* controllability at the design stage of dynamic systems under uncertainty in which interactions between process and control designs can be exploited to reveal potential benefits. The latter work (Mohideen et al., 1996b), however, considers the issue of stability which the previous work did not.

Most of the methods of flexibility analysis are being adapted in this way in order to make define a problem formulation that incorporates both steady-state and dynamic operation under uncertainty.

2.2 Dynamic Operability Assessment

2.2.1 Internal Model Control (IMC) Methods

Morari (1983) introduced the concept of *dynamic resilience* where he defines 'resilience' as the ability of a plant to move fast and smoothly from one operating condition to another (including start-up and shut-down) and to deal effectively with disturbances. In addition, dynamic resilience refers to the quality of the regulatory and the servo behaviour which can be obtained for the plant by feedback.

The objective of Morari's work is to provide the tools for the quantitative assessment of the resilience characteristics which are inherent in the plant itself and independent of the control system used. In addition, Morari and coworkers (Morari, 1983; Holt and Morari, 1985) developed resilience measures that are also important for the synthesis of control structures. One of the problems in control structure design is selecting the appropriate set of manipulated variables. By selecting alternate sets of manipulated variables, the input structure and therefore the system subjected to

feedback control is modified.

For single-variable control, heuristics have been developed over the years to guide the engineer making plant modifications in order to improve the quality of control. These are:

1. *Choose systems where the manipulated variable has a large effect on the controlled output.* This refers to the steady state behaviour of the process;

2. *Choose systems where the manipulated variable is 'close' to the controlled variable.* Physical closeness implies short time constants and small dead times and thus superior control performance;

3. *Avoid systems with inverse response characteristics.* Control difficulties associated with this type of practice are widely recognized in practice, see Shinskey (1988);

4. *Avoid systems with varying parameters and strong nonlinearities.* To guarantee acceptable performance over the range of possible system parameters, conservative controllers are required and thus system performance deteriorates.

For the multivariable problem, however, heuristics such as these were not so easy to apply. Dynamic simulation is the only tool commonly applied in industry to study the resilience of a new design. It is also generally necessary to have a controller present in this study—this makes it difficult to tell whether poor performance is a result of some inherent weakness in the design or simply a poor choice of controller tuning.

In the development of the IMC approach, Morari (1983) made the following assumptions:

1. The inputs and outputs have been scaled to have the same order of magnitude;

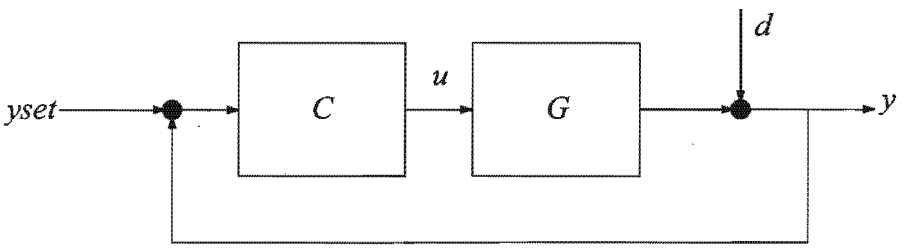

Figure 2.1: The classical feedback framework

2. The system outputs which are to be controlled are measured directly with the measurement noise which is negligible when compared with the process disturbances and model uncertainty effects;

3. The assumptions made about the controller are that it is linear and has constant parameters.

The IMC approach is based on the fundamental insight that the ultimate closed-loop behaviour is determined by constraints in the system. In the following discussion, refer to Figures 2.1 and 2.2.

Assume that a physical system can be represented by the system transfer matrix $G(s)$, i.e.

$$y(s) = G(s)u(s) + d(s) \tag{2.7}$$

where $y(s) \in R^n$ are the outputs to be controlled and $u(s) \in R^m$ the manipulated variables. In the classical feedback structure, the controller transfer matrix is $C(s) \in R^{m \times n}$, $y_s \in R^n$ the setpoints, and $d \in R^n$ the disturbances.

Ideally it is desired to have perfect regulatory and servo behaviour so that $y(t) = y_s(t)$ at all times and for all disturbances affecting the system which requires

$$u(s) = G^{-1}(s)\Big(y_s(s) - d(s)\Big) \tag{2.8}$$

where $G^{-1}(s)$ is the right inverse of $G(s)$. One can conclude that perfect control is only possible if G does in fact have a right inverse.

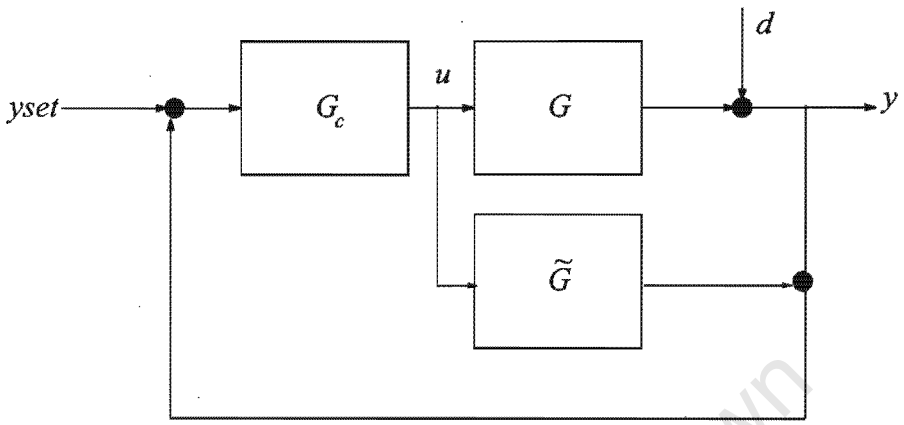

Figure 2.2: The IMC framework

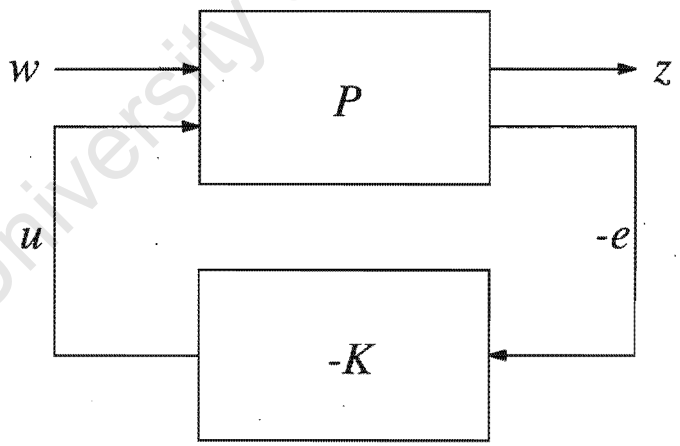

Figure 2.3: The general feedback framework of Boyd *et al.*

From a classical feedback block diagram, we can find

$$u = C\left(I + GC\right)^{-1}\left(y_s - d\right) \tag{2.9}$$

and $C(I+GC)^{-1}$ is called the *closed-loop controller transfer matrix (CLCTM)*. If C is made large, then $C(I+GC)^{-1} \approx G^{-1}$.

The conclusions that can be made from this discussion are the following:

1. Any feedback controller provides an approximate inverse of the plant transfer matrix. Perfect control is obtained when the CTLCM is equal to the right inverse of the system transfer matrix.

2. Closed loop control quality is limited by system invertibility. The goal is assessing controllability of the plant is then to identify system characteristics which **prevent** us from making the CLCTM equal to G^{-1}.

The system characteristics that limit prevent inversion of the plant model, and therefore prevent perfect control are:

1. Right-half-plane transmission (RHPT) zeros, which would cause unstable poles in the controller;

2. Time delays, which would cause predictive behaviour in the controller;

3. Input constraints;

4. Model uncertainty.

In the presence of any of these elements of G, it may be factored into an invertible part, G_-, and a non-invertible part, G_+

$$G = G_+ G_- \tag{2.10}$$

such that $\|G_+\|_2 = 1$ and G_-^{-1} is realizable and stable. G_c is then chosen as $G_c = \tilde{G}_-^{-1}$ and is the IMC controller. Note that it is not the same as C, the classical feedback controller as described in equation 2.9.

We then have the relationship $y = G_+(y_s - d) + d.$ and it is now apparent that the properties of G_+ will determine the achievable closed loop performance. This equation follows from the closed-loop relation for IMC structure for $G = \tilde{G}$:

$$y = G_c G\,(y_s - d) + d$$

RHPT Zeros

RHP *transmission* zeros are the MIMO analogs of the SISO case, and how they affect performance is not well understood. In the SISO case, zeros cause inverse responses, but this behaviour may or may not occur in the multivariable case. Morari (1983) explores how RHP transmission zeros affect resilience. In the SISO case, zeros are the roots of the numerator polynomial of the transfer function, e.g.

$$g_1(s) = \frac{-0.5s + 1}{(0.25s + 1)\,(0.3s + 1)} \quad \text{has a zero at} \quad s = 2. \tag{2.11}$$

However, for the MIMO case Holt and Morari (1985) give an illustration into the importance of properly defining zeros:

$$G(s) = \begin{bmatrix} \frac{-s+2}{s+2} & \frac{-s+5}{s+4} \\ \frac{-s+1}{s+2} & \frac{-s+3}{s+4} \end{bmatrix}. \tag{2.12}$$

While each element in this transfer function has a RHP zero, the overall system can be shown not to contain a single RHPT zero.

RHPT zeros can be obtained from the Smith-McMillan form of $G(s)$. It can be shown (Kailath, 1980) that if $\tilde{G}$ has transmission zeros in the RHP then *any* right inverse $\tilde{G}^{-1}$ is unstable. Holt and Morari (1985) found that when $\tilde{G}$ is non-square it has generally no finite zeros at all.

Using this methodology, Holt and Morari (1985) provide quantitative measures of how the location of the zeros affect the achievable performance for both single and multiple input systems. For MIMO systems it is shown how the impact of the zero can be isolated on one output.

Time Delays

In addition to the realization problem, time delays can also introduce an infinite number of poles into $\tilde{G}^{-1}$ and the stability criterion is not so easily confirmed.

Time delays are common in the process industries, a simple example of which is the piping of fluids. For a SISO process, Kwakernaak and Sivan (1972) showed that the ISE response to step inputs is minimized by choosing

$$\tilde{G}_+(s) = \prod_{i=1}^{m} \frac{\left(-\frac{1}{z_i}s + 1\right)}{\left(\frac{1}{z_i}s + 1\right)} e^{-\theta s} \tag{2.13}$$

where the process contains m RHP zeros z_i and exhibits a time delay of θ.

In the MIMO case, however the problem is made more complex since both the magnitude and distribution of the time-delays within the process transfer matrix affect operation.

Holt and Morari (1985) went further to develop two bounds to represent the minimum possible response time of a system with and without dynamic decoupling.

Input Constraints

Morari (1983) also shows how constraints on the manipulated variable affects resilience. For $\tilde{G} = G$,

$$\begin{aligned} \|u\|_2 &= \|G_c(i\omega)\|_2 \, \|y_s - d\|_2 \\ &= \left\|\tilde{G}_-^{-1}(i\omega)\right\|_2 \|y_s - d\|_2 \\ &= \left\|\tilde{G}_+(i\omega)\right\|_2 \left\|\tilde{G}^{-1}(i\omega)\right\|_2 \|y_s - d\|_2 \\ &= \left\|\tilde{G}^{-1}(i\omega)\right\|_2 \|y_s - d\|_2 . \end{aligned} \tag{2.14}$$

The constraint on the manipulated variable $\|u\|_2 \leq \|u\|_{\max}$ is always satisfied if $\|y_s - d\|_2 \leq \left\|\tilde{G}(i\omega)\right\|_2 \|u\|_{\max}$. If we normalize u to 1, we find that the amplitude ratio plot $\left\|\tilde{G}(i\omega)\right\|_2$ for the open loop system is also a plot of the maximum 'disturbance' $(y_s - d)$ which can be handled by the closed loop system when equipped with a 'perfect controller'. In particular, a disturbance exceeding $\|G(0)\|_2$ in the steady state always leads to offset due to saturation of the manipulated variable. Thus, the amplitude ratio plot may be used as a convenient measure of resilience.

Plant/Model Mismatch

The sources of model uncertainty include:

1. Unmodelled high-frequency dynamics;
2. Linearization of a nonlinear model around a steady-state operating point (usually the nominal operating point);
3. Operating conditions that lead to changes in the model parameters, such as heat exchanger fouling, which affects the heat transfer parameter;
4. Imperfect knowledge of the model parameters, and the order of the model at high frequencies.

Morari (1983) investigates the effect of plant/model mismatch (i.e. what if $\tilde{G} \neq G$). To illustrate this with a SISO treatment, assume that the plant is somewhere in a 'ball' of radius $l(\omega)$ around $\tilde{G}$ so that

$$G(s) = \big(1 + L_0(s)\big)\tilde{G}(s) \tag{2.15}$$

where $L_0(s)$ is constrained

$$\frac{\left\|G(i\omega) - \tilde{G}(i\omega)\right\|_2}{\left\|\tilde{G}(i\omega)\right\|_2} = \|L_0(i\omega)\|_2 \leq l(\omega) \tag{2.16}$$

If we assume the above equation, then a necessary and sufficient condition for robust stability (closed-loop stability under plant variations) is that the loop gain must be less than 1 (Morari, 1983).

$$\left\|\tilde{G}(i\omega)G_c(i\omega)\right\|_2 < \frac{1}{l(\omega)} \tag{2.17}$$

and if we select $G_c = \tilde{G}_-^{-1}$ then the above becomes

$$l(\omega) < 1. \tag{2.18}$$

This implies that the system is closed-loop stable only if the uncertainty radius $l(\omega)$ never exceeds 1. Since this will be violated at high frequency for all practical problems, we must add a dynamic compensator, so that the controller has the form

$$G_c(s) = \tilde{G}_-^{-1}(s)F(s) \tag{2.19}$$

with $F(0) = 1$. The purpose of F is to lower $\|G_c\|_2$ at all high frequencies to make the system robust against model uncertainties.

Robust stability of MIMO systems

Robust stability of MIMO systems may be dealt with in the following way (Morari, 1983). Because matrix multiplication is not commutative, we must distinguish between uncertainty associated with the system inputs

$$G(s) = \tilde{G}(s)(I + L_I(s)) \tag{2.20}$$

and the system outputs

$$G(s) = (I + L_O(s))\tilde{G}(s) \tag{2.21}$$

where either L_O or L_I is constrained

$$\frac{\left\|G(i\omega) - \tilde{G}(i\omega)\right\|_2}{\left\|\tilde{G}(i\omega)\right\|_2} = \|L(i\omega)\|_2 \leq l(\omega) \qquad L \in \{L_I, L_O\} \tag{2.22}$$

Note that L_I and L_O are square matrices and will have different dimensions when G is non-square.

The following results are given in Morari (1983): If G_c, $\tilde{G}$ and L_O are open-loop stable, the system will be closed loop stable for all $L_O(s)$ satisfying $\|L_O(i\omega)\|_2 < l(\omega)$ if and only if

$$\left\|\tilde{G}G_c(i\omega)\right\|_2 = \sigma_M(\tilde{G}G_c(i\omega)) < \frac{1}{l(\omega)}$$

where σ_M is the maximum singular value of G.

If $L(s)$ occurs at the *inputs* then

$$\left\|G_c\tilde{G}(i\omega)\right\|_2 = \sigma_M(G_c\tilde{G}(i\omega)) < \frac{1}{l(\omega)}$$

and if both are to be satisfied then

$$\left\|G_c(i\omega)\right\|_2 \left\|\tilde{G}(i\omega)\right\|_2 < \frac{1}{l(\omega)}.$$

If we include a dynamic filter as in the single-variable case, the stability requirement becomes

$$\gamma(\omega) \equiv \left\|\tilde{G}^{-1}(i\omega)\right\|_2 \left\|\tilde{G}(i\omega)\right\|_2 = \frac{\sigma_M(\tilde{G})}{\sigma_m(\tilde{G})} < \frac{1}{\sigma_M(F)l(\omega)} \tag{2.23}$$

where σ_m is the minimum singular value of G.

The result is that the larger $\gamma(\omega)$, the more sensitive the control performance to a possible plant/model mismatch will be. Therefore $\gamma(\omega)$ is called the sensitivity function of the system, and it is a system inherent property independent of the controller. This result provides an indication of possible sensitivity to model error since the method is conservative, as stated previously.

Saboo and Morari (1984) continued with the IMC structure and developed a resilience index which was applied to HENs. Saboo and Morari show how HENs designed on the basis of heuristics that may seem plausible are generally inefficient, and often cannot satisfy target temperature specifications when operating conditions

change. In addition, it is shown that a greater degree network resilience may be obtained by proper placement of exchangers in the network, rather than adding extra exchangers to make up for poor design. Saboo and Morari (1984) developed a theoretically rigorous synthesis technique for designing HENs that are able to cope with input temperature variations whilst maintaining a high degree of energy efficiency. The method is, however, restricted to a certain class of problems in which changes in the pinch point[1] are dependent only on one of the streams in the network in light of variations in operating conditions.

2.2.2 Optimization-based methods

Palazoglu and Arkun (1986) developed an optimization-based approach to design chemical plants with robust, dynamic operability characteristics. A nonlinear programming approach is taken for the simultaneous treatment of both steady-state and dynamic constraints. The problem is formulated within a multiobjective optimization framework and makes extensive use of singular-value decomposition and nonlinear semi-infinite programming techniques.

The measure of operability used by Palazoglu and Arkun (1986) are the robustness indices which are placed as constraints within the optimization scheme, and their problem formulation uses the IMC framework of Morari (1983).

Bahri et al. (1995) provided a systematic way to find a quantitative measure of flexibility and controllability. Using this procedure, the designer can systematically evaluate different process alternatives and/or control systems, while maximizing or minimizing an economic objective, as well as reducing the distance between the selected operating point and the optimum. Thus controllability was expressed in terms of regulatory performance (disturbance rejection).

[1]The pinch point is related to the minimum amount of additional heating or cooling required by the network to be able to exactly satisfy target temperatures.

Bahri et al. (1996) include additional controllability aspects and make use of an integrated solution strategy based on a MINLP formulation. The measures of operability used are the squared errors between outputs and their setpoints and a bound on the response time of the system. It should be noted, however, that these measures of controllability are based on a worst case scenario .

These performance measures included in the optimization problem result in the final plant:

1. Operating feasibly over a given range of disturbances (flexibility criterion);
2. Operates in the steady state condition which has the minimum distance from the optimum (economic or otherwise);
3. Takes the least possible time to reach the original steady-state after a disturbance change (controllability measure), and;
4. Responds to the entering disturbances smoothly (operability criterion).

The optimization-based framework for dynamic operability assessment allows one to include flexibility requirements as well as operability requirements in the same problem formulation (Mohideen et al., 1996). The drawback with these methods, however, is that they tend to have steep computational requirements.

Q-parametrization

Controller parametrization may also be used within an optimization framework for dynamic operability analysis. Boyd et al. (1990) formulated a program to find the optimal linear feedback controller for a plant using technique called Q-parametrization as described by Youla et al. (1976).

Referring to the general feedback framework in Figure 2.3, the inputs are partitioned into two vector signals. The first signal is actuator input vector, u, which are

all inputs that can be manipulated by the controller. The second vector signal is the exogenous input vector, w, which are all other input signals like disturbances and setpoints.

The outputs also consist of two vector signals, namely the sensor output vector y which are the signals accessible to the controller, and the regulated output vector z which are signals that may be required to meet some given performance specifications. Note that P is not the plant model of classical controllers, but rather a mapping of all inputs of interest to all outputs of interest. Thus, P is defined as:

$$\begin{bmatrix} z \\ y \end{bmatrix} = P \begin{bmatrix} w \\ u \end{bmatrix} \tag{2.24}$$

The plant P can then be described by a set of transfer functions for each input to output pairing as in

$$P = \begin{bmatrix} P_{zw} & P_{zu} \\ P_{yw} & P_{yu} \end{bmatrix} \tag{2.25}$$

One of the requirements of the controller K is that it must be linear time-invariant and lumped. Forming the closed-loop transfer matrix from w to z yields

$$H_{zw} = P_{zw} - P_{zu}K(I + P_{yu}K)^{-1}P_{yw} \tag{2.26}$$

An alternative parametrization of the closed-loop maps is given by the following: if we let H_{stab} represent the set of H_{zw}'s achievable with stabilizing controllers, then

$$H_{stab} = \{ H_{zw} = T_1 + T_2 Q T_3 \mid Q \text{ stable} \} \tag{2.27}$$

where Q is any stable transfer function matrix, and is the free parameter in the above equation. The T_i's are stable transfer matrices that may be determined from co-prime factorizations of the plant model P_{yu} and a nominal stabilizing controller. Since H_{zw} is affine in Q, closed loop specifications convex in H_{zw} are convex in Q. It should be noted that few robustness specifications are closed-loop convex.

Using the above expressions, it is then possible to formulate a convex optimization problem to calculate the limit of achievable performance for some linear controller Q, as in

$$\begin{aligned} & \min_{Q \in \bar{\Omega}} \ \bar{\Phi}(Q) \\ \text{where } \bar{\Omega} &= \{Q \mid \ T_1 + T_2 Q T_3 \in \Omega, \ \ Q \text{ stable}\} \\ \text{and } \bar{\Phi}(Q) &= \Phi(T_1 + T_2 Q T_3) \end{aligned} \tag{2.28}$$

where $\bar{\Phi}(Q)$ is a convex functional of Q and Ω is the set of all H_{zw}'s that satisfy the performance constraints.

Q is the set of all stable transfer function matrices which is infinite dimensional, but can discretized by:

$$Q_{ij} = \sum_{k=0}^{L} q_{ij}(k) z^{-k} \tag{2.29}$$

The application of Q-parametrization to operability assessment has been investigated by Swartz (1996) and Ross and Swartz (1995, 1997). The primary advantage of the Q-parametrization method is that it allows one to determine the best possible performance of a closed loop system using a linear controller, with no other requirement on structure. A disadvantage of the problem formulation is that the method as posed above is limited to linear control.

The Optimal Open-Loop Input Trajectory method (OOLIT)

This method of operability assessment has been developed largely by Perkins and coworkers (Perkins and Wong, 1985; Russell and Perkins, 1987; Cao et al., 1994). This method is in part based on the concept of functional controllability first introduced by Rosenbrock (1970).

The method seeks to find the input trajectory to a plant that results in the optimal output trajectory in an open-loop formulation. Mathematically, this is represented

by

$$J = \min_{u} \int_0^{t_f} (y - y_{sp})^T W (y - y_{sp})\, dt \tag{2.30}$$
$$\text{s.t. } f(\dot{x}, x, y, u, t) = 0$$

$$\begin{aligned} x(0) &= x_0 \\ u^L \leq\ u &\leq u^U \\ \dot{x}(t_f) &= 0 \\ y(t_f) &= y_{sp} \end{aligned}$$

where y_{sp} is the set of step functions representing the desired output trajectory, W is diagonal weighting matrix, and x is the vector of system states. The vector of the manipulated variables u may be constrained between upper and lower bounds. Note that in the formulation above, the objective function to be minimized is the MIMO ISE measure, which is the measure of performance used .

The result of this problem formulation is the best achievable performance of the system, regardless of controller structure used. It thus provides an upper bound of dynamic operability assessment.

Cao et al. (1994) shows how a modified singular value technique and a modified form of the above optimization approach predicts the closed-loop performance in the presence of control constraints.

2.3 Open-loop indicators of dynamic operability

Open loop indicators may also be used to assess dynamic operability. The advantages of open-loop indicators are that they represent controller-independent measures and thus give an indication of the controllability of the plant itself, and open-loop indicators are not as computationally intensive as other methods of dynamic operability assessment such as the optimization based methods.

Open-loop indicators, as defined in this paper, are based on a linearized model of the process:

$$\left\{ \begin{array}{l} \frac{dx}{dt} = f(x,y,u) \\ g(x,y,u) = 0 \end{array} \right\}^{2} \quad \text{linearize} \implies \quad \left(\begin{array}{l} \frac{dx}{dt} = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{array} \right) \tag{2.31}$$

A process may be said to be functionally controllable (Rosenbrock, 1970) if inputs u can be found that can drive the plant along a smooth trajectory y, i.e. if $G(s)$ is non-singular (invertible). Singular value analysis makes use of the l_2 vector and matrix norms. The singular values of a matrix A, for example, are defined as the positive square roots of the eigenvalues of A^TA. For dynamic systems, vector and matrix norms are functions of frequency, ω.

Scaling is important for calculating norms: For l_2, each vector is scaled by dividing by the steady state nominal value. With $u = G^{-1}y$ and RHP zeros in G become poles are therefore detrimental to stability, especially near zero. Similarly, time delay elements in G become predictive terms in G^{-1} which is physically impossible.

Skogestad et al. (1988) made an investigation of the robust control of ill-conditioned plants, using a high purity distillation case study to illustrate the proposed methodologies.

Barton et al. (1991) describes the use of various open-loop indicators of dynamic stability/performance including matrix condition numbers, the evaluation of a minimum singular value, a MIMO interaction measure defined by angle and the evaluation of RHP zeros and their effect on stability.

Hovd and Skogestad (1992) have explored relationships between the relative gain array (RGA) and RHP zeros, and in particular, the use of the RGA as a sensitivity measure with respect to individual element uncertainty and diagonal input uncertainty. They show how frequency-dependent plots of the RGA and the CLDG

[2]That is, since y is a function of x and u, it can be equal to zero when defined as a function of x, u and itself.

(closed-loop disturbance gains) can be used to evaluate achievable performance of a plant under decentralized control.

These measures are controller independent and they give constraints on the design of the individual loops which, if satisfied, will guarantee that the overall system will satisfy the performance objectives with regard to setpoint tracking and disturbance rejection.

Hovd and Skogestad (1992) acknowledge that the RGA was originally intended for steady state, but Bristol (1966) shows how the method can be extended to higher frequencies. One of the foremost advantages of the RGA is that it depends solely on the plant and not the controller, and it is scaling independent. In addition, Hovd and Skogestad (1992) have introduced a new measure of performance based on the RGA called the *performance* RGA (PRGA). These, and other open-loop indicators used later in this work are described here.

2.3.1 The Relative Gain Array (RGA)

The relative gain array (RGA) as presented by Bristol (1966) may be described in the following way: Assume the plant is $y(s) = G(s)u(s)$ and let the open loop gain from u_j to y_i be $g_{ij}(s)$. The closed loop gain from u_j to y_i is $1/[G^{-1}(s)]_{ji}$ when all other y's are perfectly controlled. The matrix of relative gains 'open loop' vs. 'closed loop' forms the RGA, computed using

$$\Lambda(s) = G(s) \times \left(G^{-1}(s)\right)^T$$

where $\times$ denotes the Schur complement[3].

The frequency dependent RGA is a simple extension of the steady-state RGA, whereby the RGA is computed as a function of frequency. Skogestad and Postlethwaite (1996) show how the frequency dependent RGA may be used as a measure of

[3]That is, element by element multiplication

control performance.

2.3.2 The Performance Relative Gain Array (PRGA)

Wolff et al. (1992) note that a potential inadequacy of the RGA is that it may not indicate interaction problems while significant one-way coupling may exist. Consequently, Hovd and Skogestad (1992) introduced the performance relative gain array (PRGA), defined as

$$\Gamma(s) = G_{diag}(s)G(s)^{-1}$$

where $G_{diag}(s)$ is the matrix consisting only of the diagonal elements of $G(s)$. Wolff et al. (1992) mention that it is preferred for the elements of $\Gamma(s)$ to be small at low frequency, and for the matrix to be triangular at high frequency, with the diagonal values close to 1 (corresponding to the input-output pairings for decentralized control).

2.3.3 The Condition Number (γ) and Disturbance Condition Number (γ_d)

The disturbance condition number, introduced by Skogestad and Morari (1987), may be computed as

$$\gamma_d = \frac{\|G^{-1}g_d\|_2}{\|g_d\|_2}\sigma_M(G) \tag{2.32}$$

where g_d is a column vector of the disturbance transfer function matrix G_d for the specific disturbance being evaluated and σ_M is the maximum singular value of G. This measure may be used in conjunction with the condition number to determine whether the disturbances to a plant lie in directions that may be difficult of control. If γ_d is close to unity for all frequencies, the plant will likely be easy to control, and if γ_d is close to the plant condition number[4] $\gamma = \sigma_M/\sigma_m$, the plant is likely to suffer

[4] σ_m is the minimum singular value of a process transfer function

from poor control as the disturbances lie in a direction corresponding to low plant gain.

2.3.4 The Closed-loop Disturbance Gain (CLDG)

As a measure of disturbance rejection, Wolff and Skogestad (1992) describe the closed-loop disturbance gain (CLDG) for decentralized control, where they define it as

$$\Delta_{CLDG} = G_{diag} G^{-1} G_d \tag{2.33}$$

Wolff and Skogestad (1992) explain how the elements of Δ_{CLDG} represent the disturbance gain from each disturbance to each output, and can therefore be used to determine the effect of the disturbances on the plant using decentralized control.

2.3.5 The Morari Resiliency Index (MRI)

The Morari resiliency index (MRI) is defined as the minimum singular value of a plant transfer matrix (open-loop) and gives a measure of the inherent controllability of the plant (Morari, 1983). An example of its use is presented by Chiang and Luyben (1988). The MRI is a relative measure and should be used for comparing different design alternatives. Greater values of the MRI indicate greater controllability while smaller values indicate less controllability. Yu and Luyben (1986) use the MRI to guide the selection of manipulated variables, and note that it is a measure of the inherent ability of a process (control structure) to handle disturbances, plant-model mismatches and changes in operating conditions.

2.3.6 The Method of Psarris and Floudas

While this method has not been used in this work, it deserves a brief description since it is quite different from the other methods already described. It has previously been stated how Morari has diagnosed the factors which limit the achievable performance

of a plant. Psarris and Floudas (1991a) have focused specifically on the nonminimum phase elements of time delays and transmission zeros, and have developed methods which may be used to analyze the effects of these elements.

Psarris and Floudas (1991b) state that multivariable processes with time delays usually have infinite RHPT zeros, which makes their numerical determination difficult. The effect of the RHPT zeroes is inversely proportional to their closeness to the origin in the complex plane.

The objective of the work done by Psarris and Floudas (1991b) is to suggest process design modifications to enhance the dynamic operability of MIMO delay systems with infinite RHPT zeroes. They demonstrate that the closed loop performance of systems with time delays could be improved by *increasing* certain delays of the transfer function matrix of the process.

In addition, a framework is proposed which will allow the identification of which delays should be increased, as well as the magnitude of the change necessary to accomplish this. The suggested approach then leaves only finite RHP zeroes, which can then be identified using the argument principle, and their location can be evaluated numerically.

2.4 Summary of operability evaluation methods

The following methods of operability assessment have been described:

1. Flexibility analyses: these are typically optimization based approaches conducted at steady-state plant conditions for which feasible operation is ensured under conditions of parametric uncertainty or slow-varying (persistent) disturbances;

2. Dynamic operability analyses including:

   (a) Methods used in Internal Model Control (IMC)

(b) Optimization based methods

(c) Open-loop measures

Factors that inhibit the closed loop performance of a plant need to be considered simultaneously in operability analyses. Optimization based formulations do this, but aside from small problems, they are often difficult to solve.

Each measure of operability appears to have a different measure of performance. It would be beneficial to the field if a comparison or a methodology for comparison between the various operability evaluation methods was made. For example, if the open-loop input trajectory optimization of Perkins found a similar achievable limit of performance for a plant as the method of Q-parametrization, it can be said that for the plant in question, there may not be much gain in exploring the use of nonlinear control techniques. This is because the open-loop method finds the limit of performance for all controller types while Q-parametrization finds a limit of performance for the best linear controller possible.

In this way, one can assess the effect more advanced control techniques will have on a plant with control difficulties, if any.

Chapter 3

Modelling and Operation of Heat Exchanger Networks

Rather than model the steady-state and dynamic behaviour of heat exchanger systems in as rigorous a level of detail as possible, the intention in this work was rather to generate *typical* behaviour of heat exchangers in as simple a framework as possible while maintaining a good degree of accuracy with real systems, particularly at steady-state. Consequently, this chapter presents the methodology followed in setting up a usable model of heat exchanger networks.

3.1 HEN studies in open literature

There are a number of features of HENs that make them particularly appropriate for operability studies. These include:

- Well defined operating constraints in the form of target temperatures;
- Well defined design variables in the form of heat transfer areas;
- Economics that is easily quantified from costing data related to heat transfer area and steam and cooling water utility costs;

- Manipulated variables that are naturally constrained by logical bypass fraction limits of 0 and 1, and;
- Parametric uncertainty that is easily conceptualized and implemented in the form of uncertainty in heat transfer coefficients which may be studied in flexibility analyses. Indeed, the very fact that fouling occurs in heat exchangers means that uncertainty in the values of heat transfer coefficients is inevitable, and must always be considered in design.

Many other types of processes exhibit some or all of these characteristics, and thus research involving the flexible design and operability analysis of HENs may apply equally well to a host of other operations.

The primary focus of HEN studies in literature has been the *heat exchanger network synthesis problem*, as described in Gundersen et al. (1991). This form of study is concerned with discrete decisions about the network structure such as which streams should be matched with each other, the sequence in which different process exchangers should occur, as well as the determination of the continuous variables of heat exchanger network operation, such as heat loads and stream splitting ratios. The aim of such studies is typically to find the optimal structure of a HEN application so that the process is cost optimal[1] at steady state. One method of solving this problem is the mathematical programming of a large scale combinatorial framework that usually results in a non-convex formulation with multiple local minima.

Another method of solution, Pinch technology, is a sequential method of HEN design in which an initial design is 'evolved' to the final solution in order to avoid the combinatorial problem. Gundersen et al. (1991) make the observation that both of these methods have significant limitations that may prevent the global structural design solution being found.

[1] smallest area requirements and lowest utility requirements to ensure feasible operation.

However, it is important to note that both of these approaches that are fairly well known and commonly used in this field are conducted at *steady state*. In fact, the steady-state economic optimal synthesis of HENs is a mature research field, and Gundersen and Naess (1988) cite nearly 200 articles and papers in a review of this area of research. It is interesting that the amount of work dedicated to the operation and control of HENs is much less.

With regard to flexibility analyses in open literature, Kotjabasakis and Linnhoff (1987) have investigated the use of sensitivity tables in order to make a HEN design flexible, while Papalexandri and Pistikopoulos (1994) introduce flexibility requirements into the mathematical programming of a HEN synthesis procedure.

With regard to controllability studies in open literature, Shinskey (1988) and others have looked at the control of single heat exchangers. For HENs, Nisenfeld (1973) looked at the use of the Relative Gain Array[2] to find the best pairing between measurements and manipulations. Mathisen and Skogestad (1992) has investigated the controllability and control structure selection of HENs, and has included economic aspects in a later work (Mathisen, 1994), while Marselle et al. (1982) has explored the use of heuristic rules in the design of operable HENs.

Glemmestad (1997) notes that while process integration is motivated from economic benefits, the characteristics of the plant are likely to be very different from the characteristics of the individual plants that are integrated. In particular, process integration

- Increases the level of *interaction* in the plant, and;
- May dramatically change the plant dynamics. The integration of units that are individually stable may result in an unstable plant.

[2]See §2.3.1 on page 26

An alternative HEN design problem was posed by Mathisen (1994) as follows:

> "For a HEN with given structure, heat exchanger areas and bypasses, and a given steady-state operating point (supply temperatures, heat capacity flowrates, heat transfer coefficients and target temperatures), find the set of manipulated inputs (bypasses and possible split fractions) that minimizes energy cost."

This problem statement is adequate only for pairing studies, and does not address the issue of dynamic operability.

3.2 Operation and Control

Glemmestad (1997) specifies four categories of manipulated inputs that may be used to control HENs:

1. Bypass fractions across a single heat exchanger;
2. Bypass fractions across multiple heat exchangers;
3. Duty of utility exchangers, and;
4. Split fractions.

The implementation of a bypass across a single heat exchanger may be implemented in practice with a three-way control valve as the splitter, a flow transmitter in each of the process streams and a flow ratio controller. It is possible to install a typical control valve in the bypass stream, but this option will reduce the operating range of the manipulated input and could possibly add additional pressure drop across the exchanger.

In this thesis however, the mechanical details of the bypass manipulation are not considered, and it is assumed that the bypass fraction variable directly determines

the fraction of flow of the bypassed stream diverted from the heat exchanger feed and then added to the exit stream.

In many literature studies, the utility exchanger duty is used directly as the manipulated variable. In this thesis, however, utility exchangers are modeled with the same level of detail as the process heat exchangers. This not only provides a more complete picture of the dynamics of HENs, but also gives a physical interpretation to the significance of utility exchanger duties in a more direct way, i.e. in terms of utility heat capacity flows.

Mathisen (1994) gives several heuristic guidelines for pairing manipulated inputs and outputs, but the primary rule is that manipulated inputs should be paired with outputs on which they have the most direct effect. For HEN examples used in this thesis, this rule has been applied where no information on the pairing has been given in the original article.

3.3 Dynamic modeling of heat exchangers

The goal in the modeling of heat exchanger networks was to develop a dynamic model that exhibits behaviour typical of such systems while remaining fairly simple. Simplicity is necessary for isolating the significant design factors of heat exchangers that pertain to their operability, as well as allowing quick computational dynamic simulation.

3.3.1 Differential equations

Heat transfer in a counter-current heat exchanger is described by partial differential equations. However, the system is approximated here as a series of well-mixed tanks which removes space as an independent variable.

If we describe the heat transfer between two countercurrent process flow streams in well mixed cell, then referring to Figure 3.1, the following differential equations

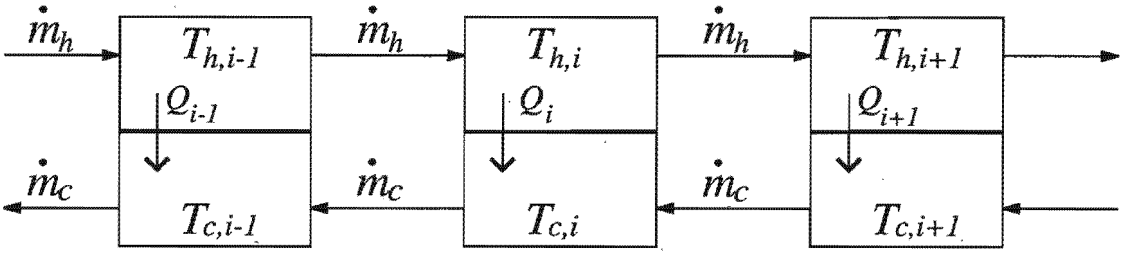

Figure 3.1: A mixing tank model for countercurrent heat exchange.

apply:

$$\begin{aligned}\frac{dT_{h,i}}{dt} &= \frac{\dot{m}_h}{\rho_h V_{h,i}}(T_{h,i-1} - T_{h,i}) - \frac{UA_i}{\rho_h V_{h,i} c_{p,h}}\Delta T_{m,i} \\ \frac{dT_{c,i}}{dt} &= \frac{\dot{m}_c}{\rho_c V_{c,i}}(T_{c,i+1} - T_{c,i}) + \frac{UA_i}{\rho_c V_{c,i} c_{p,c}}\Delta T_{m,i}\end{aligned} \tag{3.1}$$

where the subscripts h and c refer to 'hot' and 'cold', $\dot{m}$ refers to mass flowrate, U refers to the heat transfer coefficient between the hot and cold partitions in the cell, V refers to the volume in each *partition* of the cell, ρ refers to density, c_p refers to heat capacity, A refers to the heat transfer area between each partition in the cell, and T refers to temperature.

Note that the temperature inside the cell is the same as the exit temperature, since an assumption has been made that the cell is well-mixed. In addition, most literature works use the 'heat capacity flowrate' as a measure of flow rather than volumetric flowrate—however, this is a simple transformation through density ρ and heat capacity C_p.

The differential equations above represent the basic mechanism for countercurrent heat exchange: There are still two important factors to consider, namely the inclusion of heat exchanger bypasses as a manipulated variable for dynamic heat exchange, and the characterization of the heat transferred.

Incorporation of bypasses in a dynamic heat exchanger model

The use of a bypass affects the dynamic model only at the entrance and exit points. The mass flow entering the heat exchanger is typically reduced according to the setting of the bypass value which lies between 0 and 1. The relationship may be expressed by

$$\dot{m}_e = (1-u)\,\dot{m} \tag{3.2}$$

where $\dot{m}_e$ represents the effective mass flow through the heat exchanger and u represents the fractional bypass of a feed stream to the heat exchanger.

The bypass stream is modeled using the same principle as the heat exchanger, and the energy balance for a well-mixed cell in the bypass stream can be expressed as

$$\rho V_j C_p \frac{dT_j}{dt} = u\,\dot{m}\,C_p T_{j-1} - u\,\dot{m}\,C_p T_j \tag{3.3}$$

where V_j is the volume of each of the hot and cold sides in a well-mixed cell[3]. The exit stream from the bypass unit[4] must then be mixed with the exit stream of the heat exchanger unit, for which a mass-weighted average of the two temperatures is used:

$$T_{out} = u\,T_{bypass} + (1-u)T_{exchanger} \tag{3.4}$$

Characterization of the heat transferred, Q

The standard relationship used to calculate the heat transferred is

$$Q = U \cdot A \cdot \Delta T_m \tag{3.5}$$

where U represents the heat transfer coefficient, A the heat transfer area, and ΔT_m some representative measure of the temperature difference (the driving force for heat transfer). There are three methods which may be used to quantify ΔT_m:

[3]Thus $V_j = \frac{V_{\text{total, 1 side}}}{N}$ where N is the number of well-mixed cells approximating the heat exchanger.

[4]'unit' refers to the set of well-mixed cells

1. The temperature difference:

$$\Delta T_m = T_h - T_c \tag{3.6}$$

2. The arithmetic mean temperature difference (AMTD):

$$\Delta T_m = 0.5\left[(T_{h,in} - T_c) + (T_h - T_{c,in})\right] \tag{3.7}$$

3. The logarithmic mean temperature difference (LMTD):

$$\Delta T_m = \frac{(T_{h,in} - T_c) - (T_h - T_{c,in})}{\ln\left[\frac{(T_{h,in} - T_c)}{(T_h - T_{c,in})}\right]} \tag{3.8}$$

The first option results in a significant steady-state error if few well-mixed cells are used, and the heat duty is typically underestimated. The AMTD method significantly decreases the steady-state error as found with the first method, and the LMTD represents the correct solution for ideal countercurrent heat exchange at steady state.

As shown in Figure 3.2, as long as the ratio of temperature differences at each end of a countercurrent well-mixed heat exchange cell is near unity, the AMTD approximation for one cell corresponds closely with the LMTD method.

In addition, the LMTD method is not appropriate for use in dynamic simulation since it is undefined when $(T_{h,in} - T_c)$ and $(T_h - T_{c,in})$ are equal. Thus, the AMTD has been used in these studies. The error in using the AMTD method decreases further when a number of well-mixed cells are used in series.

Mathisen (1994) recommends that at least 4 well-mixed cells be used to describe a heat exchanger if the AMTD method for the temperature difference driving force is used, and this recommendation has been followed in this thesis.

3.3.2 Assumptions and simplifications

The following assumptions and simplifications have been made:

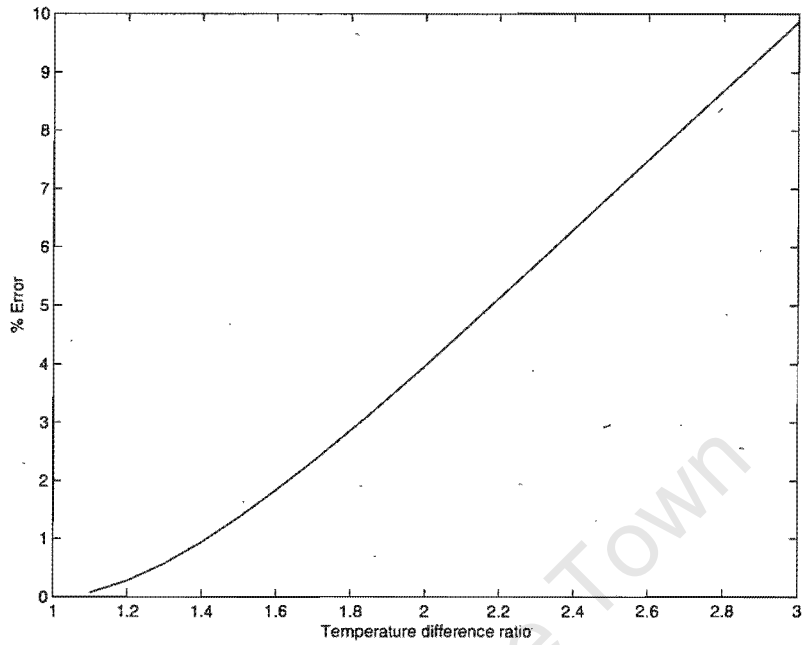

Figure 3.2: Relative error between the logarithmic and arithmetic mean temperature difference driving force relationships

1. A nominal heat transfer coefficient of $400\frac{W}{K \cdot m^2}$ has been used for all heat exchangers considered. This figure is used by Glemmestad (1997). Since all HEN design examples presented in this work are based on examples from literature examples, and in general these examples do not supply heat transfer coefficient data. The magnitude of this estimate has been found to be comparable to the typical values encountered in the design of heat transfer equipment.

2. The material properties for all streams have been estimated to be those of water at standard conditions. As in point 1 above, literature examples in general do not supply information about the heat capacity of streams, for example, and thus it is required that an estimate be made.

3. The volume for the hot and cold sides of each heat exchanger have been assumed to be the same. This is in general not true as in the case of shell and tube heat

exchangers, but it is an assumption that allows simplification of the dynamic model with very little significant impact on the results of this study.

3.3.3 SIMULINK Implementation

The SIMULINK dynamic simulation toolbox that forms part of the MATLAB© 5.3 package is an ideal environment in which to model dynamic systems. The graphical user interface and block-connectivity functionality of SIMULINK© implementation releases the designer from computational issues, and allows the focus to be on the representation of the physical system under investigation.

A dynamic heat exchanger model was set up in SIMULINK© , and a typical structure of this model is described in Appendix A. The complete heat exchanger networks under investigation were then constructed by connecting single heat exchanger models in the correct sequences.

3.4 Steady-state model

The steady-state model may readily be derived from the dynamic model equation set 3.1 by setting $\frac{dT_x}{dt}$ to 0:

$$\begin{aligned} \text{Hot side:} \quad 0 &= \dot{m}_h C_{p,h} T_{h,in} - \dot{m}_h C_{p,h} T_h - Q \qquad \text{(for 1 well-mixed cell)} \quad (3.9) \\ \text{Cold side:} \quad 0 &= \dot{m}_c C_{p,c} T_{c,in} - \dot{m}_c C_{p,c} T_h + Q \qquad \text{(for 1 well-mixed cell)} \end{aligned}$$

These equations are repeated for each well-mixed cell in the dynamic heat exchanger model, and the complete set of steady-state equations may be solved as a linear problem to find the exit temperatures for the hot and cold sides.

3.4.1 Design and operating costs of heat exchanger networks

The heat exchanger costing data used in calculating network economics in the flexibility studies presented later is taken as that given by Gundersen et al. (1991).

- Cost of exchangers:

$$C_i = 8600 + 670A_i^{0.83} \quad (1991\ \$) \tag{3.10}$$

- Cost of hot utilities

$$C_{HU,i} = 0.03\frac{(1991\ \$)}{kWh} \tag{3.11}$$

- Cost of cold utilities

$$C_{CU} = 0.003\frac{(1991\ \$)}{kWh} \tag{3.12}$$

A_i refers to the area in m^2 of each exchanger i in a HEN.

Chapter 4

Operability Assessment Techniques for Heat Exchanger Networks

In this chapter, the alternative methods of dynamic operability assessment are proposed and discussed. These methods are quantitative and the results found are plant-specific. In addition, information about the relationship between the plant and the complexity of the control structure employed may found as a direct consequence of application of these methods. The methods involve finding the optimal controller tunings for four different controller structures in an optimization framework in which the setpoint error trajectory resulting from a step disturbance is minimized in the closed loop. The aims of this project are presented, and the implementation and significance of each method is discussed in a discrete time environment for which optimization formulations are given. The computational aspects and implementation of these methods is presented in Chapter 5.

4.1 Project Aims

The aims of this study are threefold:

1. To compare several operability analysis techniques as applied to the dynamic operation of HENs under load disturbances.

2. To demonstrate how the comparison above may be used to infer the operability characteristics of HENs (although the methods are applicable to a wider range of processes).

3. To develop a software package that allows implementation of the above studies.

4. To investigate whether each optimally tuned control method predicts a consistent performance ranking for different alternative designs.

Because HENs are integrated systems, it is believed that operability analysis should play a key role in their design. As is the case with most integrated systems, problems encountered in operation and control are usually non-intuitive, and thus there is a need for techniques that may quantitatively assess the relative benefits of alternative design configurations based on both steady-state economics *as well as* ease of dynamic control in an environment of steady-state uncertainties and dynamic disturbances.

Although several operability approaches are implemented, they share a basic approach: that approach consists of calculating the optimal performance attainable for HENs with *different* specified controller structures, in an optimization framework. Thus, for each control method used, a separate optimization is implemented in which the controller parameters for the specific controller method under investigation form the decision variables of the optimization strategy, the objective function is some measure of the dynamic performance the the closed loop system, and the constraints are the operating constraints of the closed loop system, which are typically setpoint specifications and limits on the manipulated variables of the process.

In addition, the comparisons made between the optimal performance measures of alternative HEN designs are discussed with reference to some common open-loop

measures in order to establish why one design performs better than another.

The economics of alternative flexible HEN designs are compared with the respective operability measures for each flexible design in order to show how flexibility may be included in the analysis methods.

In addition, flexibility considerations will also be taken into account, particularly from a design point of view. The flexible design problem is solved through an optimization strategy that seeks to minimize design and operating costs subject to design and operating constraints, as well as the satisfaction of these constraints under operating conditions that may arise due to slow varying uncertainties or fast acting disturbances. Once the design variables that facilitate the flexibility of the design are found, the achievable dynamic performance of the system can then be ascertained though the methods discussed above.

More specifically, the dynamic operability of different network configurations is assessed by calculating an operability performance measure for each of the following control strategies:

1. An optimally tuned PI controller (feedback);
2. An optimally tuned Model Predictive controller (MPC,feedback);
3. An optimal linear controller (feedback), and;
4. The optimal manipulated input trajectory able to reject the specified disturbance (open-loop).

By comparing the optimal performance of each of these controllers, insight can be gained into the operability characteristics of the process itself. The PI controller is an industrial standard, and may be used as 'benchmark' for the other methods; Model Predictive Control (MPC) is becoming quite popular in industrial application, and is thus also important as a reference for the more advanced methods.

MPC is a multivariable control strategy, whereas the PI controller implemented in this work is a multi-loop controller. The difference in operability measures for PI and MPC can then be an estimation of the degree of interaction in the plant, and to what extent this interaction may be overcome using a multivariable controller.

The optimal linear controller found via Q-parametrization represents the best possible linear feedback controller. The difference in performance between this controller and the performance of the optimal input trajectory method represents the degree to which non-linear control methods may improve on the control of the process.

Finally, if each of these control methods predicts a consistent ranking for a range of alternative HEN designs, it implies that it may be possible to use just one of them as a measure of operability for different designs, since it would imply that other methods of control would show the same trends for the different designs.

This would be an important result as it would mean that Q-parametrization would be a reliable measure of closed-loop achievable performance via feedback, and its use as a measure of dynamic operability across alternative designs would accurately predict the improvement/decline in dynamic operability achievable with other controller structures of lower order (such as PI control). This will be an observed result rather than one found by rigorous mathematical proof.

4.2 Flexibility program

The flexibility problem investigated in this study may be specified in the mathematical form

$$\begin{aligned} \min_{d,z^N,\theta^N} \quad & C \\ \text{s.t. } h(d,z^N,\theta^N) &= 0 \\ g(d,z^N,\theta^N) &\leq 0 \end{aligned} \tag{4.1}$$

$$\forall\, \theta\,, \exists z \text{ s.t. } \begin{aligned} h(d,z,\theta) &= 0 \\ g(d,z,\theta) &\leq 0 \end{aligned}$$

$$\text{where } \theta \in \left\{\theta^L \leq \theta^N \leq \theta^U\right\}$$

$$\text{where } \begin{cases} \theta &= \text{uncertain parameters} \\ d &= \text{design variables} \\ z &= \text{control variables} \\ C &= \text{design and operating cost} \\ h &= \text{equality constraints} \\ g &= \text{inequality constraints} \end{cases}$$

In words, the problem can be explained as finding HEX areas, utility exchanger areas and utility heat capacity flowrates for which all operating constraints are satisfied at the nominal point as well as each uncertain vertex. The uncertain vertices are the corner points of the hyperrectangle space created by the defined ranges of θ. For example, if there were two uncertain parameters θ_1 and θ_2, then the vertices of this space are shown in Figure 4.1.

There is only one set of design variables, but the control variables may be manipulated between the nominal point and each uncertainty vertex independently.

4.3 Optimal PI Control

PI control is very well known in both academic and industrial applications, and is the most commonly used form of control for chemical process applications. This method is included here to give a frame of reference for the other control methods studied.

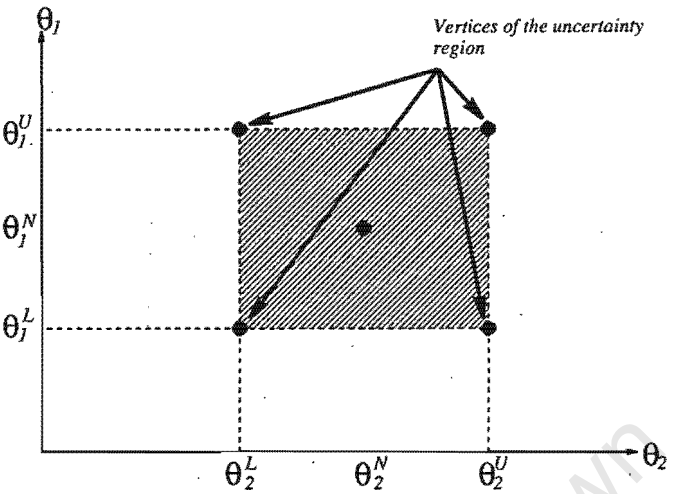

Figure 4.1: Illustration of the uncertainty region resulting from the specification of uncertain parameters in a flexibility design optimization.

An optimally-tuned PI controller is found through an optimization strategy in which the gains and integral time constants are the decision variables. The optimal measure of dynamic performance found through this optimization strategy represents the optimal performance in a closed-loop system for the HEN design studied under optimal PI control.

The mathematical description may be stated[1] as

$$\min_{K_c,\tau_I} \int_0^\infty (y - y_{set})\, W\, (y - y_{set})^T\, dt \tag{4.2}$$

$$\begin{aligned} y &= GC\,(I + GC)^{-1}\, y_{set} + G_d\,(I + GC)^{-1}\, d \\ u &= C\,(I + GC)^{-1}\, y_{set} + G^{-1} G_d\,(I + GC)^{-1}\, d \end{aligned}$$

where K_c refers to the vector of PI controller gains and τ_I refers to the vector of PI controller integral time constants. These are used to build the controller C.

[1]For the calculation of the formula for u: The formula for y in the closed-loop follows from the structure of Figure 2.1, and since $y = Gu$, the substitution gives the result for u shown in the optimization formulation.

The calculation of y and u is based on the classical feedback framework described in Figure 2.1. A simple input saturation filter may be adopted within the simulation strategy to prevent data overflow for unstable controllers in the following way:

$$u^* = \begin{cases} u^U & u \geq u^U \\ u^L & u \leq u^L \\ u & u^L \leq u \leq u^U \end{cases}$$

However, using this method of input saturation means that the control algorithm has effectively become nonlinear. In the analyses presented in this work, the above input saturation mechanism was implemented simply to prevent a computational data overflow for regions in which the PI control algorithm became unstable. It may be seen by the solution trajectories of the examples presented later that the input action never became saturated for the disturbances specified.

The difficulty in solving the above formulation appears because the decision variables K_c and τ_I are not linear in the formulation. This is because of the inversion necessary to create the closed-loop transfer function matrix. For the optimization of PI controllers, the velocity form of the digital PI controller was implemented. This option is attractive because input saturation is easy to implement, and the integral term may be suspended on input saturation which prevents reset windup.

The velocity form of the digital PI controller may be described by

$$\Delta p_{n,i} = K_{c,i}\left[(e_{n,i} - e_{n-1,i}) + \frac{\Delta t}{\tau_{I,i}} e_{n,i}\right]$$

where $\Delta p_{n,i}$ is the change in the manipulated input i from the previous timestep to the current one, $e_{n,i}$ is the setpoint error for input i for the current time-step, and $e_{n-1,i}$ is the error from the previous timestep for input i. The vector of all input moves Δp_n in a MIMO problem is found by calculating the above equation for each input.

The reader is referred to §5.3.1 for a description of how the digital PI controller is implemented in the software package developed for this work.

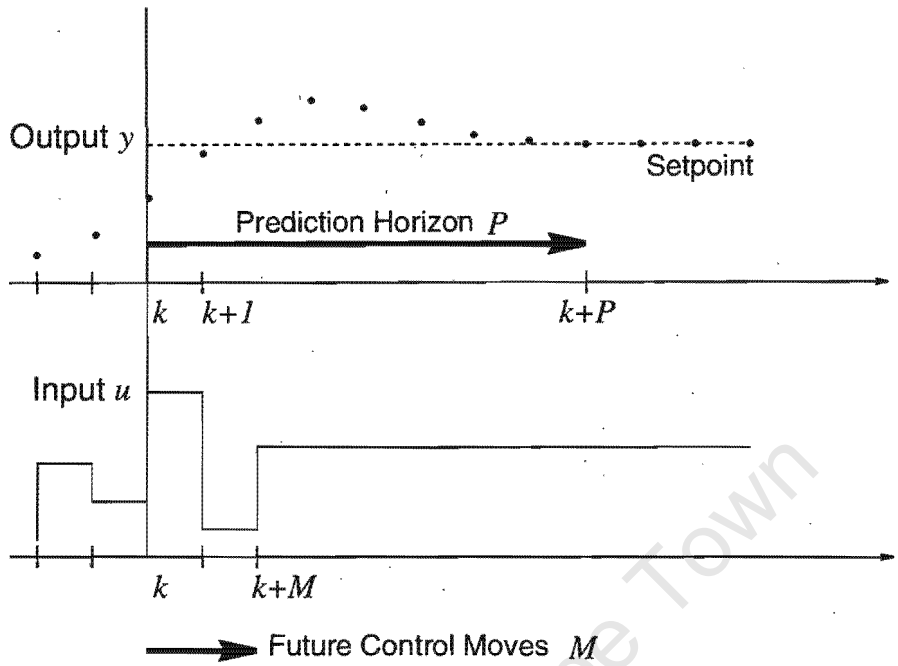

Figure 4.2: Description of variables used in the MPC formulation

4.4 Optimal model predictive control (MPC)

4.4.1 Theory behind MPC

The material presented in this section is based primarily on the lecture notes for the Advanced Process Control (CHE519Z) course offered at the University of Cape Town in 1998. The reader is referred to Cutler and Ramaker (1979) for more details of the algorithm, and Garcia et al. (1989) and Morari and Lee (1997) for more details about the industrial application of the method.

The particular implementation of MPC presented here corresponds to Dynamic Matrix Control (DMC). The variables used in the MPC formulation are described in Figure 4.2.

Once step response coefficients that describe (or approximate) the process model have been identified, the output response for an arbitrary input can be written in

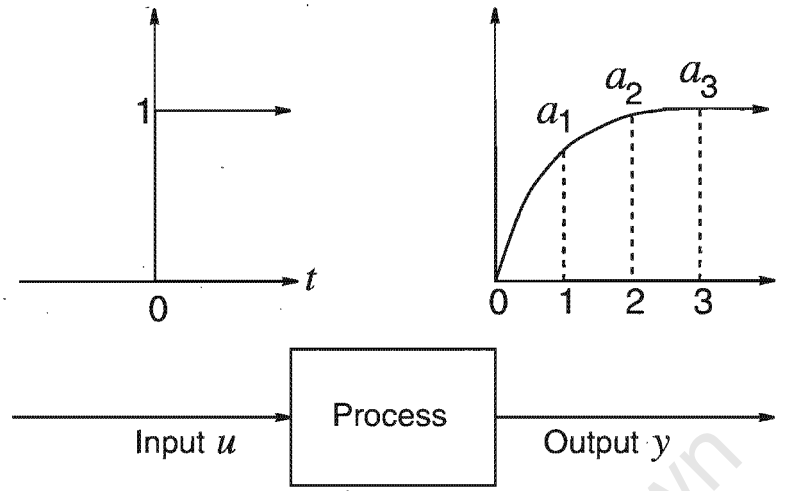

Figure 4.3: Model identification via a step response model. Based on a step input, the value of the output is recorded at discrete timesteps. These y-values form the step-response model.

terms of those step response coefficients as

$$y(k) = \sum_{i=1}^{\infty} a_i \Delta u(k-i) + d(k) \tag{4.3}$$

where a_i represents the output values at timestep i resulting from input changes Δu; these variables are shown in Figure 4.3 for a unit step change in the input ($\Delta u = 1$ initially and then zero for the remaining time steps). The summation may be shortened to N which is the time step at which the system reaches steady state, as in

$$y(k) = \sum_{i=1}^{N-1} a_i \Delta u(k-i) + a_N u(k-N) + d(k) \tag{4.4}$$

In addition, future outputs may be predicted based on the step response coefficients where, if the predicted outputs are $\hat{y}(k+1), \ldots, \hat{y}(k+P)$ and future inputs are given by $u(k), \ldots, u(k+M)$, then the full prediction horizon may be generated by

$$\begin{array}{ll} \hat{y}(k+1) = a_1 \Delta u(k) & \lceil +a_2 \Delta u(k-1) + a_3 \Delta u(k-2) + \cdots \rceil + d(k+1) \\ \hat{y}(k+2) = a_1 \Delta u(k+1) + a_2 \Delta u(k) & \underbrace{\lfloor +a_3 \Delta u(k-1) + a_4 \Delta u(k-2) + \cdots \rfloor}_{\text{effect of past inputs}=y^*} + d(k+2) \end{array} \tag{4.5}$$

In general, this may be represented by

$$\hat{y}(k+l) = \sum_{i=1}^{l} a_i \Delta u(k+l-i) + y^*(k+l) + d(k+l) \tag{4.6}$$

where $y^*(k+l)$ involves only past inputs.

In full matrix form, this relationship may be described as

$$\begin{bmatrix} \hat{y}(k+1) \\ \hat{y}(k+2) \\ \vdots \\ \hat{y}(k+P) \end{bmatrix} = \begin{bmatrix} a_1 & 0 & 0 & \cdots & 0 \\ a_2 & a_1 & 0 & & 0 \\ a_3 & a_2 & a_1 & & 0 \\ & & & & \vdots \\ \vdots & \vdots & & & a_1 \\ & & & & \vdots \\ a_P & a_{P-1} & a_{P-2} & \cdots & a_{P-M+1} \end{bmatrix} \begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \Delta u(k+2) \\ \vdots \\ \Delta u(k+M-1) \end{bmatrix} + \begin{bmatrix} y^*(k+1) \\ y^*(k+2) \\ \vdots \\ y^*(k+P) \end{bmatrix} + \begin{bmatrix} d(k+1) \\ d(k+2) \\ \vdots \\ d(k+P) \end{bmatrix}$$

where P is the prediction horizon and it is assumed that $\Delta u(k+M+i) = 0$ for $i = 0, 1, \ldots$. This may be further simplified notationally by

$$y = A\Delta u + y^* + d \tag{4.7}$$

An optimization strategy may then be set up in the following formulation:

$$\min_{\Delta u} \quad \big(y_{set} - y\big)^T Q \big(y_{set} - y\big) + \Delta u^T \Lambda \Delta u \tag{4.8}$$

$$\text{Subject to: } y = A\Delta u + b.$$

where $b = y^* + d$.

The solution to optimization equation 4.8 may be calculated as

$$\Delta u = \Big(A^T Q A + \Lambda\Big)^{-1} A^T Q \big(y_{set} - b\big).$$

Note that P, M and Λ are user-specified and may be used for tuning of the control action.

Implementation of the control algorithm under feedback

The implementation of the DMC control algorithm may be described by the following iterative steps:

1. The output measurement $y(k)$ is obtained by plant measurement ;

2. The disturbance is estimated via

$$\begin{aligned} d(k) &= y(k) - y^*(k) \\ d(k+i) &= d(k) \qquad i = 1, \ldots, P \end{aligned} \tag{4.9}$$

3. The vector b is updated;

4. The control move vector

$$\Delta u = \left(A^T Q A + \Lambda\right)^{-1} A^T Q \left(y_{set} - b\right)$$

is calculated and the *first* control move is implemented;

5. Time increment k is updated and the sequence returns to step 1.

Extension to MIMO systems

The extension of the model predictive control strategy to MIMO systems follows naturally from the SISO formulation using intuitive matrix ordering methods[2]. The output and input vectors stack the time trajectories above each other for each output

[2]This is one of the reasons why the MPC implementation works so well in MATLAB© .

and input respectively.

$$y = \begin{bmatrix} \hat{y}_1(k+1) \\ \vdots \\ \hat{y}_1(k+P) \\ \vdots \\ \hat{y}_r(k+1) \\ \vdots \\ \hat{y}_r(k+P) \end{bmatrix} \qquad \Delta u = \begin{bmatrix} \Delta u_1(k) \\ \vdots \\ \Delta u_1(k+M-1) \\ \vdots \\ \Delta u_s(k) \\ \vdots \\ \Delta u_s(k+M-1) \end{bmatrix} \tag{4.10}$$

where r and s represent the number of outputs and inputs respectively.

The predictions equation of 4.7 then become

$$y = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1s} \\ A_{21} & & & \\ \vdots & & & \\ A_{r1} & \cdots & & A_{rs} \end{bmatrix} \Delta u + b \tag{4.11}$$

where A_{ij} is the dynamic matrix corresponding to output i and input j.

The optimization problem of 4.8 then becomes

$$\min_{\Delta u} \; (y_{set} - y)^T Q (y_{set} - y) + \Delta u^T \Lambda \Delta u \tag{4.12}$$

$$\text{Subject to: } y = A\Delta u + b$$

$$\text{where} \quad \begin{aligned} Q &= \text{diag}(q_1 \cdots q_1 \quad q_2 \cdots q_2 \quad \cdots \quad q_r \cdots q_r) \\ \Lambda &= \text{diag}(\lambda_1 \cdots \lambda_1 \quad \lambda_2 \cdots \lambda_2 \quad \cdots \quad \lambda_s \cdots \lambda_s) \end{aligned}$$

The solution to this MIMO MPC optimization is completely analogous to the solution of the SISO case:

$$\Delta u = \left(A^T Q A + \Lambda \right)^{-1} A^T Q (y_{set} - y).$$

Different outputs can be weighted through the specification of Q, or different input moves may be suppressed through the specification of Λ. These may be used as tuning parameters for the MPC algorithm.

Handling of input and output constraints

Bounds on inputs and outputs may be incorporated into MPC optimizations as additional constraints, as in:

$$\min_{\Delta u} (y_{set} - y)^T Q (y_{set} - y) + \Delta u^T \Lambda \Delta u \tag{4.13}$$

subject to

$$\underbrace{y = A\Delta u + b}_{\text{Satisfaction of plant step response model}}$$

$$\underbrace{\Delta u^L \leq \Delta u \leq \Delta u^U}_{\text{Constraints on rate of input changes}}$$

$$\underbrace{u^U \leq u \leq u^L}_{\text{Manipulated input bounds}}$$

$$\underbrace{y^U \leq y \leq y^L}_{\text{Output bounds}}$$

This formulation constitutes a quadratic program which is readily solved using standard software. The formulation corresponds to Quadratic Dynamic Matrix Control (QDMC), and has not been used in this work, i.e. bounds on manipulated variables have been enforced by saturation methods (as for the PI control case) and *not* as part of DMC optimization strategy.

4.4.2 Optimal MPC with Λ as optimization decision variables

Since Λ values may be used to tune the MPC controller, it is possible to set up an optimization framework in which setpoint error is minimized over a disturbance-induced output trajectory, with Λ values as decision variables in the optimization program. This may expressed mathematically in formulation 4.14.

$$\min_{\Lambda} \int_0^{\infty} (y - y_{set})^T W (y - y_{set})\, dt \tag{4.14}$$

subject to

$$f(y, u, t, A, \Lambda) = 0$$

where f refers to the MPC feedback control algorithm handled by the MATLAB© toolbox MPCtools© and described on page 52. The input and output trajectories, as well as the tuning parameters Λ are provided to the upper level optimization (in which Λ are the decision variables) by the lower level MPC optimization (in which the Λ are fixed and the Δu values are the decision variables. Bounds on manipulated inputs are enforced by saturation within MPCtools© , and not as constraints within the DMC optimization performed within the MPCtools© toolbox.

The MPC prediction horizon P and the input move horizon M may also be used as tuning parameters, but introducing these parameters into the optimization procedure above as decision variables would make the above formulation a mixed-integer problem. It was therefore decided to set these values high enough so that the MPC method would not be restricted because of short horizons; It was found that setting each to 40 was sufficient[3].

4.5 Implementation of a Q-parametrized linear controller

The underlying theory behind Q-parametrization was explained in § 2.2.2. Here we discuss the implementation of this control strategy.

With reference to Figure 2.3, recall that P may be partitioned

$$P = \begin{bmatrix} P_{zw} & P_{zu} \\ P_{yw} & P_{yu} \end{bmatrix}.$$

[3]More specifically, it was observed for the systems studied later in this document, that the sensitivity of the P and M settings to the performance of the MPC system was very low, and only became significant at horizon settings around 5 and less, where performance decrease became significant.

and the outputs z and y are then related to the inputs by

$$\begin{bmatrix} z \\ y \end{bmatrix} = P \begin{bmatrix} w \\ u \end{bmatrix}$$

The closed-loop map H_{zw} is readily determined from the partitions of P as

$$H_{zw} = P_{zw} - P_{zu}K(I + P_{yu}K)^{-1}P_{yw}$$

Since

$$\begin{aligned} y &= G_d d + G_p u \\ u &= u \\ -e &= -r + G_d d + G_p u \end{aligned}$$

This relationship may then be expressed in matrix form as

$$\begin{bmatrix} y' \\ u \\ -e \end{bmatrix} = \begin{bmatrix} \mathbf{0} & G_d & G_p \\ \mathbf{0} & \mathbf{0} & I \\ -I & G_d & G_p \end{bmatrix} \begin{bmatrix} r \\ d \\ u \end{bmatrix} \tag{4.15}$$

where

$$P = \begin{bmatrix} \mathbf{0} & G_d & G_p \\ \mathbf{0} & \mathbf{0} & I \\ -I & G_d & G_p \end{bmatrix}$$

And the stable coprime factorizations[4] of P are

$$T_1 = \begin{bmatrix} 0 & G_d \\ 0 & 0 \end{bmatrix}, \quad T_2 = \begin{bmatrix} G_p \\ I \end{bmatrix}, \quad T_3 = \begin{bmatrix} -I & G_d \end{bmatrix}$$

such that the parametrized closed loop transfer function becomes

$$H_{zw}(Q) = T_1 + T_2 Q T_3 \tag{4.16}$$

[4]A procedure for determining the stable coprime factorizations of a transfer function is presented by Doyle et al. (1992)

and is expressed as a function of Q.

Now, in order to find the parametrization that minimizes the ISE, the following mathematical optimization formulation is proposed:

The mathematical description may be stated as

$$\min_{Q} \int_0^{\infty} (y - y_{set})\, W\, (y - y_{set})^T\, dt \tag{4.17}$$

subject to

$$y(\infty) = y_{set}$$

where the output trajectory y and input trajectory u are calculated within the optimization program by simulating

$$\begin{bmatrix} y \\ u \end{bmatrix} = H_{zw}(Q) \begin{bmatrix} y_{set} \\ d \end{bmatrix}$$

where H_{zw} is calculated as a function of Q, the decision variables in the optimization routine.

4.6 Implementation of the optimal input trajectory optimization

The general theory and formulation behind the optimal input trajectory optimization method was given in equation 2.30 on page 24. Here, the formulation is geared more specifically for the optimization of the input trajectories of heat exchanger networks.

$$\min_{u} \int_0^{\infty} (y - y_{set})\, W\, (y - y_{set})^T\, dt \tag{4.18}$$

$$y(\infty) = y_{set}$$

where the output trajectory y is calculated by simulating

$$y = Gu + G_d d$$

within the optimization procedure, where G and G_d are as described in Figure 2.1.

Chapter 5

Development of a Software Package for Operability Analysis of Heat Exchanger Networks

The studies presented in this dissertation involve the application of specialized computational tools that were not available from another source. Consequently, these tools were hand coded in the MATLAB© development environment, and structured so that they may be used in any future operability studies. Described here is the functionality of the software, and a description of how the operability analysis methods presented in Chapter 4 have been set up for efficient numerical computation. In addition, various utility functions that are applicable for use in a wide variety of control or operability studies are also mentioned.

5.1 Software functionality

The MATLAB© code in Appendix F[1] is a multi-functional framework for operability analysis applied to HENs. The software generated for this project is able to perform

[1] p. 179

the following functions:

- Optimization for finding an optimal multi-loop PI controller;
- Optimization for finding an optimal MPC controller;
- Optimization for finding an optimal linear controller;
- Optimization for finding an optimal input trajectory;
- Optimization for finding a flexible HEN design, and;
- Optimization for fitting a linear transfer function to the dynamic behaviour of a SIMULINK© modeled HEN.

In addition, SIMULINK© templates are provided to make the construction of user-defined HEN configurations easier.

Of prime importance the creation of this software was modularity—the different functions that may be performed have been separated in different files as much as possible. This maximizes the re-usability of the code, which is an important factor in enabling further research to move beyond what is covered here. In other words, many computational issues will not have to be repeated.

Instead of setting up a graphical user interface, for the software, the approach that has been followed is better described by the term 'script processing'. All required settings such as all relevant data, settings, simulation parameters and optimization algorithm parameters are set up in an input file. Each different function of the software package is enabled through a command line setting, such as

```
>> Results = HENdynopt('inputfile',method);
```

The different methods that may be specified are listed in Table 5.1. The output variable, `results`, is actually a structure variable containing all results and settings

Table 5.1: List of different computational techniques that may be accessed.

Function	`method`	Note
Optimal PI	0	Applicable to any transfer function matrix
Optimal MPC	1	Applicable to any transfer function matrix
Optimal Q	2	Applicable to any transfer function matrix
OOLIT	3	Applicable to any transfer function matrix
SIMULINK© model simulation	4	Specifically for HENs
Steady-state model simulation	5	Specifically for HENs
Model identification	6	Applicable to any input-output trajectory set
Flexibility optimization	7	Specifically for HENs

pertinent to the particular method invoked. Using this strategy, the software package is completely self contained, but can still easily be incorporated into a larger program with little additional development overhead.

Additionally, Table 5.1 describes the range of applicability for each of the functions listed. It should be noted that the methods that are specific to HENs may be altered quite easily so as to allow for different processes.

The following sections describe the significant aspects of the implementation of the different computational methods.

5.2 Flexibility optimization

5.2.1 Steady-state HEN simulation in the flexible design optimization

The flexible design optimization contains within itself a steady-state simulation of a heat exchanger network. The user specifies the HEN configuration being studied in a MATLAB© function file according to a specified criterion.

Using Figure 5.1 as an example, the MATLAB© function that describes this network configuration is listed in Figure 5.2.

The procedure that must be followed in order to ensure that the function gives the correct output temperatures is the following:

1. Identify in the network the heat exchangers that have both input streams

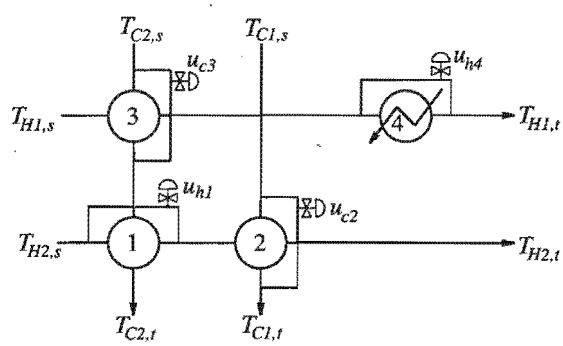

Figure 5.1: Process flow diagram of a HEN for which the steady-state solution may be found using the MATLAB© code in Figure 5.2.


```
function y = calcymarsex2d1(u,theta,thetad);

% This function calculates the output temperatures y of
% A HEN given manipulated bypass fractions u, area
% uncertainties theta and asymptotic input disturbance
% values thetad

h = 0.4; % Heat transfer coefficient (kW/K)
% HEX 3
[T1,T3] = calcHEX(9,8+thetad(2),h+theta(3),u(3),170,60,0,u(8));
% HEX 1
[T2,Tc2out] = calcHEX(10,6,h+theta(1),u(1),250+thetad(1),115,0,u(6));
% HEX 2
[Th2out,Tc1out] = calcHEX(10,8+thetad(2),h+theta(2),u(2),T2,T3,u(7),0);
% CX 1
[Th1out,Tcoolout] = calcHEX(9,u(5),h+theta(4),u(4),T1,25,u(9),0);
% Each HEX has been calculated and answers ready for output:
y(1) = Th1out;
y(2) = Th2out;
y(3) = Tc1out;
y(4) = Tc2out;
y(5) = Tcoolout;
```

Figure 5.2: MATLAB© code used to find steady-state output temperatures of the HEN in Figure 5.1

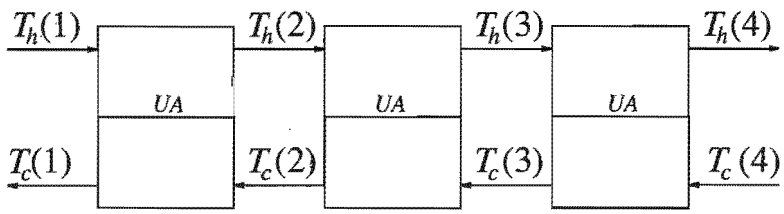

Figure 5.3: A heat exchanger modeled by three well-mixed cells using the arithmetic mean temperature difference driving force

specified, first;

2. The `calcHEX.m` function solves those heat exchangers individually and the output temperatures for each heat exchanger should are returned as variables ;

3. The output temperatures from point 2 are then used as *inputs* to the other heat exchangers in the network that require these as inputs.

In this way, by building up the HEN from the outside in, the complete network may be calculated.

5.2.2 Solving the steady-state heat exchanger

The steady-state heat exchange equations of Equation 3.9 are used, and it is clear that the number of equations defining the heat exchanger becomes larger as the number of well-mixed cells used to approximate the exchanger increases.

Referring to Figure 5.3 which describes an example heat exchanger modeled by 3 well mixed cells and using the arithmetic mean temperature difference for heat transfer driving force, the following working shows how a linear equation system may be set up to solve the steady-state heat exchanger.

Equations 5.1 define the steady-state energy balances for the set of well mixed

cells[2].

$$\begin{aligned}
\text{HXCELL1:} \quad 0 &= \dot{C}_{p,h}T_h(1) + (-\dot{C}_{p,h} - UA)T_h(2) + UAT_c(1) \qquad (5.1)\\
\text{HXCELL2:} \quad 0 &= \dot{C}_{p,h}T_h(2) + (-\dot{C}_{p,h} - UA)T_h(3) + UAT_c(2)\\
\text{HXCELL3:} \quad 0 &= \dot{C}_{p,h}T_h(3) + (-\dot{C}_{p,h} - UA)T_h(4) + UAT_c(3)\\
\text{HXCELL1:} \quad 0 &= \dot{C}_{p,c}T_c(2) + (-\dot{C}_{p,c} - UA)T_c(1) + UAT_h(2)\\
\text{HXCELL2:} \quad 0 &= \dot{C}_{p,c}T_c(3) + (-\dot{C}_{p,c} - UA)T_c(2) + UAT_h(3)\\
\text{HXCELL3:} \quad 0 &= \dot{C}_{p,c}T_c(4) + (-\dot{C}_{p,c} - UA)T_c(3) + UAT_h(4)
\end{aligned}$$

We have 8 variables (4 hot-side temperatures and 4 cold-side variables) and only 6 equations thus far. The other two equations are simply

$$T_h(1) = T_{h,\text{in}} \qquad T_c(4) = T_{c,\text{in}} \qquad (5.2)$$

where $T_{h,\text{in}}$ and $T_{c,\text{in}}$ are specified.

These equations may now be put into the form of a linear equation $A\mathbf{x} = \mathbf{b}$ where

$$A = \begin{bmatrix}
\dot{C}_{p,h} & -\dot{C}_{p,h} - UA & 0 & 0 & UA & 0 & 0 & 0\\
0 & \dot{C}_{p,h} & -\dot{C}_{p,h} - UA & 0 & 0 & UA & 0 & 0\\
0 & 0 & \dot{C}_{p,h} & -\dot{C}_{p,h} - UA & 0 & 0 & UA & 0\\
0 & UA & 0 & 0 & -\dot{C}_{p,c} - UA & \dot{C}_{p,c} & 0 & 0\\
0 & 0 & UA & 0 & 0 & -\dot{C}_{p,c} - UA & \dot{C}_{p,c} & 0\\
0 & 0 & 0 & UA & 0 & 0 & -\dot{C}_{p,c} - UA & \dot{C}_{p,c}\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \qquad (5.3)$$

[2]The notation $\dot{C}_p$ is used here to denote a *heat capacity flowrate* which is simply the product of mass flowrate $\dot{m}$ and the heat capacity C_p.

and

$$\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ T_{h,\text{in}} \\ T_{c,\text{in}} \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} T_h(1) \\ T_h(2) \\ T_h(3) \\ T_h(4) \\ T_c(1) \\ T_c(2) \\ T_c(3) \\ T_c(4) \end{bmatrix}$$

Solving this linear system for $\mathbf{x}$ gives us the complete temperature profile of the heat exchanger, or at least at points defined by the number of well mixed cells implemented. In the MATLAB© environment, the linear system is solved as easily as

```
x = A\b;
```

For the purposes of flexible design optimization, we are really only interested in $T_h(4)$ and $T_c(1)$ which, when referring to Figure 5.3 are seen to be the exit temperatures of both the hot and cold streams respectively.

The syntax for calling the procedure `calcHEX.m` that solves a heat exchanger is

```
[Th_out,Tc_out] = calcHEX(Cph,Cpc,U,A,Th_in,Tc_in,uh,uc);
```

where all the variables have their usual meaning. The code for this function is listed on page 213.

5.2.3 Structure of the input vector for simulation

The structure of the input vector for the *simulation* functions that are specific to HENs is described in Figure 5.4. This structured arrangement allows the code to be general for any user defined size or configuration of a HEN, simply by prespecifying

certain network settings, such as number of process exchangers, number of cooling utilities, number of uncertain steady-state parameters and others.

Note that while there are volume entries in Figure 5.4, these entries will not be used in the steady-state optimization. They are included here for the following reason.

There are two kinds of simulation used in this project: steady state simulation, and dynamic SIMULINK© simulation. The simulation of both of these methods has been set up so that they both use the same input structure. This ensures consistency among the different computational methods.

5.2.4 Defining an initial guess for flexible design optimization

The structure for the input vector to the SIMULINK© and steady-state heat exchanger network algorithms has already been discussed in §5.2.3. The flexible design optimization naturally uses that structure when performing the steady-state simulations which are need to verify whether operational constraints are satisfied.

However, because the flexibility optimization strategy was required to ensure feasibility of constraints at a number of different operating conditions, the input vector to the *flexibility optimization* must contain enough information to run the steady-state HEN simulation at each operating point required.

For this reason, the initial guess passed to the flexibility optimization and the solution vector are structured as shown in Figure 5.5.

5.2.5 Handling uncertain vertices in a flexibility optimization

The design variables (areas) for HENs are fixed for each uncertainty vertex. However, the manipulated variables (bypasses) are allowed to change for each uncertainty

$$u = \begin{bmatrix} \text{Area of process HEX: 1} \\ \vdots \\ \text{Area of process HEX: } N_{HEX} \\ \hline \text{Area of utility heater: 1} \\ \vdots \\ \text{Area of utility heater: } N_h \\ \hline \text{Area of utility cooler: 1} \\ \vdots \\ \text{Area of utility cooler: } N_c \\ \hline \text{Steam flowrate: 1} \\ \vdots \\ \text{Steam flowrate: } N_h \\ \hline \text{cooling water flowrate: 1} \\ \vdots \\ \text{cooling water flowrate: } N_c \\ \hline \text{Manipulated variable: 1} \\ \vdots \\ \text{Manipulated variable: } N_{mv} \\ \hline \text{Volume of HEX: 1} \\ \vdots \\ \text{Volume of HEX: } N_{HEX} \\ \hline \text{Volume of utility heater: 1} \\ \vdots \\ \text{Volume of utility heater: } N_h \\ \hline \text{Volume of utility cooler: 1} \\ \vdots \\ \text{Volume of utility cooler: } N_c \\ \hline \text{Uncertain parameter: 1} \\ \vdots \\ \text{Uncertain parameter: } N_\theta \\ \hline \text{Dynamic disturbance: 1} \\ \vdots \\ \text{Dynamic disturbance: } N_d \end{bmatrix}$$

Figure 5.4: The structure of the input vector for simulation for SIMULINK© or steady-state HEN models

$$u = \left[\begin{array}{c}
\text{column vector of process exchanger areas} \\ \hline
\text{column vector of heater exchanger areas} \\ \hline
\text{column vector of cooler exchanger areas} \\ \hline
\text{column vector of steam} \\ \text{flowrates for heaters - nominal} \\ \text{operating point} \\ \hline
\text{column vector of cooling water} \\ \text{flowrates for coolers - nominal} \\ \text{operating point} \\ \hline
\text{Manipulated variable set} \\ \text{for nominal operating conditions} \\ \text{and no dynamic disturbances} \\ \hline
\text{Manipulated variable set} \\ \text{for nominal operating conditions} \\ \textit{with} \text{ dynamic disturbances} \\ \hline\hline
\text{column vector of steam} \\ \text{flowrates for heaters - uncertainty} \\ \text{vertex 1} \\ \hline
\text{column vector of cooling water} \\ \text{flowrates for coolers - uncertainty} \\ \text{vertex 1} \\ \hline
\text{Manipulated variable set} \\ \text{for uncertainty vertex 1} \\ \text{and no dynamic disturbances} \\ \hline
\text{Manipulated variable set} \\ \text{for uncertainty vertex 1} \\ \textit{with} \text{ dynamic disturbances} \\ \hline\hline
\text{column vector of steam} \\ \text{flowrates for heaters - uncertainty} \\ \text{vertex 2} \\ \hline
\text{column vector of cooling water} \\ \text{flowrates for coolers - uncertainty} \\ \text{vertex 2} \\ \hline
\text{Manipulated variable set} \\ \text{for uncertainty vertex 2} \\ \text{and no dynamic disturbances} \\ \hline
\text{Manipulated variable set} \\ \text{for uncertainty vertex 2} \\ \textit{with} \text{ dynamic disturbances} \\ \hline\hline
\vdots \\ \hline\hline
\text{column vector of steam} \\ \text{flowrates for heaters - uncertainty} \\ \text{vertex } N_\theta \\ \hline
\text{column vector of cooling water} \\ \text{flowrates for coolers - uncertainty} \\ \text{vertex } N_\theta \\ \hline
\text{Manipulated variable set} \\ \text{for uncertainty vertex } N_\theta \\ \text{and no dynamic disturbances} \\ \hline
\text{Manipulated variable set} \\ \text{for uncertainty vertex } N_\theta \\ \textit{with} \text{ dynamic disturbances}
\end{array}\right]$$

Figure 5.5: The input structure required by the flexible design optimization algorithm. Note that operational information for every uncertainty vertex considered may be included, *and then extracted* in an ordered way.

vertex. In addition, a method for specifying the different uncertain parameter set for each uncertainty vertex is required.

Given a matrix of lower and upper uncertainty limits for n variables

$$\Theta = \begin{bmatrix} -\Delta\theta_1 & \Delta\theta_1 \\ -\Delta\theta_2 & \Delta\theta_2 \\ -\Delta\theta_3 & \Delta\theta_3 \\ \vdots & \\ -\Delta\theta_n & \Delta\theta_n \end{bmatrix},$$

it is desired to create a new matrix F that can describe each one of the vertices of Θ as a row vector in an iterative environment, such that if the inputs to a simulation include

$$u = \left\{ [\cdots \text{ process inputs } \cdots], \theta_1, \theta_2, \ldots, \theta_n \right\}$$

then the matrix F must be generated so that each of its rows must be a unique vertex of the set Θ.

Example

Given

$$\Theta = \begin{bmatrix} -\Delta\theta_1 & \Delta\theta_1 \\ -\Delta\theta_2 & \Delta\theta_2 \\ -\Delta\theta_3 & \Delta\theta_3 \end{bmatrix},$$

Then F must be set up as

$$F = \begin{bmatrix} -\Delta\theta_1 & -\Delta\theta_2 & -\Delta\theta_3 \\ \Delta\theta_1 & -\Delta\theta_2 & -\Delta\theta_3 \\ -\Delta\theta_1 & \Delta\theta_2 & -\Delta\theta_3 \\ \Delta\theta_1 & \Delta\theta_2 & -\Delta\theta_3 \\ -\Delta\theta_1 & -\Delta\theta_2 & \Delta\theta_3 \\ \Delta\theta_1 & -\Delta\theta_2 & \Delta\theta_3 \\ -\Delta\theta_1 & \Delta\theta_2 & \Delta\theta_3 \\ \Delta\theta_1 & \Delta\theta_2 & \Delta\theta_3 \end{bmatrix}$$

Now, each row of F can iteratively be appended to a simulation input vector to be able to simulate the system at any uncertainty vertex according to the row of F.

The MATLAB© code that accomplishes this is listed in Appendix F.3.4. In addition, there is a switch that may be used to activate intelligent structuring. This functionality allows the user to give a fixed parametric uncertainty instead of a range. The problem size for this special case can be greatly reduced, as is shown in the following example.

Example

Given

$$\Theta = \begin{bmatrix} -\Delta\theta_1 & \Delta\theta_1 \\ -\Delta\theta_2 & \Delta\theta_2 \\ \Delta\theta_3 & \Delta\theta_3 \end{bmatrix}$$

where $\Delta\theta_3$ is a fixed value of uncertainty, and not a range. F would normally be set up as

$$F = \left[\begin{array}{ccc} -\Delta\theta_1 & -\Delta\theta_2 & \Delta\theta_3 \\ \Delta\theta_1 & -\Delta\theta_2 & \Delta\theta_3 \\ -\Delta\theta_1 & \Delta\theta_2 & \Delta\theta_3 \\ \Delta\theta_1 & \Delta\theta_2 & \Delta\theta_3 \\ \hline -\Delta\theta_1 & -\Delta\theta_2 & \Delta\theta_3 \\ \Delta\theta_1 & -\Delta\theta_2 & \Delta\theta_3 \\ -\Delta\theta_1 & \Delta\theta_2 & \Delta\theta_3 \\ \Delta\theta_1 & \Delta\theta_2 & \Delta\theta_3 \end{array}\right]$$

but there is redundancy in F, and so the code for removing redundant rows creates instead an F matrix with just the top half, thereby reducing the problem size by nearly[3] half.

5.3 Dynamic operability optimizations

This section describes how the implementation of finding optimal control strategies/tunings was accomplished.

The settings for each control algorithm are set up in the input file, as well as the length and timestep of the input-output trajectories used for calculating the ISE and enforcing bounds such as input constraints and setpoint satisfaction. A typical input file is listed on page 219 where the user-friendly structure of the input file is shown.

The settings for each control strategy are created in named structures, which makes it much easier to follow the internal logic of the software if changes are needed to be made.

Even though there are four control strategies currently available, others are easily added because each of these strategies is selected internally by means of a switch-

[3]Not exactly half, because there will be one additional constraint set for the nominal condition, i.e. where $\Delta\theta = 0$

ing mechanism, similar to the way they are selected from the global command line function shown on page 179. Adding a new switch would allow a different control method to be used without modifying the framework of the code.

This switching mechanism is used in both the objective function calculation as well as the function that specifies constraints. This means that there is only one place in which a different objective function can be specified, or additional constraints specified. The code for the switching mechanism is shown in the objective function code and the constraints function code on pages 189 and 190 respectively.

5.3.1 Optimal PI control: Computational Implementation

As mentioned in §4.3, a digital PI control implementation was used. In particular, the methodology was the following:

1. the output resulting from the previous input move is found via simulation for the length of one timestep, and the final states of the system are saved

2. the setpoint error is calculated

3. the corresponding input move is calculated by the velocity form of the digital PI controller

4. input saturation is checked, and if saturated, the integral summation of the PI controller is suspended; This functionality makes this PI control implementation nonlinear if the solution to the optimization results in a manipulated input trajectory that becomes saturated. However, the intent of enforcing input saturation is to prevent unstable closed-loops from causing computational numerical errors. It was found that none of the PI controller solutions found in this work resulted in saturated input trajectories, and so the PI controller method has remained linear.

5. the outputs at the end of this timestep are found by simulation using the states saved in (1) as the initial states for this timestep segment

6. the algorithm continues with point (2).

A mathematical formulation that corresponds to the computational implementation is given here.

$$\underbrace{\min_{K,\tau} \sum_{i=1}^{N} \sum_{c_c=1}^{N_y} \left(y(i,c_c) - y_{sp}(i,c_c)\right)^2}_{\textbf{Setpoint error minimization objective function}} \tag{5.4}$$

subject to

$$\underbrace{u^L(i,c_c) \leq u(i,c_c) \leq u^U(i,c_c) \qquad \forall i \in \{1,2,\ldots,N\} \quad \forall c_c \in \{1,2,\ldots,N_u\}}_{\textbf{Satisfy manipulated variable constraints } \textit{through enforced saturation in closed-loop simulation}}$$

$$\underbrace{\tau_i(c_c) > 0 \qquad \forall c_c \in \{1,2,\ldots,N_u\}}_{\textbf{Ensure all integral time constants are positive}}$$

In order to generate the input and output trajectories required for the evaluation of the objective function and satisfaction of constraints, the closed-loop plant is simulated for the complete time trajectory using the following steps:

1. The disturbance step d is activated and the output and input trajectories are calculated up to the 1st timestep at which the setpoint error is sampled. The system states are stored;

2. A digital PI controller calculates the manipulated inputs based on the setpoint error as a function of the controller gains K_c and integral time constants τ_I;

3. Manipulated input constraints are tested against the calculated input moves and inputs are saturated if the constraints are violated;

4. The algorithm simulates the system until the next timestep starting with the saved system states from the previous timestep, and using the calculated manipulated inputs.

Symbol	Meaning
K	vector of controller gains
τ	vector of controller integral time constants
N	scalar number of discrete timesteps in simulation trajectory
N_y	scalar number of outputs in transfer function process model
N_u	scalar number of inputs
i	scalar counter for trajectory position
r_c	scalar counter for output number
c_c	scalar counter for input number
N_{SPspec}	setting for number of setpoint satisfaction constraints
N_{SSspec}	setting for number of input settling points

Table 5.2: Explanation of variables used in the optimal control formulations

The explanation of the variables in the above formulation is given in Table 5.2, and these parameters are also used in the other computational formulations to follow.

The MATLAB© code for the digital PI control implementation is listed on page 193. Because this file is separated from the rest of the code, the PI control implementation can be modified without affecting the functionality of the rest of the computational framework.

5.3.2 Optimal MPC: Computational Implementation

The actual simulation of the MPC controller is handled by the MPCtools© toolbox that is used within MATLAB© . However, the function used in this work, and which is listed on page 196 provides a gateway to toolbox functions. Setpoints, manipulated variable timesteps, the prediction and input move horizons as well as all other MPC settings are all passed to this gateway function in a logical way.

The mathematical formulation corresponding to the computational implementation is given below.

$$\underbrace{\min_{\Lambda} \sum_{i=1}^{N} \sum_{c_c=1}^{N_y} \left(y\left(i, c_c\right) - y_{set}\left(i, c_c\right)\right)^2}_{\text{Setpoint error minimization objective function}} \tag{5.5}$$

subject to

$$\underbrace{0 \leq \lambda_i \leq 100 \qquad i \in \{1, 2, \ldots, N_u\}}_{\text{Constrain } \Lambda \text{ to be positive } (> 0) \text{ and reasonable } (< 100)}$$

The y trajectory is created within the optimization framework with the sequence of computational step described on page 52, according to a standard DMC implementation. The DMC formulation corresponds to the unconstrained algorithm, although simple saturation of manipulated variables may be activated.

5.3.3 Optimal linear control via Q-parametrization

The coprime factorizations T_1, T_2 and T_3 mentioned in §2.2.2 that are necessary for Q parametrization are all created in separate function files. This is in keeping with the goal of modularity in the software development. The code for each of these functions is listed on page 199.

In addition, the initial guess input vector is correctly restructured in a separate `makeQ.m` file to create the Q transfer function using discrete transfer function polynomials in z^{-1}.

Once these utility functions have been assembled, the closed-loop transfer function specified in equation 2.27 on page 22 may be created. Since LTI systems can be specified like scalar variables in MATLAB© , the command that builds the closed loop transfer function *actually looks like*

```
Hzw = T1 +T2*Q*T3
```

The function that creates the above closed-loop transfer function matrix, and ultimately simulates the closed loop with this control algorithm is listed on page 198.

5.3.4 Optimal open-loop input trajectory optimization

The computational implementation of finding an input trajectory that minimizes setpoint error subject to a step disturbance is relatively straightforward compared

to the other methods.

The decision variables in this optimization are the manipulated variable values themselves, and so the output trajectory used to calculate the ISE is generated within the optimization strategy by direct simulation.

$$\underbrace{\min_{u} \sum_{i=1}^{N} \sum_{c_c=1}^{N_y} \left(y(i,c_c) - y_{sp}(i,c_c)\right)^2}_{\text{Setpoint error minimization objective function}} \tag{5.6}$$

subject to

$$\underbrace{\begin{aligned} 0 = \quad & y(i,c_c) - y_{sp}(i,c_c) \\ & i \in \{N-1, N-2, \ldots, N-N_{SPspec}\} \forall c_c \in \{1,2,\ldots,N_y\} \end{aligned}}_{\text{Satisfy setpoint objective for } N_{SPspec} \text{ timesteps at end of trajectory}} \tag{5.7}$$

$$\underbrace{\begin{aligned} 0 = \quad & u(i-1,c_c) - u(i,c_c) \\ & i \in \{N-1, N-2, \ldots, N-N_{SSspec}\} \forall r_c \in \{1,2,\ldots,N_u\} \end{aligned}}_{\text{Ensure that } N_{SSspec} \text{ consecutive inputs at end of trajectory are equal (ensure settling)}} \tag{5.8}$$

$$\underbrace{u^L \leq u \leq u^U \qquad \forall i \in \{1,2,\ldots,N\}}_{\text{Satify manipulated variable constraints}} \tag{5.9}$$

$$\underbrace{y = Gu + G_d u}_{\text{Open-loop simulation of the system}} \tag{5.10}$$

The code for generating the output trajectory as a function of the input trajectory is listed on page 201.

5.4 Computational issues for identification

The methodology followed for system identification is more direct than most regression techniques: the polynomials for the numerator and denominator of a linear transfer function are used as the decision variables of an optimization in which the difference between the output from the nonlinear SIMULINK© model and the output generated by the transfer function for the same input trajectory is minimized.

Performing the identification in this way has both advantages and disadvantages. The primary disadvantage is that the optimization procedure may push the polynomial entries into a region where the transfer function becomes unstable. The optimizing program then ceases to operate correctly, since the finite-differencing used to calculate gradient information can return values beyond machine precision.

The advantage is that a good initial guess of the estimated transfer function may be made on the basis of the observed response of the nonlinear system. If inverse responses are observed, for instance, then the initial guess could include a right-half-plane zero.

It has been found that the best way of performing a system identification procedure is an interactive one. Consequently, that is the approach used here. The computational procedure iterates through each input and output pair (assuming a MIMO model). For each input-output pair, the following steps are performed:

1. A graph of the nonlinear SIMULINK© model output is displayed, generated from a random multiple step input centralized around the nominal input value and according to a user defined magnitude and frequency

2. The user is asked to select the number of numerator and denominator terms used in the transfer function model for approximating the given output.

3. The user is asked to give the value of the time-delay that is *estimated from the graph* in (1). Note that in MATLAB© , it is easy to zoom in on displayed graphs, and so the time delay estimate is usually quite accurate.

4. The user is then asked if an initial guess different from the default one should be provided. This is recommended for outputs displaying behaviour typical of systems with right-half plane zeros.

5. The optimization procedure is then activated. Once a solution has been found,

the solution transfer function is display, along with a graph showing the fit between the two trajectories

6. The user is then asked if the approximation is satisfactory. If not, the procedure will be repeated where the user may specify a different order for the estimated transfer function by increasing or decreasing the length of the numerator and denominator polynomials

7. Following a successful fit, the program proceeds to the next input-output pair.

Each time a successful fit is obtained, the solution transfer function is added into a MIMO transfer function matrix in the correct IO position. In this way, it will not be necessary to enter all the data in after the full MIMO identification is complete— the full MIMO transfer function matrix will be completely defined, and ready for other analyses.

The code for system identification is listed on page 226.

5.5 Utility functions

Described here are functions that are more standalone than the other computational methods described, and may consequently be used in a wider variety of applications.

The corresponding code is provided in the Appendices. Given here is a brief description of each.

- *Open-loop indicators.* These open-loop measures are easy to calculate, but a computational implementation was not found from any other source, and these short programs were therefore developed.
  - RGA: page 223;
  - PRGA: page 223;

  - Condition number: page 223;
  - Disturbance condition number: page 224;
  - Morari resiliency index: page 225;
  - Closed-loop disturbance gain: page 224, and;
  - Relative disturbance gain: page 225.

- *Asymptotic input values required for disturbance rejection.* Given transfer function matrices for the process and disturbances and the values of the step disturbances, this function calculates the input set that enforces zero setpoint error at steady-state. In the operability optimizations presented previously, these input values are used for constraints on the final values of the input trajectories to aid convergence. The code is listed on page 216.

Chapter 6

Results and Discussion

A number of studies have been performed to investigate the operability characteristics of HENs, as well as to compare different methods of operability analyses with one another. Flexibility studies are presented first, since flexibility is a necessary condition for dynamic operability. Then, operability studies are presented for a range of examples, but with a different focus in each case. It is found that the comparison presented in this thesis does provide a practical means of operability analysis for HENs, and in addition allows one to infer operability characteristics about the dynamic behaviour of heat exchanger networks in a controller-independent way.

6.1 Dynamic Operability Analyses

Here we will present the implementation and results of a dynamic operability assessment methodology whereby a measure of achievable dynamic performance is calculated for closed-loop HEN systems using control structures of varied complexity.

6.1.1 Illustration of operability analysis for a simple example

In order to test the method of operability analysis proposed in Chapter 4, it was first decided to investigate a simple example. The example used is described by the Laplace domain transfer function

$$G(s) = \frac{2s-1}{s^2+0.5s+1}e^{-2.1s} \tag{6.1}$$

with the disturbance transfer function given by

$$G_d(s) = \frac{0.2}{s+1}. \tag{6.2}$$

A step disturbance of magnitude -0.2 was used and the time horizon and control move time step were set to 100 seconds and 5 seconds respectively. The time step of the output sampling on which the output setpoint error was calculated was 0.1 seconds. The input move horizon and prediction horizon for DMC simulation were each set to 30, and the number of q coefficients used in the Q-parametrization procedure was 10.

This truncation means that the Q performance predicted will not reflect the strict optimal performance for all linear controllers, but the point of truncation used should mean that a large number of linear controllers is still represented by this set.

By including setpoint deviation data points in between control move points, it was possible to penalize inter-sample oscillatory behaviour that may have occurred if the sampling times for the setpoint error calculations had been the same length of time as for the control moves.

The results found can be seen in Table 6.1. The values of the performance measures are consistent in that more advanced control methods generally show better performance which is characterized by the lower setpoint error sum. However, it is strange that the OOLIT method should be so much better that the Q-parametrization method. After an analysis of the trajectories in Figure 6.1, the

Table 6.1: Dynamic performance results of a SISO system

	ISE
OOLIT	2.7253
Q	4.7755
DMC	4.9028
PI	4.9080

reason was clear: the Q method applies corrective input action only when setpoint error in the trajectory is detected, whereas the OOLIT method has 'full view' of the output trajectory and applies input action 'early'. Note that even though a disturbance step occurs at $t = 0$, the process output at this time is still zero. This is why the Q, DMC and PI methods do not apply input action at zero, but rather at $t = 5$ which is when the first time-step after zero the respective control algorithms determine the setpoint error.

It should be noted, however, that if the timestep is reduced from 5 to 1, then all of the Q, OOLIT (constrained at $t = 0$) and the unconstrained OOLIT methods give almost exactly the same result[1] (ISE = 3.973, 4 significant digit precision), for which the input and output trajectories are shown in Figure 6.2. This means that the severity of this effect depends on the sample time for control; it should, however be considered.

While it is true that the OOLIT method represents the best possible performance for any control strategy, it is the opinion of the author that this method should be constrained to behave in a more physically possible way, i.e. by constraining the input action at $t = 0$ to zero. This may seem to restrictive for a method that is supposed to

[1]It is also interesting that a good agreement between Q with 10 coefficients and the constrained OOLIT case was achieved; comparing with Table 6.2, there was still error between the constrained OOLIT case and the Q method for the *same* number of Q coefficients—It would appear that decreasing the timestep also allows lowering the number of Q coefficients for the same accuracy. The effect of the sampling time on the Q-parametrization method should be explored in a future study, but it is not one of the aims of this work.

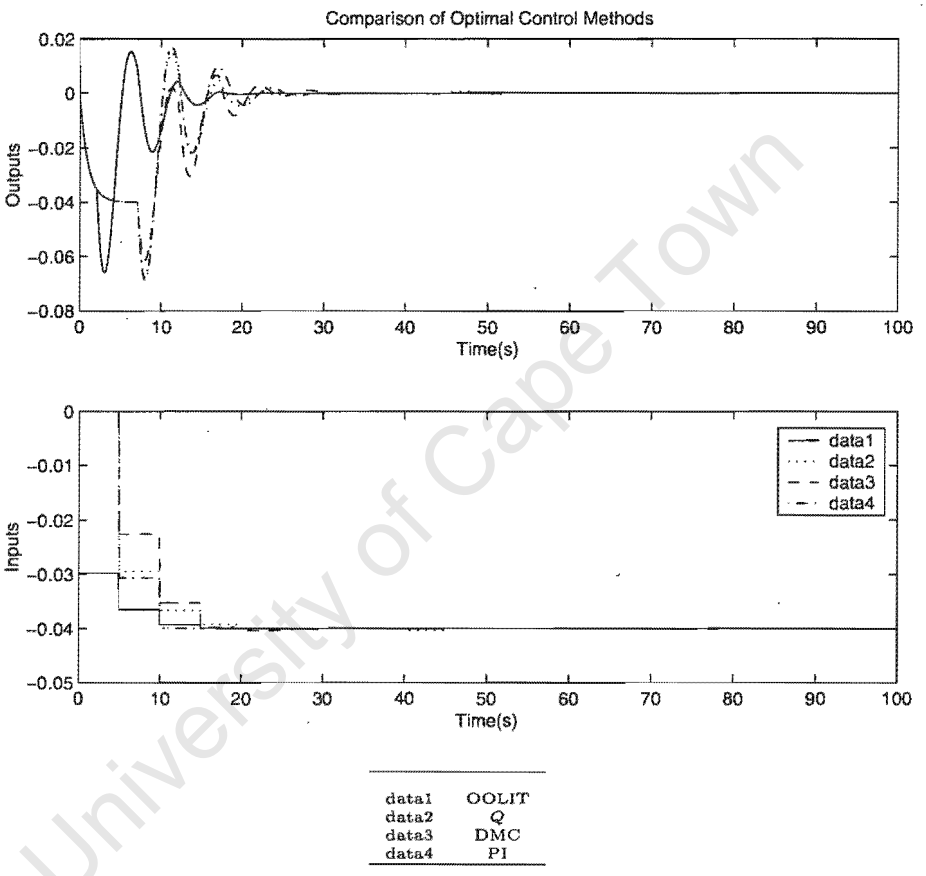

Figure 6.1: Input and output trajectories of closed-loop simulations for a simple SISO system

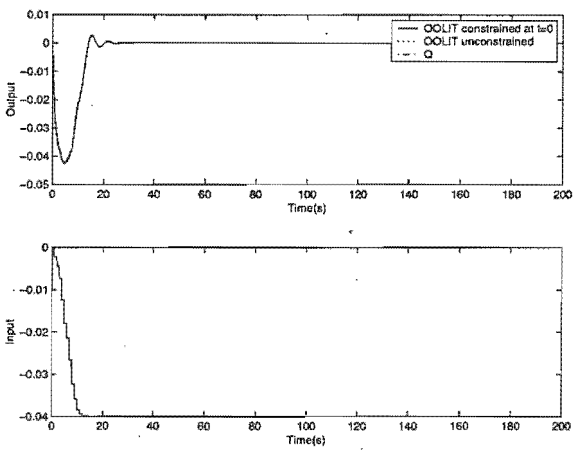

Figure 6.2: Input and output trajectory for Q, OOLIT constrained at $t = 0$ and OOLIT *unconstrained* at $t = 0$. Because the timestep is much smaller, the gains made in not constraining the first OOLIT move are much less significant than for larger timestep values

Table 6.2: Dynamic performance results of a SISO system with the OOLIT method restricted at $t = 0$.

	ISE
OOLIT	4.7216
Q	4.7755
DMC	4.9028
PI	4.9080

yield the best possible dynamic performance of a plant under disturbance action, but this is *not* a restriction on the behaviour of the controller, but rather the possibility of 'seeing the future' which the other methods cannot do.

By implementing this prediction constraint, it is believed that a more meaningful comparison of the various control methods may be made, Clearly, the measures in Table 6.1 are only useful for performance comparisons for Q, DMC and PI, since the huge advantage the OOLIT method has is that it does not have to wait for a timestep to pass before corrective input action can be taken. If the OOLIT method is constrained in its first control move timestep, then the results table becomes more understandable, as shown in Table 6.2.

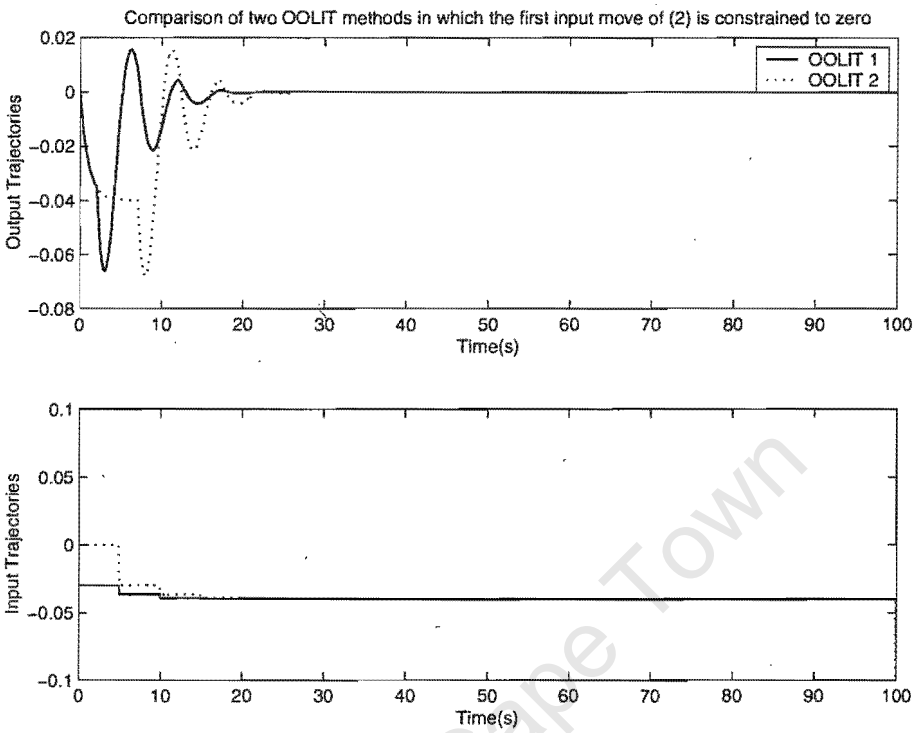

Figure 6.3: Comparison of two OOLIT methods in which one is restricted at $t = 0$ for a simple SISO system

In addition, the output trajectories for both OOLIT methods is shown in Figure 6.3. One can see that the second trajectory correctly applies no input action at t_0 because there is no setpoint offset at this time.

The method for each curve in Figure 6.3 is exactly the same—however, one is constrained so that input action at $t = 0$ is not allowed. The large difference in setpoint error, then, can only be attributed to the input action at the start of the trajectory.

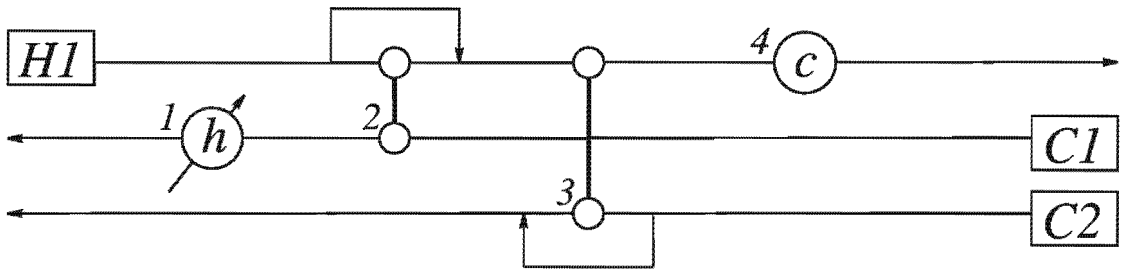

Figure 6.4: Design A for a heat exchanger network example 1.

6.1.2 Dynamic operability analysis example 1: Comparison of achievable limits of performance for alternative HEN designs

The HEN example studied here is based on an example presented by Glemmestad (1997). The complete HEN forms a 3-input-3-output system, and the two alternative design strategies are formed by considering different sets of manipulated variables. Note that from the point of view of controller design, the specific choice of manipulated variables selected for the closed-loop system *forms an integral part of the design itself.*

An operability analysis performed on this system will provide a description of the operability assessment methodology described in this document, as well as show the importance of good loop pairing in control system design for heat exchanger networks.

Description of the HEN specifications

The operating and design specifications for this heat exchanger network example appear in tables 6.3 and 6.4 respectively, where T_s refers to the *source* or feed temperature, T_t refers to the *target* or setpoint temperature and $\dot{C}_p$ refers to the heat capacity flowrate. The process flow diagrams for the two designs appear in Fig-

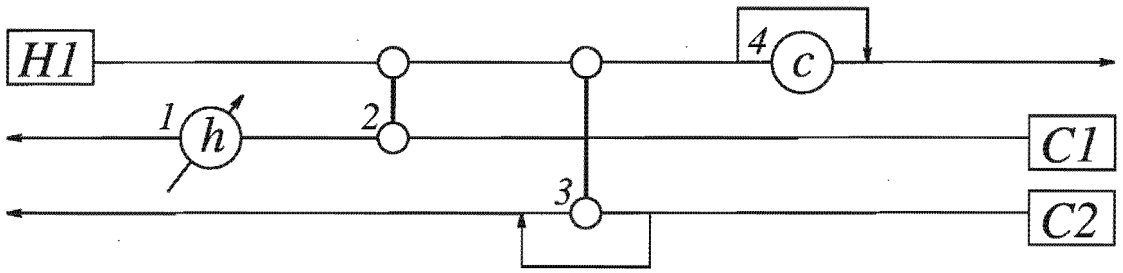

Figure 6.5: Design B for a heat exchanger network example 1.

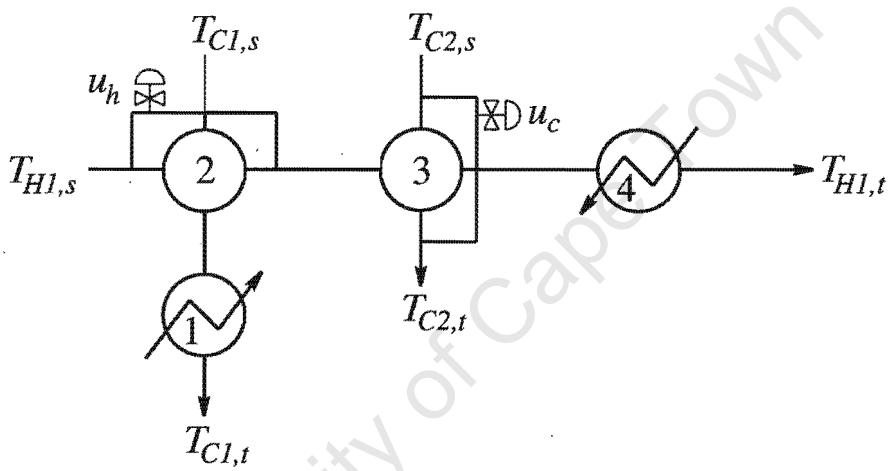

Figure 6.6: Process flow diagram for design A.

ures 6.6 and 6.7.

Linear model identification

The MATLAB© Identification Toolbox was used for finding linearized models for the nonlinear SIMULINK© models constructed to represent models A and B. These models are presented in Appendix E[2] in state-space form and it can be seen that they are of a high order. It was found that the Identification Toolbox could not

[2] p.171

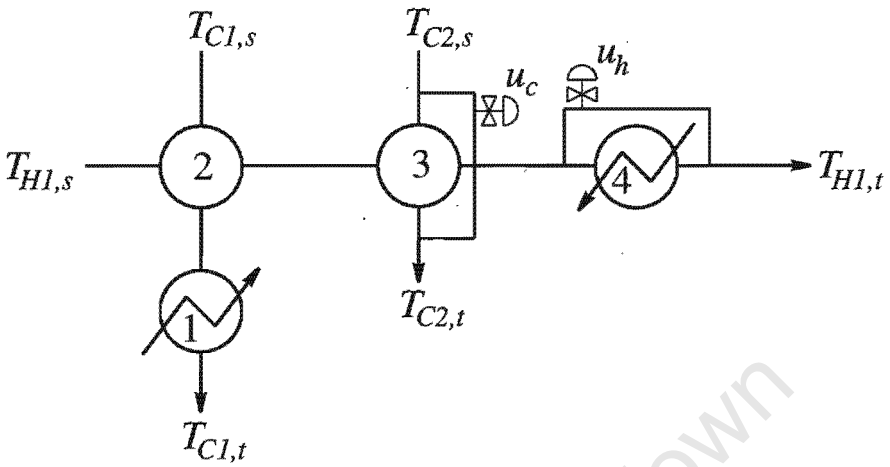

Figure 6.7: Process flow diagram for design B.

Table 6.3: Operating specifications for heat exchanger network example 1.

Stream Number	$T_s(°C)$	$T_t(°C)$	$\dot{C}_p(kW/°C)$
H1	190	30	1.0
C1	80	160	1.5
C2	20	120	0.5

Table 6.4: Nominal design specifications for heat exchanger network example 1.

	UA $(kW/°C)$
HEX 2	0.523
HEX 3	1.322

decrease the order of the linearized models without severely affecting the accuracy of the linear model-nonlinear SIMULINK© model match. In addition, the author found the Identification Toolbox difficult to use.

For these reasons, a different identification strategy was implemented for operability studies 2 and 3.

Results of dynamic operability optimizations

The 1^{st} manipulated variable set consists of

1. A hot bypass on exchanger 1;
2. A Cold bypass on exchanger 2;
3. The steam flowrate on the heater.

and the 2^{nd} manipulated variable set consists of

1. A hot bypass on the cooler;
2. A Cold bypass on exchanger 2;
3. The steam flowrate on the heater.

Intuitively, the choice is trivial from an operability point of view since design B stands a much better chance of being controlled better—there is less chance of interaction when both utility exchanger duties are manipulated variables, since interaction is minimized.

However, an operability study using the methods previously set out will be instructive for a number of reasons:

1. It will be made clear that the choice of input-output pairings strongly influences the behaviour of the closed loop system, and that the same physical design should be seen as a range of different potential designs based on the choice of the manipulated variable set.

2. A quantitative assessment of the relative improvement of selecting a different input-output pairing will be made. This quantitative measure is a direct function of the setpoint error trajectory of the system under a specified disturbance.

3. A study of the dynamic performance predicted by a range of different optimized control methods gives an idea of the overall operability of each system, and shows whether poor dynamic behaviour due to the use of a poor input-output pairing may perhaps be corrected by some extent through the use of more advanced control methods.

An operability analysis performed on this system will provide a description of the operability assessment methodology described in this document, as well as show the importance of good loop pairing in control system design for heat exchanger networks.

Settings

The disturbance set included a $+10°C$ step in H1 and a -0.05 $kW/°C$ step in the heat capacity flowrate of stream C2 with these step disturbances taken to occur simultaneously. The time horizon for simulation was 400 with a sampling time of 5. For multiloop PI control, the input-output pairing as indicated on the process flow diagrams was:

- Design A: $u_{h,unit\,2}$ controls $T_{H1,t}$, u_c controls $T_{C2,t}$ and the steam flowrate on unit 1 controls $T_{C1,t}$;

- Design B: $u_{h,cooler}$ controls $T_{H1,t}$, u_c controls $T_{C2,t}$ and the steam flowrate on unit 1 controls $T_{C1,t}$;

Table 6.5: Dynamic performance measures for two alternative 3×3 heat exchanger network designs.

Control Method	Design A	Design B
OOLIT	52.7520	13.6659
Q_2	60.3525	13.7941
Optimal DMC	60.7810	21.0764
Optimal multi-loop PI	211.9944	66.1184

Table 6.6: *Normalized* dynamic performance measures from Table 6.5.

Control Method	Design A	Design B
OOLIT	3.86	**1**
Q_2	4.42	1.01
Optimal DMC	4.45	1.54
Optimal multi-loop PI	15.51	4.84

The manipulated variable constraints for design A were

$$-0.6670 \leq u_h \leq 0.3330$$

$$-0.1750 \leq u_c \leq 0.8250$$

$$0 \leq \dot{C}_{p,steam}$$

and for design B were

$$-0.1003 \leq u_h \leq 0.8996$$

$$-0.1971 \leq u_c \leq 0.8029$$

$$0 \leq \dot{C}_{p,steam}$$

but these bounds did not become active for any of the solution trajectories in the dynamic operability optimizations.

Results of dynamic operability optimizations

The minimized ISE measures for the given step disturbances appear in Table 6.5. If the values in the above table are normalized by the lowest ISE measure, the

interpretation of the results is made easier, as can be seen in Table 6.6. Let this normalized measure of performance be referred to by the notation φ in the following observations:

1. Design B exhibits the best performance (lowest ISE) for both design alternatives under the disturbance test with the best manipulated input trajectory found for each design ($\varphi_{OOLIT}^{Plant\ B} = 1$);

2. Design B predicts *consistently* better performance for each of the four control methods;

3. To achieve a similar quality of control performance for Design A as that found for optimal PI control in Design B, one would need to use *at least* an optimally tuned DMC control strategy ($\varphi_{DMC}^{Plant\ A} \sim \varphi_{PI}^{Plant\ B}$), and this implies greater cost and control system design complexity;

4. There is a 1% difference in dynamic performance between the Q_2 and OOLIT control strategies for Design B. This suggests not only that using a higher order Q controller would not result in much improvement (this fact can save a great deal of computation time), but more importantly that it is not necessary to explore nonlinear control strategies for improved performance, because there will not be any: the OOLIT method encompasses all nonlinear control techniques. There is a larger difference between the Q_2 and OOLIT measures for design A; this can be attributed to the fact that only 2 coefficients per transfer function in the Q parametrized controller transfer function were used. A larger number is used in dynamic operability examples 2 and 3 where it is seen that the difference between the Q and OOLIT measures decreases, as it should;

5. The dynamic performance obtainable in design B using an optimally tuned DMC controller is better than the absolute best possible performance achiev-

able for design A($\varphi_{OOLIT}^{Plant\ A} > \varphi_{DMC}^{Plant\ B}$). This is the most important result as far comparison between the two designs goes, because it highlights the difference in dynamic performance between the two designs most effectively.

Open-loop measures may be used to explain why one design appears to perform better than another on the basis of the minimized ISE measures presented above. Design A has RHP-zeros

$$\begin{bmatrix} 6.640 + 2.602i \\ 6.640 - 2.602i \\ 1.3757 \\ 2.000 + 0.0001i \\ 2.000 - 0.0001i \\ 2.000 \end{bmatrix}$$

while design B has RHP-zeros

$$\begin{bmatrix} 2.000 \\ 2.000 \end{bmatrix}$$

Design A has a RHP zero closer to the origin than design B. RHP zeros limit the achievable performance of a plant, and this helps to explain why design B performed better than design A.

The RGAs for designs A and B is shown in equations 6.3 and 6.4 respectively.

$$RGA_A = \begin{bmatrix} 3.2813 & -2.2813 & 0 \\ -2.2813 & 3.2813 & 0 \\ 0 & 0 & 1.0000 \end{bmatrix} \tag{6.3}$$

$$RGA_B = \begin{bmatrix} 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 \\ 0 & 0 & 1.0000 \end{bmatrix} \tag{6.4}$$

Design A appears to exhibit more interaction than design B which explains why the latter performs better. When looking at the PRGA for each of these designs,

$$\Gamma_A = \begin{bmatrix} 3.2813 & -2.2069 & -2.0752 \\ -3.3921 & 3.2813 & 2.1452 \\ 0 & 0 & 1.0000 \end{bmatrix} \tag{6.5}$$

$$\Gamma_B = \begin{bmatrix} 1.0000 & 0 & 0 \\ -0.1160 & 1.0000 & 0 \\ 0 & 0 & 1.0000 \end{bmatrix} \tag{6.6}$$

it becomes clear that design A also exhibits severe one-way coupling in column 3. Design B also exhibits this coupling in column 1, but the magnitude is about an order of magnitude less than the input-output paired diagonal entries.

The steady-state condition number for the designs are

$$\gamma_A = 87.9621 \qquad \gamma_B = 2.1211$$

Design A has a significantly larger condition number than design B, and is therefore expected to exhibit poorer controllability. The disturbance condition numbers,

$$\gamma_{d,A} = [2187 \quad 87.70] \qquad \gamma_{d,B} = [45.84 \quad 1.837]$$

indicate that design A is extremely sensitive to the disturbances examined in this study, and therefore should exhibit less controllability than design B for which the values are far smaller. Note that both designs are more sensitive to the disturbance corresponding to temperature change in the hot stream *H1*, than to the flowrate change in cold stream *C2*.

The CLDG for designs A and B is indicated in equations 6.7 and 6.8 respectively.

$$\Delta_{CLDG,A} = \begin{bmatrix} 0.0493 & 46.6 \\ 0.0215 & -503 \\ -0.115 & -224 \end{bmatrix} \tag{6.7}$$

$$\Delta_{CLDG,B} = \begin{bmatrix} 0.0156 & 14.2005 \\ 0.5540 & 0.1469 \\ 0.1217 & 0 \end{bmatrix} \tag{6.8}$$

Dividing (element by element) the closed loop disturbance gains by the steady-state disturbance gains,

$$G_d(0) = \begin{bmatrix} 0.0507 & 14.21 \\ 0.5540 & 0.1469 \\ 0.1217 & 0 \end{bmatrix} \tag{6.9}$$

which is the same for both designs, we can see the relative disturbance gain (RDG):

$$\Delta_{RDG,A} = \begin{bmatrix} 0.9727 & 3.279 \\ 0.03875 & -3426 \\ 0.9455 & n/a \end{bmatrix} \tag{6.10}$$

$$\Delta_{RDG,B} = \begin{bmatrix} 0.3076 & 0.9993 \\ 1.0000 & 1.0000 \\ 1.0000 & n/a \end{bmatrix} \tag{6.11}$$

The open loop disturbance gains in equation 6.9 may appear to be a good indicator of the effect of the disturbances on the outputs, but this is not always the case. The relative disturbance gain for design A indicates that output 2 is affected by disturbance 1 in the closed-loop by $\approx 3.9\%$ of what was predicted in the open loop, while disturbance 2 affects output 2 by a great deal more. The relative disturbance gain for design B, however, appear to indicate that the closed loop disturbance gains are equal to *or less* than was is predicted for the open-loop disturbance gain. The conclusion is that the manner in which the disturbances affect the outputs is very different in the closed loop than in the open-loop, and in this case, design B appears to be able to deal with the disturbances far better than design A.

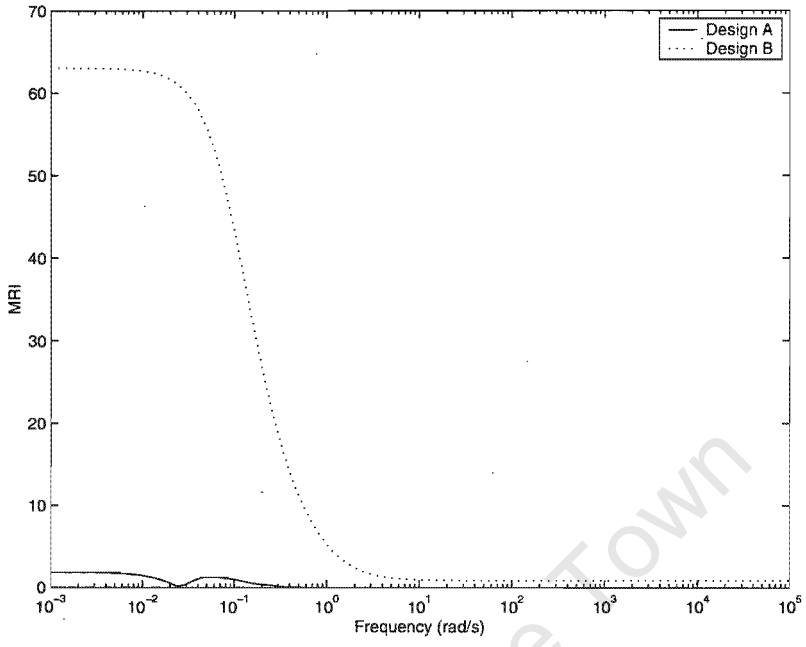

Figure 6.8: The Morari Resiliency Index for designs A and B as a function of frequency.

The MRI is shown in Figure 6.8 and it shows that design B is more resilient than design A by virtue of the fact that is has a larger minimum singular value for a wide portion of the frequency range indicated.

The open-loop indicators appear to be in good agreement obtained from the dynamic operability assessment optimization methods, and provide an understanding of why design B performs better under the given disturbance set than design A.

6.1.3 Dynamic operability analysis example 2: Comparison of achievable limits of performance for alternative *flexible* HEN designs

The previous example demonstrated how two designs may be compared on the basis of different operability assessment techniques, as well as through the use of various open-loop indicators. In this example, a flexible design that is economically optimal and can operate feasibly for both a range of uncertainty in design parameters as well

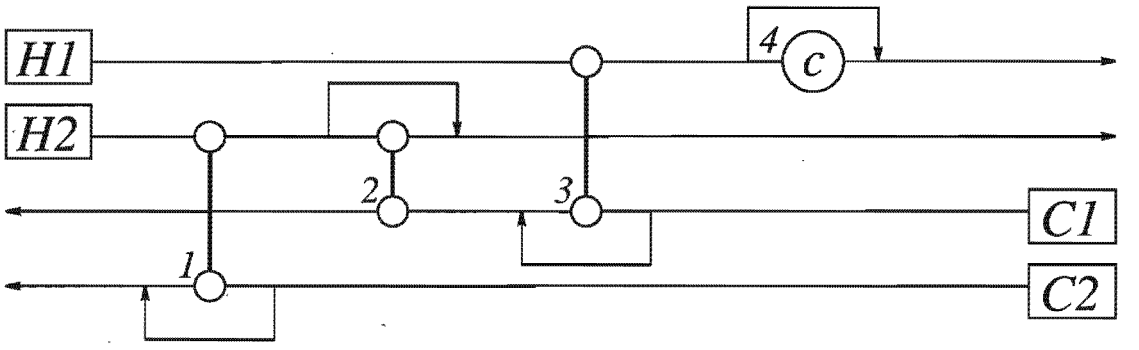

Figure 6.9: Design A for heat exchanger network example 2.

for asymptotic values of disturbances is found. The results of operability studies performed on these flexible designs are then compared with their economics.

The order followed in this study is the following:

- flexible design
- SIMULINK© modeling
- linear transfer function identification
- dynamic operability studies

The results of each of these sections is now presented.

Flexible Design

A flexible design optimization was performed on the following literature heat exchanger network example taken from Marselle *et al.* Marselle et al. (1982). The operating specifications are given in Table 6.7. The process flow diagrams are given in Figures 6.11 and 6.12.

With the nominal heat transfer coefficient $= 0.4\frac{kW}{^oC}$, an attempt was made to find a flexible design for an uncertainty range of $\pm$ 20% for the heat transfer coefficient of

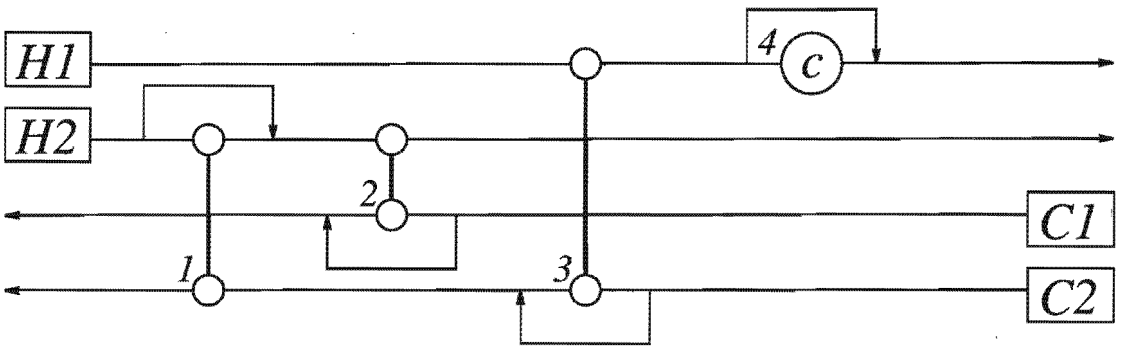

Figure 6.10: Design B for heat exchanger network example 2.

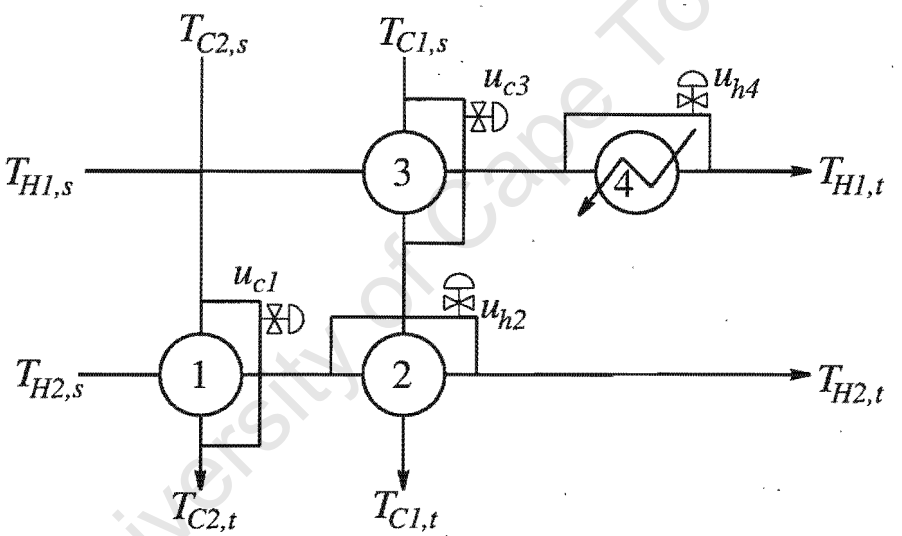

Figure 6.11: Process flow diagram for design A.

Table 6.7: Design and operating specifications for heat exchanger network example 2.

Stream Number	$T_s(°C)$	$T_t(°C)$	C_p $(kW/°C)$
H1	170	90	9
H2	250	140	10
C1	60	160	8
C2	115	200	6

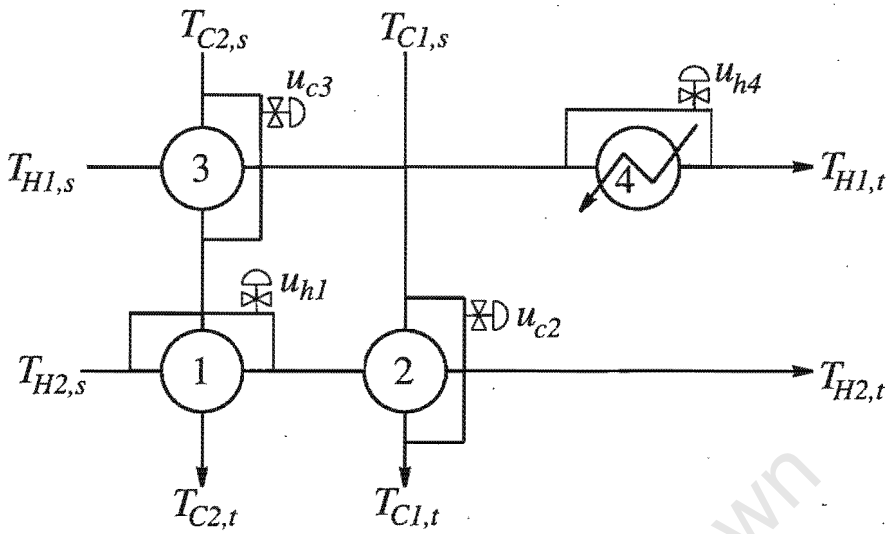

Figure 6.12: Process flow diagram for design B.

each of the 4 exchangers in each design (giving 16 vertices to the uncertain parameter region for each design, i.e. 2^{N_θ} where N_θ is the number of uncertain parameters) . The results are given in Table 6.8. The objective function of the flexible design optimization is the cost of the design which is based on the heat exchanger areas in the HEN as well as the flowrate of the cooling utilities; these minimized costs also appear in the table of results. The heat capacity flowrates for the cooling utilities were allowed to vary to compensate for parameter uncertainty as control variables.

Table 6.8: Flexibility results for HEN example 2.

	Design A	Design B
$A_1(m^2)$	24.183	15.763
$A_2(m^2)$	40.012	35.945
$A_3(m^2)$	7.7023	25.820
$A_4(cooler)(m^2)$	18.850	18.922
Cost (1991 $)	74967	77261

SIMULINK© modeling

The SIMULINK© model is given in Appendix A.

Linear identification

It was decided to follow a different strategy in identifying this example than what was used in example 1. It was thought that the identified state-space systems in example 1 were of too high an order, and this was mainly to account for dead-time which the model could not account for explicitly.

Thus the identification procedure here was performed through the use of an optimization framework in which the difference in outputs of the SIMULINK© model and a transfer function (plus dead time) was minimized, using the polynomial coefficients in the numerator and denominator of the transfer function as the decision variables in the optimization.

For design A which is described by the network in Figure 6.9, the following transfer function was identified.

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{299.9}{25.8s+192.3} & \frac{41.41s+278}{0.7553s+13.49} \\ \frac{15.38}{1.126s+1.67}e^{-0.2s} & \frac{263.3s+518.1}{4.27s+25.61} & \frac{-2.928}{0.2173s^2+1.012s+1.678}e^{-0.05s} & 0 \\ \frac{15.38}{0.7036s+1.1219}e^{-0.2s} & \frac{-2436}{26.09s+96.27} & \frac{-242.1}{26.25s+199.9}e^{-0.2s} & 0 \\ \frac{-4186s-3712}{62.06s+115.3} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_{c1} \\ u_{h2} \\ u_{c3} \\ u_{h4} \end{bmatrix} \quad (6.12)$$

The disturbance transfer function matrix has similarly been identified by the

transfer function matrix in equation 6.13.

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-0.3017s-0.1928}{0.2338s^2+1.711s+1.024} \\ \frac{38.19}{82.79s+128}e^{-0.2s} & \frac{-545.7}{62.49s+135.7} \\ \frac{88.68}{118.8s+216.7}e^{-0.2s} & \frac{-1733}{59.87s+243.9} \\ \frac{126}{21.24s+200.2} & 0 \end{bmatrix} \begin{bmatrix} \Delta T_{s,H2} \\ \Delta \dot{C}_{p,C1} \end{bmatrix} \tag{6.13}$$

Input-output comparison graphs that show both the SIMULINK© output and the identified transfer function output are given in Appendix B. A comparison of the SIMULINK© output trajectories and the output trajectories resulting from the identified transfer functions clearly reveals a very good fit.

Design B was identified in the same manner as design A, and the resulting transfer function matrix is given in equation 6.14, and the disturbance transfer function is given in equation 6.15. The graphs showing a comparison between the identified transfer functions and the SIMULINK model output is give in the appendix in Appendix B.

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{4.566}{0.2451s+1} & \frac{45.74s+400.3}{s+19.16} \\ \frac{1.589s+2.117}{0.1782s^2+0.8428s+1}e^{-0.2s} & \frac{154.3}{s+3.88} & \frac{0.5943s-27.42}{s^2+5.367s+14.36}e^{-0.85s} & 0 \\ \frac{1.567s+2.647}{0.1705s^2+0.72s+1}e^{-0.1s} & \frac{-17.81s-50.02}{0.176s+1} & \frac{-0.4323s-42.56}{s^2+6.352s+17.83}e^{-0.9} & 0 \\ \frac{-7.063}{0.06854s+1} & 0 & \frac{-2.94-172.6}{s^2+6.194s+27.11}e^{-0.4s} & 0 \end{bmatrix} \begin{bmatrix} u_{h1} \\ u_{c2} \\ u_{c3} \\ u_{h4} \end{bmatrix} \tag{6.14}$$

The disturbance transfer function matrix has similarly been identified by the

transfer function matrix in equation 6.15.

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \frac{0.518}{s+1.469}e^{-0.4s} & \frac{-245.6}{s+57.95} \\ \frac{0.6098}{s+1.382}e^{-0.25s} & \frac{-45.54}{s+6.52} \\ \frac{0.6673}{s+1.328} & 0 \end{bmatrix} \begin{bmatrix} \Delta T_{s,H2} \\ \Delta \dot{C}_{p,C1} \end{bmatrix} \tag{6.15}$$

Settings

The disturbance set included a $+10°C$ step in H2 and a $+0.8\ kW/°C$ step in the heat capacity flowrate of stream C2 with these step disturbances taken to occur simultaneously. The time horizon for simulation was 30 with a sampling time of 1, and the number of coefficients per transfer function for Q-parametrization was 10. For multiloop PI control, the input-output pairing as indicated on the process flow diagrams was:

- Design A: u_{h4} controls $T_{H1,t}$, u_{h2} controls $T_{H2,t}$, u_{c2} controls $T_{C1,t}$ and u_{c1} controls $T_{C2,t}$;
- Design B: u_{h4} controls $T_{H1,t}$, u_{h1} controls $T_{H2,t}$, u_{c2} controls $T_{C1,t}$ and u_{c3} controls $T_{C2,t}$;

The manipulated variable constraints for design A were

$$\begin{aligned} -0.0816 &\leq u_{c1} \leq 0.9184 \\ -0.1476 &\leq u_{h2} \leq 0.8524 \\ 0.0000 &\leq u_{c3} \leq 1.0000 \\ -0.1283 &\leq u_{h4} \leq 0.8717 \end{aligned}$$

Table 6.9: Operability results for dynamic operability analysis example 2 for designs A and B

	Design A	Design B
OOLIT	447.49	288.26
Q_{10}	447.50	288.26
MPC	516.44	376.15
PI	576.83	—

Table 6.10: *Normalized* operability results for dynamic operability analysis example 2 for designs A and B

	Design A	Design B
OOLIT	1.55	**1.00**
Q_{10}	1.55	1.00
MPC	1.79	1.30
PI	2.00	—

and for design B were

$$-0.0821 \leq u_{h1} \leq 0.9179$$

$$-0.1368 \leq u_{c2} \leq 0.8632$$

$$-0.0243 \leq u_{c3} \leq 0.9757$$

$$-0.1280 \leq u_{h4} \leq 0.8720$$

but these bounds did not become active for any of the solution trajectories in the dynamic operability optimizations, which can be verified from the graphs of the solution trajectories in Appendix C.

Dynamic operability results

The results for designs A and B are presented in Table 6.9, and the normalized results are shown in Table 6.10. The following remarks can be made:

1. Design B exhibits the best performance (lowest ISE) for both design alternatives under the disturbance test with the optimal manipulated input trajectory

found for each design ($\varphi_{OOLIT}^{Design\ B} = 1$);

2. Design B predicts *consistently* better performance for each of the four control methods, except for PI control for which a stable PI controller for design B could not be found;

3. The dynamic performance obtainable in Design B using an optimally tuned DMC controller is better than the absolute best possible performance achievable for Design A($\varphi_{OOLIT}^{Design\ A} > \varphi_{DMC}^{Design\ B}$).

These results appear to indicate that Design B is a more operable alternative for the disturbance set tested, but do not explain why a stable PI controller could not be found (or therefore optimized) for design B. A further analysis of the designs using open-loop measures explains why this may occur.

Design A has no RHP zeros, but design B has a RHP zero at 140.24. Even though RHP zeros do limit the achievable performance of a system, the effect is not as great when the RHP zero is reasonably far from the origin as in this case.

For measures that are matrices in which the ordering of the elements corresponds to the ordering of the elements in the plant transfer matrices, the matrices are not diagonalized; With the RGA, for example, the columns from left to right correspond to the order of the bypasses on the flow diagram figures pertaining to this example, as viewed from left to right. Each row in turn corresponds to the ordering of the process streams in the flow diagrams, from top to bottom. This format is used to allow an easy visual comparison with the open-loop measures and the flow diagrams.

The relative gain arrays for designs A and B are shown in equations 6.16 and 6.17 respectively.

$$RGA_A = \begin{bmatrix} 0 & 0 & 0 & 1.0000 \\ 0 & -1.2463 & 2.2463 & 0 \\ 0 & 2.2463 & -1.2463 & 0 \\ 1.0000 & 0 & 0 & 0 \end{bmatrix} \tag{6.16}$$

$$RGA_B = \begin{bmatrix} 0 & 0 & 0 & 1.000 \\ -4482 & -1.788 & 4484.8 & 0 \\ 4456 & 2.788 & -4457 & 0 \\ 27.51 & 0 & -26.51 & 0 \end{bmatrix} \tag{6.17}$$

The relative gain array for design A suggests that while some interaction may occur, the pairing indicated in Figure 6.9 is generally acceptable. The RGA for design B, however, indicates that severe interactions are present, and the input-output pairing (bypass placement) shown in Figure 6.10 may not be optimal. This could certainly explain why it could be very difficult to use decentralized control.

The performance relative gain arrays for designs A and B appear in equations 6.18 and 6.19.

$$\Gamma_A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -1.2463 & 26.036 & -1.9695 \\ 0 & 0.10752 & -1.2463 & 0.094276 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.18}$$

$$\Gamma_B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -84194 & -1.7882 & 93403 & -20415 \\ 4017.8 & 0.13305 & -4457.3 & 974.24 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.19}$$

The PRGA is more illustrative than the RGA for this example, and indicates that streams H2 and C1 are coupled for both design alternatives. For design B, however,

the effect is much worse since the magnitude of the coupling elements is much larger. This result reinforces the idea that design B, while capable of better dynamic performance with multivariable controllers, is not a suitable design alternative if multiloop PI control is to be used.

The condition numbers for designs A and B are

$$\gamma_A = 68.0089 \qquad \gamma_B = 262206$$

The condition number for design B is far greater than that for design A, and this indicates that design B should exhibit controllability problems, particularly with regard to the sensitivity to model uncertainty although this aspect is not considered here. However, the dynamic operability measures in Table 6.9 show that in the absence of model error considerations, the condition number may not provide valuable information regarding controllability. The same discussion applies to the disturbance condition numbers

$$\gamma_{d,A} = [110.561 \quad 10.9380] \qquad \gamma_{d,B} = [390103 \quad 35526.1]$$

The closed-loop disturbance gains for designs A and B appear in the equation set 6.20.

$$\Delta_{CLDG,A} = \begin{bmatrix} 0 & 0 \\ 0.144 & 7.75 \\ -0.0181 & -2.59 \\ 0 & 0 \end{bmatrix} \qquad \Delta_{CLDG,B} = \begin{bmatrix} 0 & 0 \\ 0.199 & 8.24 \\ 5.17 & -8056 \\ 0 & 0 \end{bmatrix} \tag{6.20}$$

The CLDG for design B, and in particular the closed-loop gain of disturbance 2 on C1 indicates that this output is very sensitive to this disturbance. Note that this information is not available from the open-loop disturbance gain for design B in equation 6.21, where the corresponding value is $8056/7.14 \approx 10^3$ times less. However, the dynamic performance measures in Table 6.9 indicate that this does not affect the

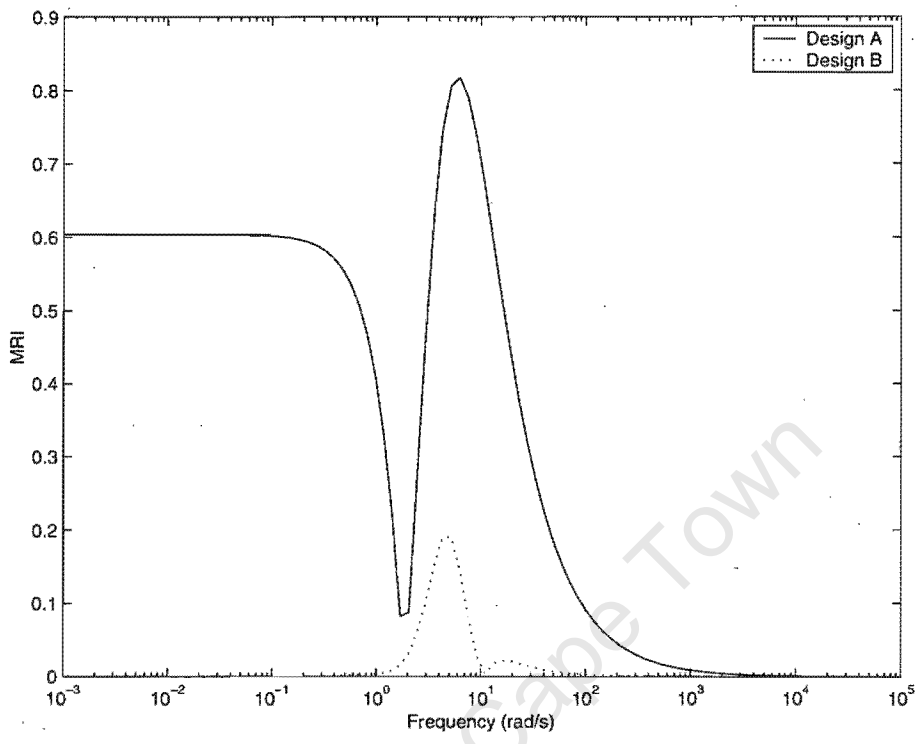

Figure 6.13: Morari resilience indices for designs A and B as a function of frequency

dynamic performance of the closed system using multivariable controllers. This is in agreement with theory, because Wolff et al. (1992) indicate that the CLDG has a particular bearing on decentralized control and input-output pairing issues.

The steady-state open-loop disturbance gains for designs A and B are

$$G_{d,A}(0) = \begin{bmatrix} 0 & 0.188 \\ 0.298 & 4.02 \\ 0.409 & 7.10 \\ 0.629 & 0 \end{bmatrix} \qquad G_{d,B}(0) = \begin{bmatrix} 0 & 0 \\ 0.353 & 4.24 \\ 0.441 & 7.14 \\ 0.502 & 0 \end{bmatrix} \tag{6.21}$$

The MRI for the two design alternatives is shown in Figure 6.13, and indicates that design A is more resilient than design B at lower frequencies because it has a larger value. For this particular case study, however, the dynamic performance measures indicate that design B is able to reject the disturbances with a smaller ISE

Table 6.11: Stream specifications for dynamic operability analysis example 3

Stream	$T_s(°C)$	$T_t(°C)$	$\dot{m}C_p\,(kW/°C)$	$\Delta E(kW)$
H1	150	60	20	-1800
H2	100	60	80	-3200
C1	20	70	25	1250
C2	25	90	35	2275

than design A with multivariable controllers, and so this result indicates that the MRI may not be a suitable measure of controllability if advanced control methods are implemented.

Referring to Table 6.8, one can see that design B is slightly more costly than design A, so there is a price to pay for improved dynamic performance.

6.1.4 Dynamic operability analysis example 3

The primary aim of this specific study was to confirm with another HEN example that the operability measures resulting from the optimal tuning of four different control strategies appear to be consistent with each other across different HEN configurations.

HEN specifications

The example studied here is based on a literature study presented by Gundersen et al. (1991). From the stream specifications presented in Table 6.11, it may be calculated that there is an excess of cooling required (3252 kW heat addition to cool streams required - 5000 kW heat removal from hot streams required= -1457 kW).

Thus, a cooler will be required in the HEN. Three proposed HEN configurations that will be assessed in terms of operability appear in Figure 6.14. The corresponding process flow diagrams appear in Figures 6.15, 6.16 and 6.17 respectively.

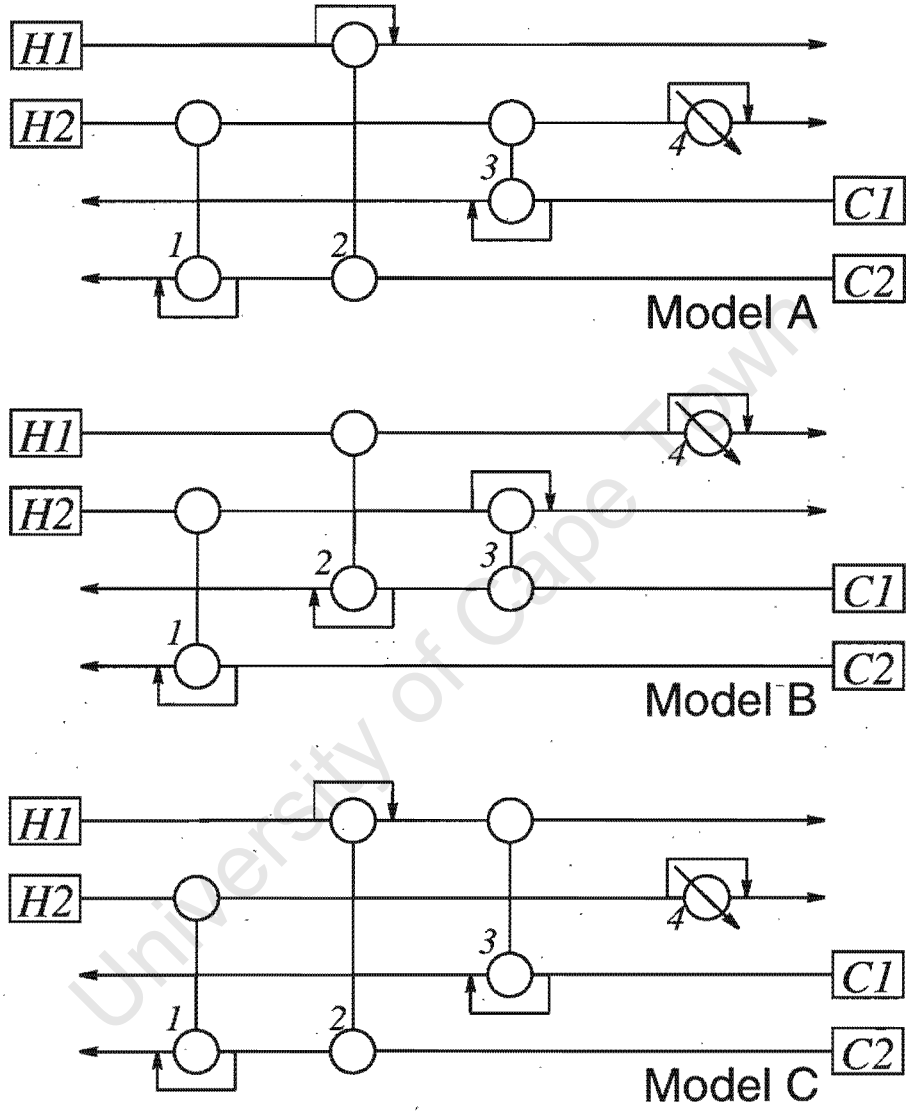

Figure 6.14: Three proposed HEN configurations for the stream specifications in Table 6.11 for dynamic operability assessment example 3

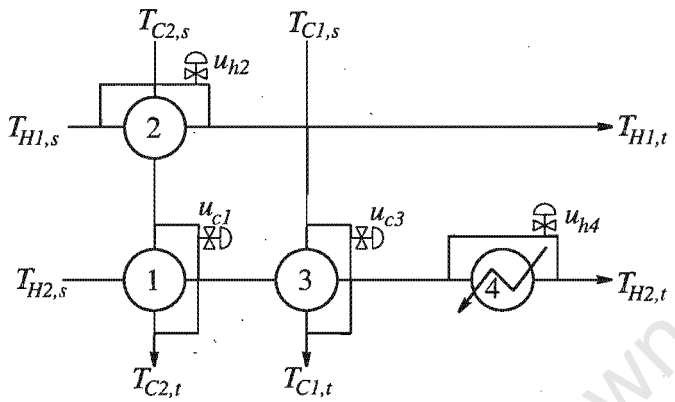

Figure 6.15: Process flow diagram for design A.

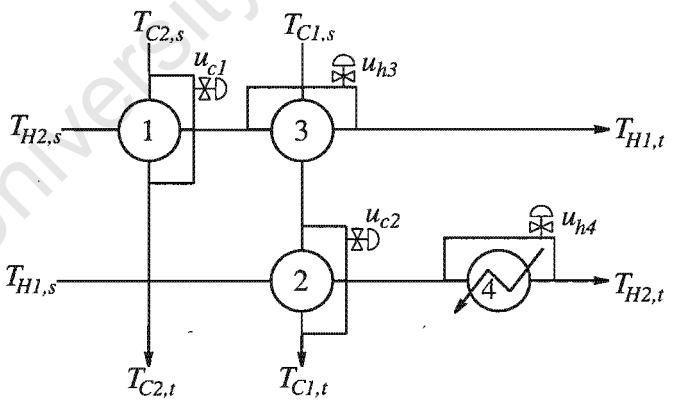

Figure 6.16: Process flow diagram for design B.

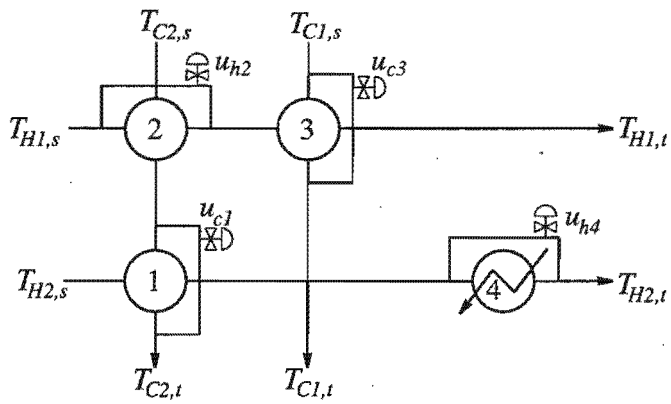

Figure 6.17: Process flow diagram for design C.

Table 6.12: Design variables and economics of three HEN configurations for dynamic operability analysis example 3

	Model A	Model B	Model C
Area 1 (m^2)	352.3	945.5	818.6
Area 2 (m^2)	369.8	41.47	53.25
Area 3 (m^2)	321.3	508.1	271.8
Area 4 (m^2)	348.4	217.4	348.4
Cooling $\dot{m}C_p$ ($kW/°C$)	51.67	51.67	51.67
Cost (1991 $)	423810	467800	429080

HEN design

For each of these three configurations, a design variable set was found that minimized the cost of each network while ensuring the network be able to operate at nominal conditions as well as be able to reject the disturbances in streams *H1* and *C1* used in the dynamic operability studies, subject to the satisfaction of target temperatures and manipulated variable bounds. These results are given in Table 6.12.

SIMULINK© model building

The SIMULINK© models for all three configurations were built in less than 30 minutes using the generic model blocks developed for earlier examples. This shows the power of SIMULINK© for use in building models of physical systems in which configuration plays an important issue, since it is necessary only to build one such block manually, and then repeat it in the model where necessary.

In addition, the graphical user interface of SIMULINK© allows one to build the model such that its on-screen structure corresponds visually with the physical description of the system. This intuitive correspondence allows for greater ease of error checking and debugging.

Linear model identification

Linear transfer function models were used to characterize the dynamic behaviour of each of the three HEN configurations around the nominal steady-state point. Once again, a very good fit was obtained.

The identified transfer function matrices are presented as:

Design A

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} 0 & \frac{70.47s+49.1}{0.7281s+1} & 0 & 0 \\ \frac{1.24}{0.7122s+1}e^{-0.4s} & \frac{-4.093}{1.111s+1}e^{-1.2s} & \frac{5.201}{0.5221s+1}e^{-0.2s} & \frac{4.553s+4.309}{0.1944s+1} \\ \frac{1.453}{0.7841s+1}e^{-0.2s} & \frac{-4.795}{1.185s+1}e^{-1.0s} & \frac{-29.41s-22.9}{0.533s+1} & 0 \\ \frac{-28.62s-5.288}{0.9126s+1} & \frac{-11.95}{0.9882s+1}e^{-0.8s} & 0 & 0 \end{bmatrix} \begin{bmatrix} u_{c1} \\ u_{h2} \\ u_{c3} \\ u_{h4} \end{bmatrix} \quad (6.22)$$

The disturbance transfer function matrix has similarly been identified by the

transfer function matrix in equation 6.13.

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} \frac{0.2803}{1.389s+1}e^{-0.2s} & 0 \\ \frac{0.05991}{1.047s+1}e^{-1.0s} & \frac{0.1543}{0.485s+1}e^{-0.15s} \\ \frac{0.07023}{1.174s+1}e^{-0.75s} & \frac{0.3255}{0.5457s+1}e^{-0.1s} \\ \frac{0.1751}{1.022s+1}e^{-0.5s} & 0 \end{bmatrix} \begin{bmatrix} \Delta T_{s,H1} \\ \Delta \dot{C}_{p,C1} \end{bmatrix} \tag{6.23}$$

Design B

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} \frac{0.9112}{2.033s+1}e^{-1.5s} & \frac{0.4561}{0.679s+1}e^{-0.2s} & \frac{-1.651}{1.59s+1}e^{-1.1s} & \frac{0.06654s+30.21}{8.778e-4s+1} \\ \frac{13.93}{1.287s+1}e^{-0.3s} & 0 & \frac{19.18s+7.298}{0.6944s+1} & 0 \\ \frac{11.09}{1.586s+1}e^{-1.1s} & \frac{-0.524s-0.9571}{0.05111s+1} & \frac{-20.1}{1.196s+1}e^{-0.65s} & 0 \\ \frac{-0.7033s-47.2}{0.0957s+1} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_{c1} \\ u_{c2} \\ u_{h3} \\ u_{h4} \end{bmatrix} \tag{6.24}$$

The disturbance transfer function matrix has similarly been identified by the transfer function matrix in equation 6.13.

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} \frac{0.3336}{0.7177s+1}e^{-0.15s} & \frac{0.01986}{s+1}e^{-1.2s} \\ 0 & \frac{0.2245}{s+1} \\ \frac{0.1399}{0.1s+1} & \frac{0.243}{s+1}e^{-0.75s} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta T_{s,H1} \\ \Delta \dot{C}_{p,C1} \end{bmatrix} \tag{6.25}$$

Design C

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} 0 & \frac{9.21s+1.234}{7.986s+1}e^{-0.6s} & \frac{14.09}{0.7878s+1}e^{-0.01s} & 0 \\ \frac{9.199}{0.9618s+1}e^{-0.25s} & \frac{-0.4797}{0.8504s+1}e^{-0.55s} & 0 & \frac{3.989s+3.938}{0.1795s+1} \\ 0 & \frac{9.159s+1.545}{6.509s+1}e^{-0.3s} & \frac{-0.083515-14.55}{0.001582s+1} & 0 \\ \frac{-49.44s-39.46}{0.7298s+1} & \frac{-0.3047}{s+1}e^{-0.7s} & 0 & 0 \end{bmatrix} \begin{bmatrix} u_{c1} \\ u_{h2} \\ u_{c3} \\ u_{h4} \end{bmatrix} \tag{6.26}$$

The disturbance transfer function matrix has similarly been identified by the transfer function matrix in equation 6.13.

$$\begin{bmatrix} T_{t,H1} \\ T_{t,H2} \\ T_{t,H3} \\ T_{t,H4} \end{bmatrix} = \begin{bmatrix} \frac{0.3049}{0.8137s+1}e^{-0.8s} & \frac{0.6098}{0.5645s+1} \\ \frac{0.03347}{s+1}e^{-0.5s} & 0 \\ \frac{0.3812}{0.8525s+1}e^{-0.5s} & \frac{0.5128}{0.6462s+1}e^{-0.05s} \\ \frac{0.02123}{s+1}e^{-0.6s} & 0 \end{bmatrix} \begin{bmatrix} \Delta T_{s,H1} \\ \Delta \dot{C}_{p,C1} \end{bmatrix} \tag{6.27}$$

Dynamic operability results

Using the linear dynamic models identified in the previous section, optimal controllers required to reject a step disturbance set of -5°C in *H1* and +5°C in *C2* were found via the optimization frameworks discussed in Chapter 4. For multiloop PI control, the input-output pairing as indicated on the process flow diagrams was:

- Design A: u_{c1} controls $T_{C2,t}$, u_{h2} controls $T_{H1,t}$, u_{c3} controls $T_{C1,t}$ and u_{h4} controls $T_{H2,t}$;

- Design B: u_{c1} controls $T_{C2,t}$, u_{c2} controls $T_{C1,t}$, u_{h3} controls $T_{H2,t}$ and u_{h4} controls $T_{H1,t}$;
- Design C: u_{c1} controls $T_{C2,t}$, u_{h2} controls $T_{H1,t}$, u_{c3} controls $T_{C1,t}$ and u_{h4} controls $T_{H2,t}$;

The manipulated variable constraints for design A were

$$-0.0540 \leq u_{h2} \leq 0.9460$$
$$-0.2175 \leq u_{h4} \leq 0.7825$$
$$0 \leq u_{c3} \leq 1.0000$$
$$-0.2732 \leq u_{c1} \leq 0.7269$$

and for design B were

$$-0.0512 \leq u_{h4} \leq 0.9488$$
$$-0.6039 \leq u_{h3} \leq 0.3961$$
$$0 \leq u_{c2} \leq 1.0000$$
$$-0.0053 \leq u_{c1} \leq 0.9947$$

and for design C were

$$-0.8735 \leq u_{h2} \leq 0.1265$$
$$-0.2322 \leq u_{h4} \leq 0.7678$$
$$-0.0724 \leq u_{c3} \leq 0.9276$$
$$-0.0386 \leq u_{c1} \leq 0.9614$$

but these bounds did not become active for any of the solution trajectories in the dynamic operability optimizations, which can be verified from the solution trajectories presented in Appendix D on page 162.

Table 6.13: Operability results for dynamic operability analysis example 3 for designs A,B and C

	Design A	Design B	Design C
OOLIT	21.381	34.591	129.39
Q_{10}	21.402	34.592	129.41
MPC	26.122	41.086	132.15
PI	457.92	556.95	665.48

Table 6.14: *Normalized* operability results for dynamic operability analysis example 3 for designs A, B and C

	Design A	Design B	Design C
OOLIT	1	1.617	6.052
Q_{10}	1.001	1.617	6.053
MPC	1.221	1.92	6.18
PI	21.42	26.04	31.12

The ISE measures for these optimal controller closed-loop responses are given in Table 6.13 and the normalized performance measures are given in Table 6.14. Regarding optimization settings, the time horizon for simulation was set to 600, the manipulated variable timestep to 10, P and M for optimal DMC each set to 40, and the number of coefficients for Q-parametrization to 10 (per transfer function matrix, i.e. $10 \times 4 \times 4 = 160$ in total).

Based on these results, the following remarks can be made:

1. Design A exhibits the best performance (lowest ISE) for all of the design alternatives under the disturbance test with the optimal open-loop input trajectory method ($\varphi_{OOLIT}^{Design\ A} = 1$);

2. Each design consistently predicts better or worse performance relative to the other designs for each dynamic operability measure; for instance, Design A predicts better disturbance rejection characteristics than designs B and C for all four control methods;

3. The optimal PI control performance measures for all three designs are markedly worse than all the other control methods (indicated by a higher ISE), and this was not seen in the other operability examples studied. This suggests that HENs of this size (4 input-4 output MIMO) may require more decoupling than smaller networks, since it is likely that interaction is greater.

4. Design C generally performs much worse than designs A and B for all performance measures, since the ISE values are higher indicating poorer disturbance rejection ability.

These results appear to indicate that Design A the more operable alternative for the disturbance set tested. However, questions remain as to why the optimal PI control method produced results so far off from the other methods. A further analysis of the designs using open-loop measures explains why this may occur.

None of the designs exhibit RHP-transmission zeros. For measures that are matrices in which the ordering of the elements corresponds to the ordering of the elements in the plant transfer matrices, the matrices are not diagonalized; With the RGA, for example, the columns from left to right correspond to the order of the bypasses on the flow diagram figures pertaining to this example, as viewed from left to right. Each row in turn corresponds to the ordering of the process streams in the flow diagrams, from top to bottom. This format is used to allow an easy visual comparison with the open-loop measures and the flow diagrams.

The relative gain arrays for designs A, B and C are shown in equations 6.28, 6.29 and 6.30 respectively.

$$RGA_A = \begin{bmatrix} 0 & 1.000 & 0 & 0 \\ 0 & 0 & 0 & 1.000 \\ 0 & 0 & 1.000 & 0 \\ 1.000 & 0 & 0 & 0 \end{bmatrix} \tag{6.28}$$

$$RGA_B = \begin{bmatrix} 0 & 0 & 0 & 1.000 \\ 0 & 0 & 1.000 & 0 \\ 0 & 1.000 & 0 & 0 \\ 1.000 & 0 & 0 & 0 \end{bmatrix} \tag{6.29}$$

$$RGA_C = \begin{bmatrix} 0 & -4.698 & 5.698 & 0 \\ 0 & 0 & 0 & 1.000 \\ 0 & 5.698 & -4.698 & 0 \\ 1.000 & 0 & 0 & 0 \end{bmatrix} \tag{6.30}$$

The RGA's for designs A and B indicate that loop interaction is minimal. Note that loop pairing has not yet been done on these RGA's, which is why the matrices are not diagonal; however, it is clear how the pairing for these two designs should proceed. For design C, there is strong loop interaction between inputs 2 and 3 and outputs 1 and 3, and this certainly helps to explain why the performance measures of design C were generally much worse than those for designs A and B.

The performance RGAs for designs A, B and C appear in equations 6.31, 6.32 and 6.32.

$$\Gamma_A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.9499 \\ 0 & 0 & 1.000 & -1.207 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.31}$$

$$\Gamma_B = \begin{bmatrix} 0 & 0 & 0 & 0.03016 \\ 0 & 0 & 0 & 0 \\ 0 & 21.00 & 0 & -0.3170 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.32}$$

$$\Gamma_C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1218 \\ -0.4142 & 53.65 & -4.698 & -5.567 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.33}$$

The PRGAs show that Designs A and C both show loop coupling in streams $H2$ and $C1$, although the effect is certainly more marked for Design C, which, in addition to have more coupling than design A (more non-zero elements), has a large entry of 53.65. Design B also has large entry of 21.00. This open-loop measures reinforces results found from dynamic operability optimizations to show why design A performs better than the other two.

The condition numbers for designs A, B and C are

$$\gamma_A = 12.569 \qquad \gamma_B = 159.16 \qquad \gamma_C = 217.01$$

The condition numbers for the three designs show the same trend as is shown in Table 6.14: Moving through designs A to C, poorer conditioning is observed due to the increased condition number. The same discussion applies to the disturbance condition numbers

$$\begin{aligned} \gamma_{d,A} &= [3.540 \quad 3.372] \\ \gamma_{d,B} &= [59.11 \quad 64.52] \\ \gamma_{d,C} &= [41.58 \quad 25.55] \end{aligned}$$

from which it may be interpreted that the disturbance set used to perturb the system lies in a direction easy for design A to manage, but more difficult for designs B and C.

The closed-loop disturbance gains for designs A, B and C appear in the equation set 6.34.

$$\Delta_{CLDG,A} = \begin{bmatrix} 0 & 0 \\ 0.02336 & 0 \\ 0.01349 & 0.3255 \\ 0 & 0 \end{bmatrix} \tag{6.34}$$

$$\Delta_{CLDG,B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0.6184 \\ 0 & 0 \end{bmatrix}$$

$$\Delta_{CLDG,C} = \begin{bmatrix} 0 & 0 \\ 0.1175 & -0.2065 \\ 0.002769 & 1.178 \\ 0 & 0 \end{bmatrix}$$

The closed-loop disturbance gains for the three designs generally indicate that there is not much difference between them, although it may be seen that design C, for which the worst performance was predicted, has the largest closed-loop gain (1.178 for disturbance 2) which is more difficult for the control system to reject. As in dynamic operability example 2, that information is not obvious from looking at

the steady-state open-loop disturbance gains for the three designs:

$$G_{d,A}(0) = \begin{bmatrix} 0.2803 & 0 \\ 0.05991 & 0.1543 \\ 0.07023 & 0.3255 \\ 0.1751 & 0 \end{bmatrix} \tag{6.35}$$

$$G_{d,B}(0) = \begin{bmatrix} 0.3336 & 0.01986 \\ 0 & 0.2245 \\ 0.1399 & 0.2430 \\ 0 & 0 \end{bmatrix} \tag{6.36}$$

$$G_{d,C}(0) = \begin{bmatrix} 0.3049 & 0.6098 \\ 0.03347 & 0 \\ 0.3812 & 0.5129 \\ 0.02123 & 0 \end{bmatrix} \tag{6.37}$$

The MRI for the three design alternatives is shown in Figure 6.18, and indicates that design A is more resilient than either design B or design C at most of the frequency range. Recalling that the dynamic operability ISE measures for the three designs showed that the performance predicted by PI controllers was significantly worse than for the other control methods, it is of interest to note that that information is not apparent when observing the MRI curves.

Referring to Table 6.12, one can see that design A is the cheaper design of the three, although this is not by a large margin. Since design A appears to lead in both economics as well as operability characteristics, it is likely that this design would be favoured for industrial application.

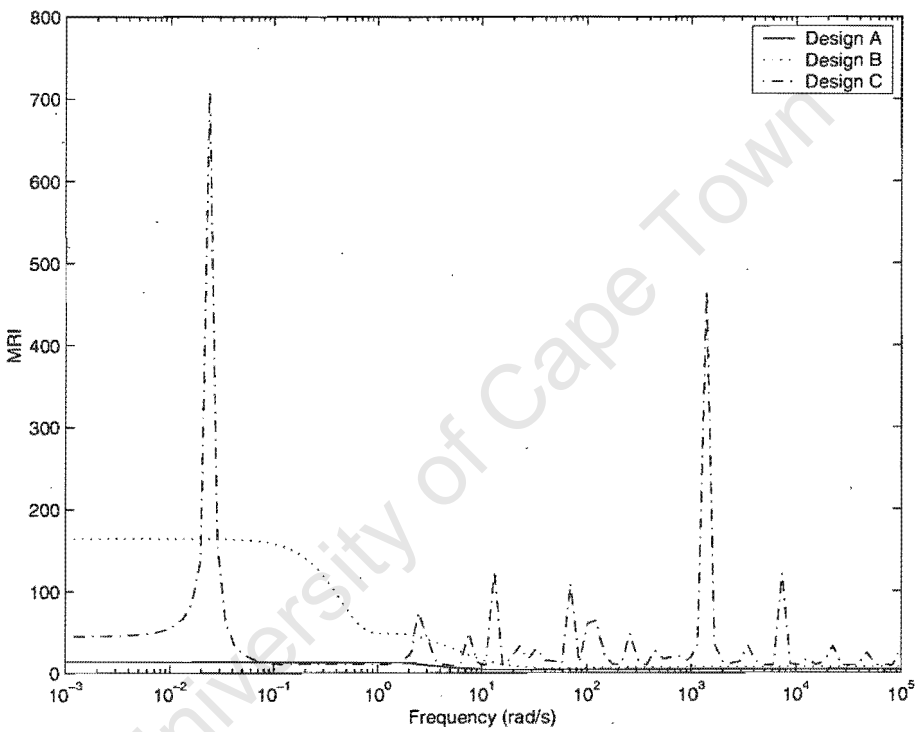

Figure 6.18: Morari resilience indices for designs A and B as a function of frequency

Chapter 7

Scope for Future Studies

There are currently a wide variety of methods available for assessing the operability of chemical plants as indicated in the literature survey in Chapter 2.

Open-loop measures provide valuable information about the expected dynamic performance of plants, but there is some difficulty in interpreting what the results obtained represent in the physical plant. In addition, there is a need for determining the effect of different performance-limiting characteristics simultaneously, since interaction may be involved. Optimization-based methods are able to do this, and recent studies have begun to explore the strategy of considering multiple performance-limiting factors as well as open-loop measures, either as multi-objective problems but more often as additional constraints.

However, a number measures cause undesirable characteristics in the optimization problems such as non-convexity; thus, a potential area of research may lie in considering a range of open-loop measures and other performance-limiting characteristics in optimization based strategies whilst retaining a convex problem formulation, particularly in the case of the Q-parametrization strategy. Specific attention should be given to the effects of input saturation and model-uncertainty on the dynamic performance of chemical plants since these are always present.

In addition, the methods used for the assessment of dynamic operability for HENs

may improve on the work presented here by implementing a nonlinear differential HEN model directly into the optimization and simulation strategies. It is hoped that the software presented in this work will accelerate the development of that objective.

Finally, the development of a simultaneous design and control multiobjective optimization problem for HENs could be investigated, in which discrete structural configuration decisions could be made on the basis of steady-state economics as well as dynamic operability issues.

Chapter 8

Conclusions

Four measures of dynamic operability have been investigated, corresponding to the optimal closed-loop performance for PI control, MPC control, linear control (via Q-parametrization) and open-loop control. These methods have been applied to three different HEN studies and for each a comparison between the different methods has been made.

It has been found that a comparison between alternative HEN designs for each method provides a quantitative measure of the achievable dynamic performance for each design, and this would allow the designer to assess different configurations on the basis of dynamic operability as well as conventional steady-state economic studies.

In addition, it was found that the relative improvement in dynamic performance with the different operability assessment methods provides a means to assess the improvement that could be obtained using more advanced control algorithms. This result allows dynamic performance to be traded-off with controller complexity which has an impact on installation and maintenance costs of the control system. In addition, plants that show good steady-state economics but poor closed-loop controllability for multi-loop controllers may become more feasible using more advanced control methods, since the steady-state economically optimal operating point may be maintained more easily through faster disturbance rejection.

For each HEN operability study, it was found that the different operability assessment methods were consistent in identifying which design exhibited better dynamic performance and which did not, *but only for multivariable controllers.* This result is not proof that this consistency will occur in every case, but it does seem to suggest that the different operability measures will generally rank alternative HEN designs in the same order.

Example 2 shows that it is possible to have different multivariable controllers predict better performance for one design than another, but still have poor decentralized control performance (in this case, unstable) for the 'better' design. It is suggested that a distinction be made between controllers with and without input-output decoupling characterstics in future operability studies in which comparisons between different controller algorithms be made.

For multivariable controllers, however, this result of consistency implies that if one of the operability measures predicts a design to be better than another, then the other three will show a similar trend. Thus for instance, if the optimal linear controller which gives a measure of the best dynamic performance of a system under feedback ranked a number of alternative designs according to dynamic performance, it seems likely that multivariable controllers of lower order will give the same ranking.

Lastly, a software package has been developed that allows the studies described in this work to be carried out in a convenient way. It is hoped that this software will further the study of dynamic operability and related areas of research.

Bibliography

Anderson, J.S. 1966. "A Practical Problem in Dynamic Heat Transfer." *Chem. Engr.*, no. CE97-CE103.

Bahri, P.A., J.A. Bandoni, and J.A. Romagnoli. 1995. "An Integrated Procedure for Flexibility and Controllability Analysis in Design and Control of Chemical Processes." *AIChE 1995 Annual Meeting*, November 12–17, Miami, USA.

Bahri, P.A., A. Bandoni, and J. Romagnoli. 1996. "Operability Assessment in Chemical Plants." *Comput. Chem. Eng.* 20 (Suppl.): S787–S792.

Barton, G.W., W.K. Chan, and J.D. Perkins. 1991. "Interaction Between Process Design and Process Control: The Role of Open Loop Indicators." *J. Proc. Cont.*, vol. 1 (May).

Boyd, S.P., C.H. Barratt, and S.A. Norman. 1990. "Linear Controller Design: Limits of Achievable Performance Via Convex Optimization." *Proceedings of the IEEE* 78 (3): 529–564.

Bristol, E.H. 1966. "On a New Measure of Interaction for Multivariable Process Control." *IEEE Trans. Autom. Control* AC-11:133–134.

Cao, Y., D. Biss, and J.D. Perkins. 1994. "Assessment of Input-Output Controllability in the Presence of Control Constraints." *IFAC workshop on integration of process design and control*, no. Baltimore, Maryland, USA:35–40.

Chiang, T. and W.L. Luyben. 1988. "Comparison of the Dynamic Performances

of Three Heat-Integrated Distillation Configurations." *Ind. Eng. Chem. Res.* 27:99–104.

Cutler, C.R. and B.L. Ramaker. 1979. "Dynamic Matrix Control — A Computer Algorithm." *AIChE National Meeting, Houston, TX.*

Downs, J.J. and B. Ogunnaike. 1995. "Design for Control and Operability: An Industrial Perspective." *Fourth International Conference on the Foundations of Computer Aided Process Design, AIChE Symposium Series* 91 (304): 115–123.

Doyle, J., B. Francis, and A. Tannenbaum. 1992. Feedback Control Theory.

Garcia, C.E., D.M. Prett, and M.Morari. 1989. "Model Predictive Control: Theory and Practice — A Survey." *Automatica* 25 (3): 335–348.

Glemmestad, B. 1997. "Optimal Operation of Integrated Processes." *PhD Thesis, Norwegian University of Science and Technology.*

Grossmann, I.E. and C.A. Floudas. 1987. "Active Constraint Strategy for Flexibility Analysis in Chemical Processes." *Comput. Chem. Eng.* 11 (6): 675–693.

Grossmann, I.E. and M. Morari. 1984. "Operability, Resiliency and Flexibility - Process Design Objectives for a Changing World." *Proceedings of the second international conference on foundations of computer-aided process design*, no. Snowmass, Colorado, USA:931–1030.

Gundersen, T. and L. Naess. 1988. "The Synthesis of Cost Optimal Heat Exchanger Networks: An Industrial Review of the State of the Art." *Comput. Chem. Eng.* 12 (6): 503–530.

Gundersen, T., B. Sagli, N. Hydro, and K. Kiste. 1991. "Problems in Sequential and Simultaneous Strategies for Heat Exchanger Network Synthesis." *Comp. Oriented Proc. Eng.* 37 (2): 259–270.

Halemane, K.P. and I.E. Grossmann. 1983. "Optimal Process Design under Uncertainty." *AIChE J.* 29:425–433.

Holt, B.R. and M. Morari. 1985. "Design of Resilient Processing Plants-VI. The Effect of Right-Half-Plane Zeros on Dynamic Resilience." *Chem. Eng. Sci* 40 (1): 59–74.

Hovd, M. and S. Skogestad. 1992. "Simple Frequency Dependent Tools for Control System Analysis, Structure Selection and Design." *Automatica* 28:989–996.

Kotjabasakis, E. and B. Linnhoff. 1987. "Better System Design Reduces Heat-Exchanger Fouling Costs." *Oil and Gas Journal*, September, 49–56.

Kwakernaak, H. and R. Sivan. 1972. "Linear Optimal Control Systems." no. Wiley, New York, USA.

Marselle, D.F., M. Morari, and D.F. Rudd. 1982. "Design of Resilient Processing Plants—II. Design and control of energy management systems." *Chem. Eng. Sci.* 37 (2): 259–270.

Mathisen, K.W. 1994. "Integrated Design and Control of Heat Exchanger Networks." no. Thesis, University of Trondheim.

Mathisen, K.W. and S. Skogestad. 1992. "Design, Operation And Control of Resilient Heat Exchanger Networks." *Paper 141g at AIChE Annual Meeting, Miami Beach*, November.

Mohideen, M.J., J.D. Perkins, and E.N. Pistikopoulos. 1996. "Optimal Design of Dynamic Systems under Uncertainty." *AIChE Journal* 42 (8): 2251–2272 (August).

Morari, M. 1983. "Design of Resilient Processing Plants-III. A General Framework for the Assessment of Dynamic Resilience." *Chem. Eng. Sci.* 38 (11): 1881–1891.

Morari, M. and J.H. Lee. 1997. "Model Predictive Control: Past, Present and

Future." *Presented at PSE'97/ESCAPE-7 Symposium, Trondheim, Norway, (May 1997).*

Nisenfeld, A.E. 1973. "Applying Control Computers to an Integrated Plant." *Chem. Eng. Prog.* 69:45–48.

Nishida, N., S. Kobayashi, and A. Ichikawa. 1971. "Optimal Synthesis of Heat Exchange Systems: Necessary Conditions for Minimum Heat Transfer Area and their Application to Systems Synthesis." *Chem. Eng. Sci.* 26:1841–1856.

Pai, C.C. and R.R. Hughes. 1987. "Strategies for Formulating and Solving Two-Stage Problems for Process Design under Uncertainty." *Comput. Chem. Eng.* 11 (6): 695–706.

Palazoglu, A. and Y. Arkun. 1986. "A Multiobjective Approach to Design Chemical Plants with Robust Dynamic Operability Characteristics." *Comput. Chem. Eng.* 10 (6): 567–575.

Papalexandri, K.P. and E.N. Pistikopoulos. 1994. "Synthesis and Retrofit Design of Operable Heat Exchanger Networks 1. Flexibily and Structural Controllability Aspects." *Ind. Eng. Chem. Res.* 33:1718–1737.

Perkins, J.D. and M.P.F. Wong. 1985. "Assessing Controllability of Chemical Plants." *Chem. Eng. Res. Des.*, vol. 63 (November).

Pistikopoulos, E.N. 1995. "Uncertainty in Process Design and Operations." *Comput. Chem. Eng.* 19 (Suppl.): S553–S563.

Pistikopoulos, E.N. and T.A. Mazzuchi. 1990. "A Novel Flexibility Analysis Approach for Processes with Stochastic Parameters." *Comput. Chem. Eng.* 14 (9): 991–1000.

Psarris, P. and C.A. Floudas. 1991a. "Dynamic Operability of MIMO Systems with

Time Delays and Transmission Zeros-I. Assessment." *Chem. Eng. Sci.* 46 (10): 2691–2707.

Psarris, P. and C.A. Floudas. 1991b. "Dynamic Operability of MIMO Systems with Time Delays and Transmission Zeros-I. Enhancement." *Chem. Eng. Sci.* 46 (10): 2709–2728.

Rosenbrock, H.H. 1970. "State Space and Multivariable Theory." no. Nelson, London, UK.

Ross, R. and C.L.E. Swartz. 1995. "The Application of Dynamic Resiliency Assessment to Flotation Circuit Design." *Proceedings of the 8th IFAC International Symposium on Automation in Mining, Mineral and Metal Processing, Sun City, I.J. Barker, ed.*

Ross, R. and C.L.E. Swartz. 1997. "Inclusion of Model Uncertainty in a Computational Framework for Dynamic Operability Assessment." *Joint 5th International Symposium on Process Systems Engineering and 30th European Symposium on Computer Aided Process Engineering, Trondheim (1997); Comput. Chem. Engng.* 21 (Suppl.,): S415–S420 (April).

Russell, L.W. and J.D. Perkins. 1987. "Towards a Method for Diagnosis of Controllability and Operability Problems in Chemical Plants." *Chem. Eng. Res. Des.* 65:453–461.

Saboo, A.K. and M. Morari. 1984. "Design of Resilient Processing Plants IV: Some new results on Heat Exchanger Network Synthesis." *Chem. Eng. Sci.* 39:579.

Shinskey, F.G. 1988. *Process Control Systems. Application, Design and Tuning.* 3rd. New York: McGraw-Hill Inc.

Skogestad, S. and M. Morari. 1987. "Design of Resilient Processing Plants-IX. Effect of Model Uncertainty on Dynamic Resilience." *Chem. Eng. Sci.* 42 (7): 1765–1780.

Skogestad, S. and I. Postlethwaite. 1996. "Multivariable Feedback Control : Analysis and Design." no. John Wiley and Sons, London, United Kingdom.

Skogestad, S., M. Morari, and J.C. Doyle. 1988. "Robust Control of Ill-Conditioned Plants: High-Purity Distillation." *IEEE transactions on automatic control* 33 (12): 1092–1105.

Swaney, R.E. and I.E. Grossmann. 1985a. "An Index for Operational Flexibility in Chemical Process Design. Part I: Formulation and Theory." *AIChE J.* 31 (4): 621–630.

Swaney, R.E. and I.E. Grossmann. 1985b. "An Index for Operational Flexibility in Chemical Process Design. Part II: Computational Alogorithms." *AIChE J.* 31 (4): 631–641.

Swartz, C.L.E. 1996. "A Computational Framework for Dynamic Operability Assessment." *Comput. Chem. Eng* 20 (4): 365–371.

Wolff, E.A. and S. Skogestad. 1992. "Controllability of Integrated Plants applied to Recycle Systems." *AIChE Spring National Mtg.*, pp. New Orleans, March/April 1992.

Wolff, E.A., S. Skogestad, M. Hovd, and K.N. Mathisen. 1992. "A Procedure for Controllability Analysis." *IFAC workshop on Interactions between process design and process control, London, Sept. 1992.*

Youla, D.C., H.A. Jabr, and J.J. Bongiorno. 1976. "Modern Wiener-Hopf design of optimal controllers. Paper II." *IEEE Tr. Aut. Cont.* AC-21:319–338.

Yu, C.C. and W.L. Luyben. 1986. "Design of Multiloop SISO Controllers in Multivariable Processes." *Ind. Eng. Chem. Process Des. Dev.* 25:498–503.

Appendix A

SIMULINK Implementation and graphic description

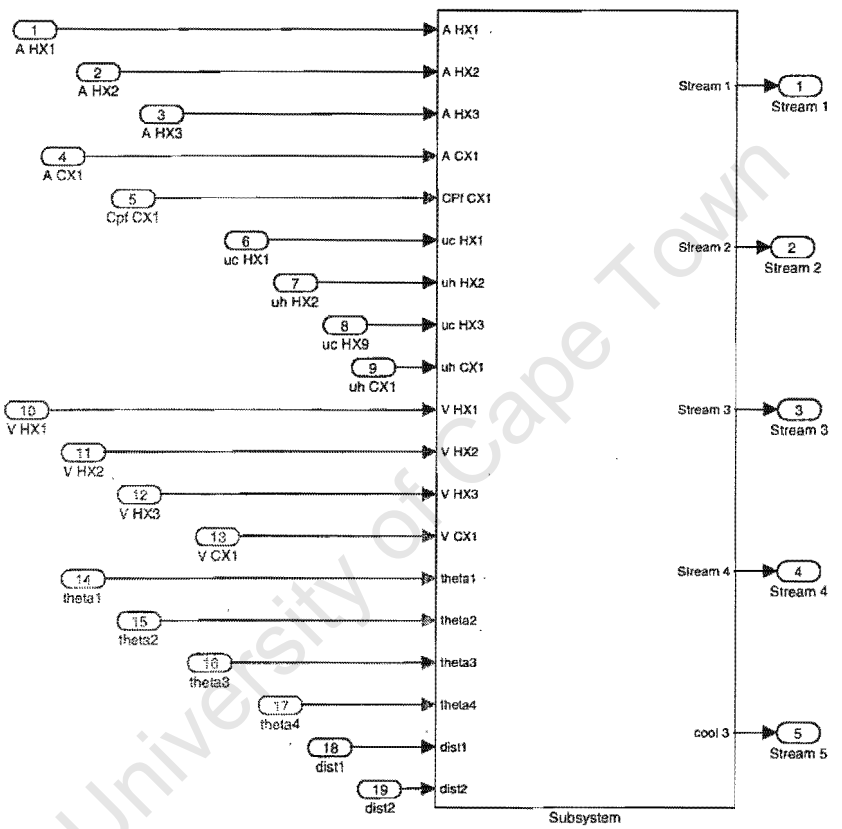

Figure A.1: The top level of the complete simulink model of a heat exchanger network.

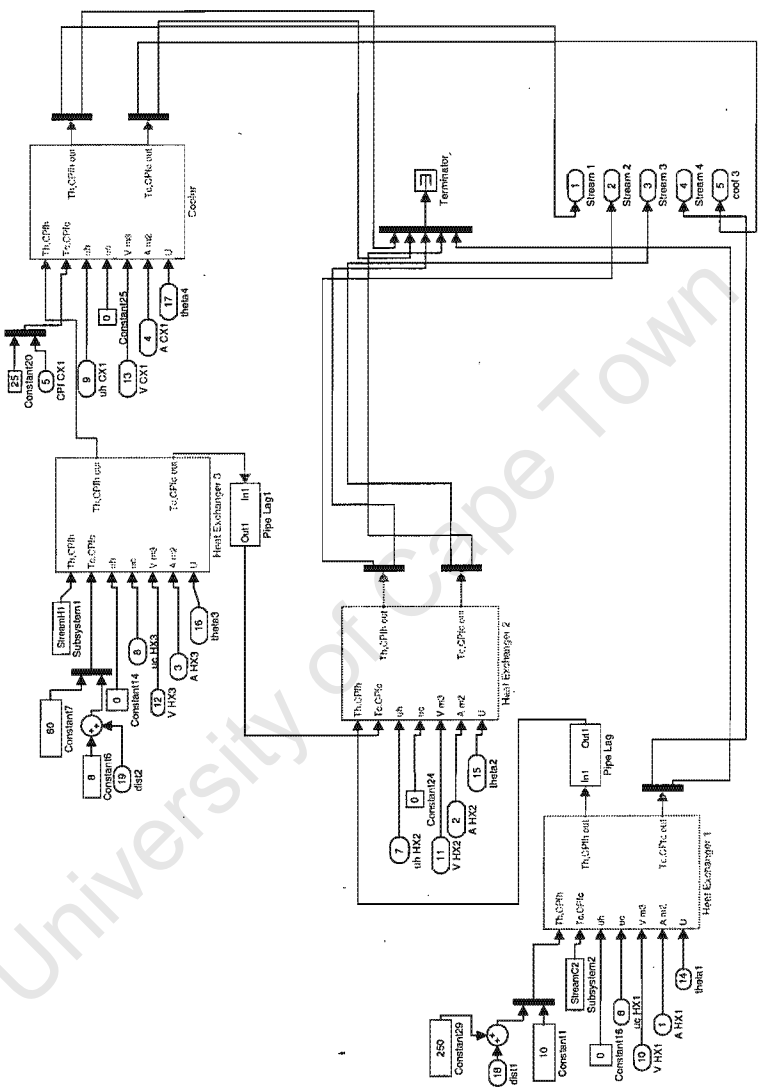

Figure A.2: View of a complete heat exchanger network.

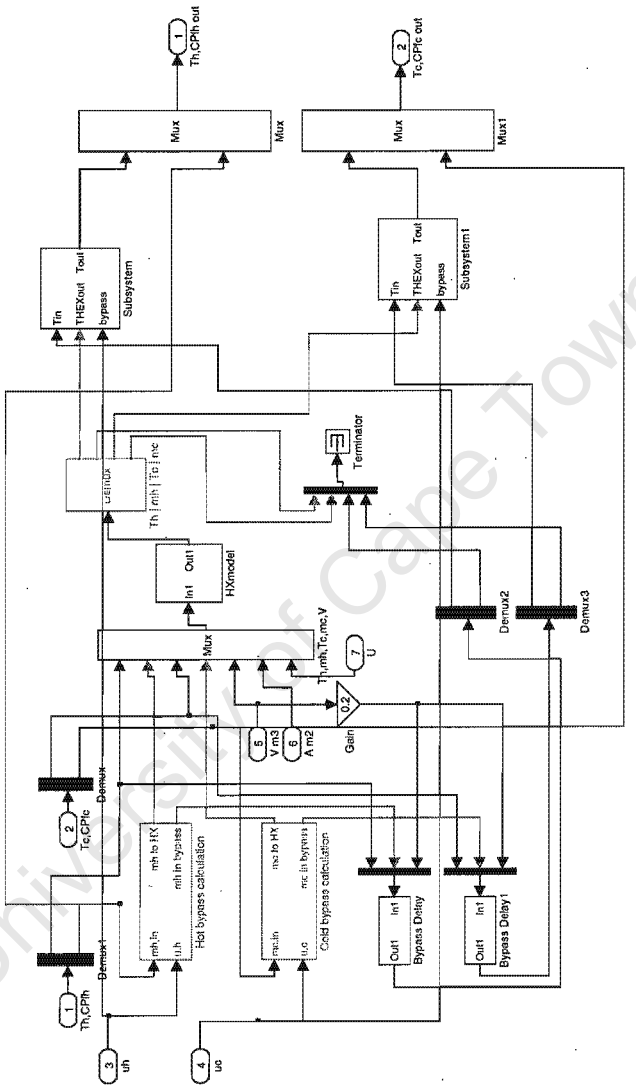

Figure A.3: Model for a single heat exchanger.

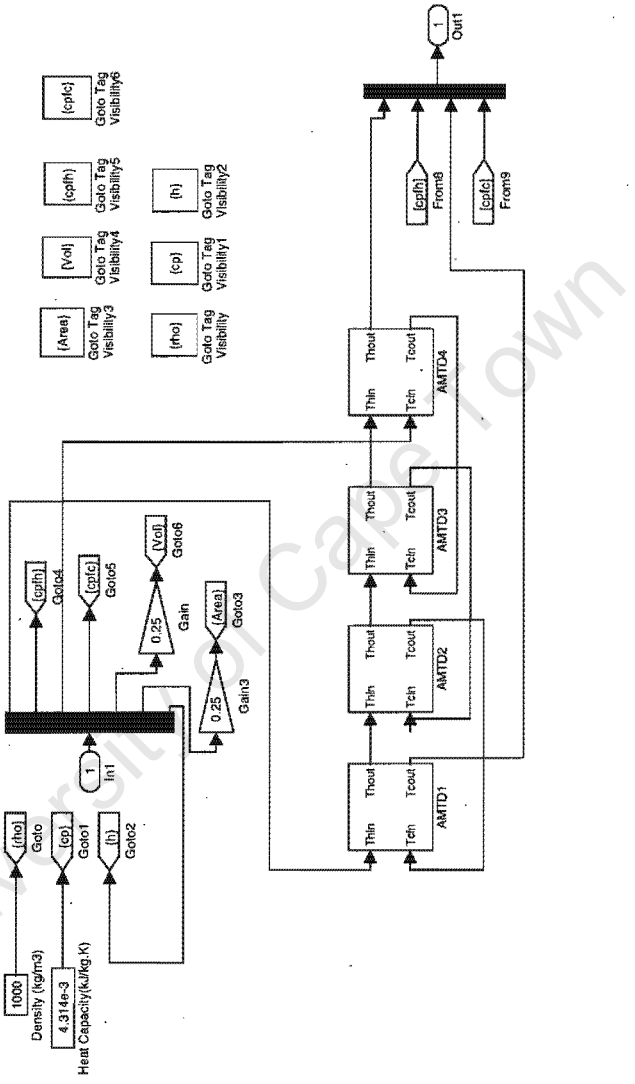

Figure A.4: Core heat exchanger model using four well mixed cells.

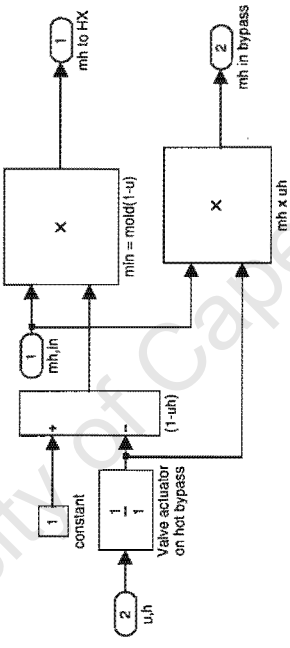

Figure A.5: SIMULINK model for splitting a heat exchanger process steam fro partial bypass.

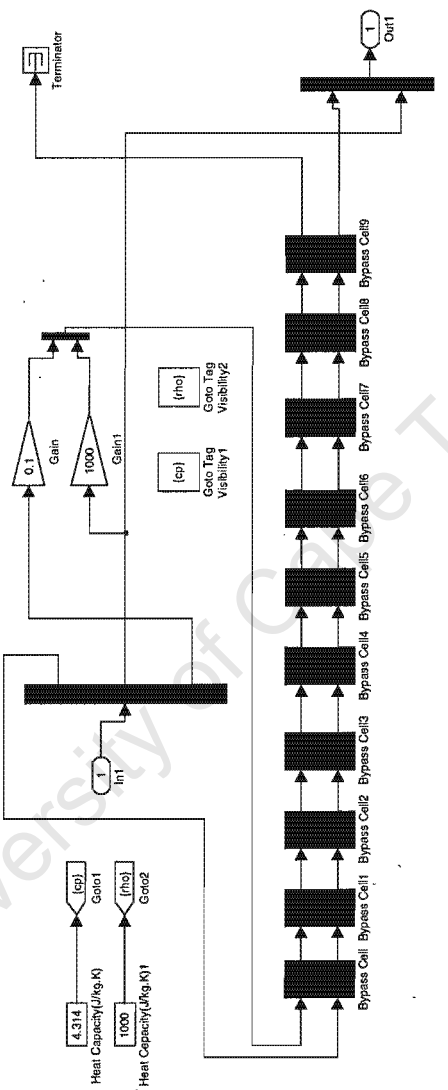

Figure A.6: Heat exchanger bypass model using ten well mixed cells to simulate heat flow lag through the bypass SISO system

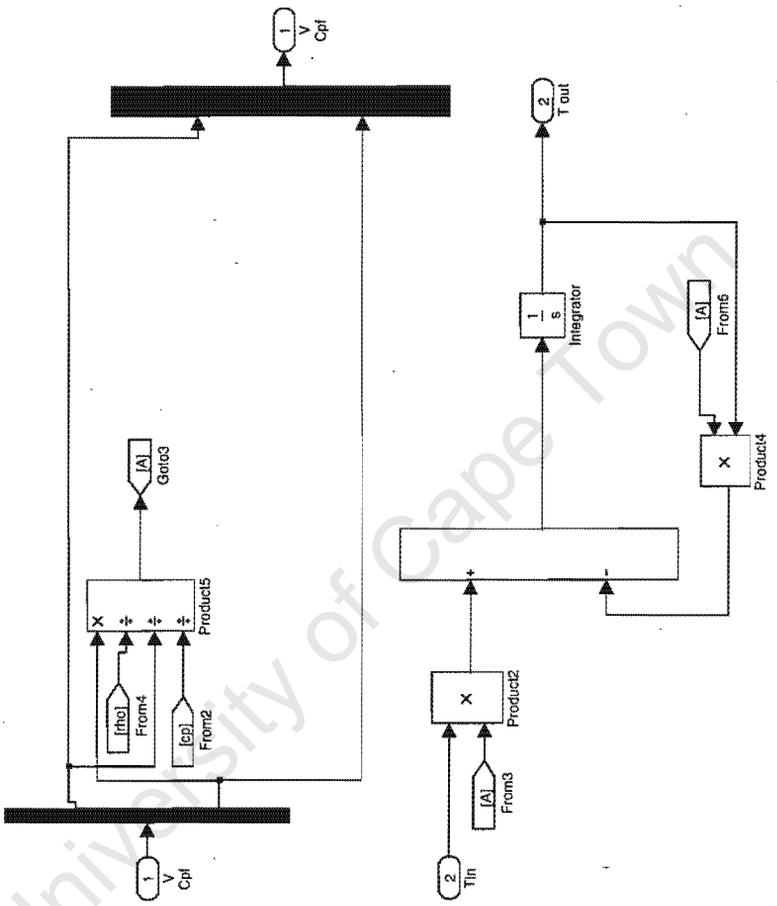

Figure A.7: A well mixed cell through which heat flow in the bypass stream occurs.

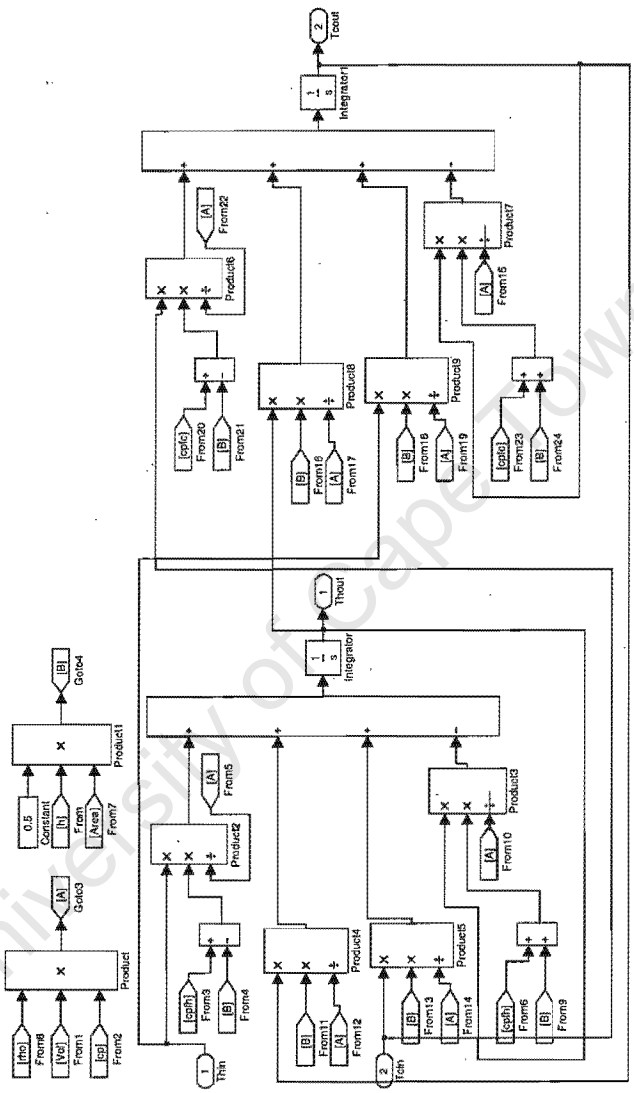

Figure A.8: A well mixed heat exchange cell using the arithmetic mean temperature difference (AMTD)

Appendix B

Verification of the accuracy of linear transfer function identification

Identification graphs for operability analysis example 2, design A

Here graphs are provided that compare the SIMULINK© dynamic nonlinear model output with the output from linear transfer functions found through the optimization identification procedure. These graphs are described and referred to in §6.1.3 on page 100, and the reader is referred to that section for a description of how these graphs are generated and a discussion these results.

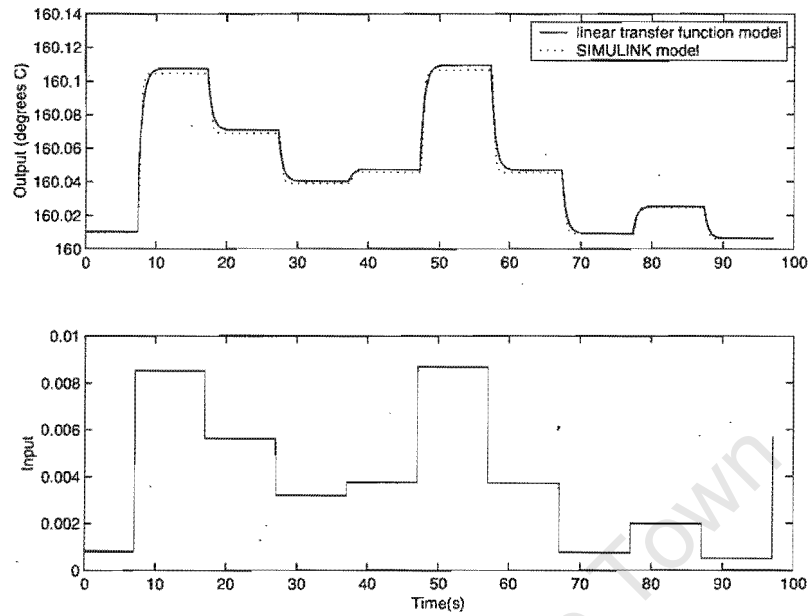

Figure B.1: Comparison of SIMULINK© and identified transfer function output from **input 1** and **output 3**

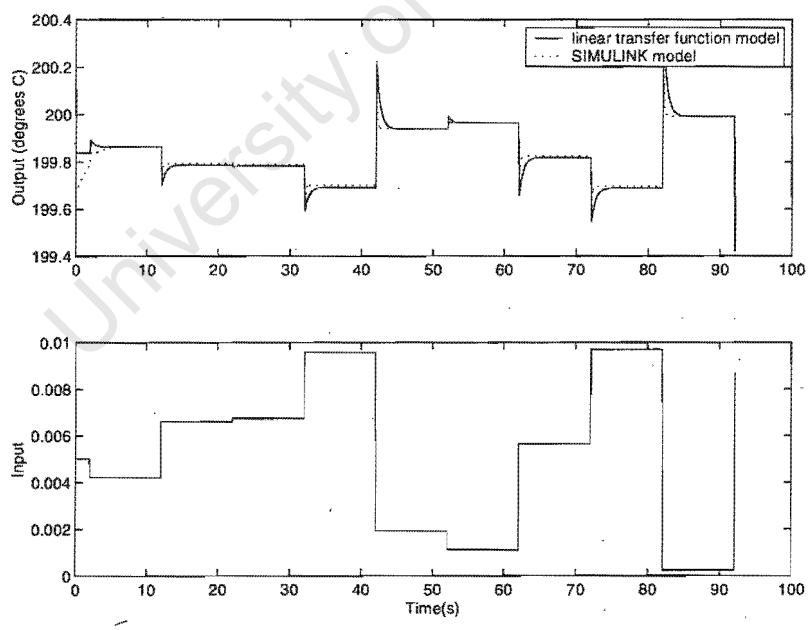

Figure B.2: Comparison of SIMULINK© and identified transfer function output from **input 1** and **output 4**

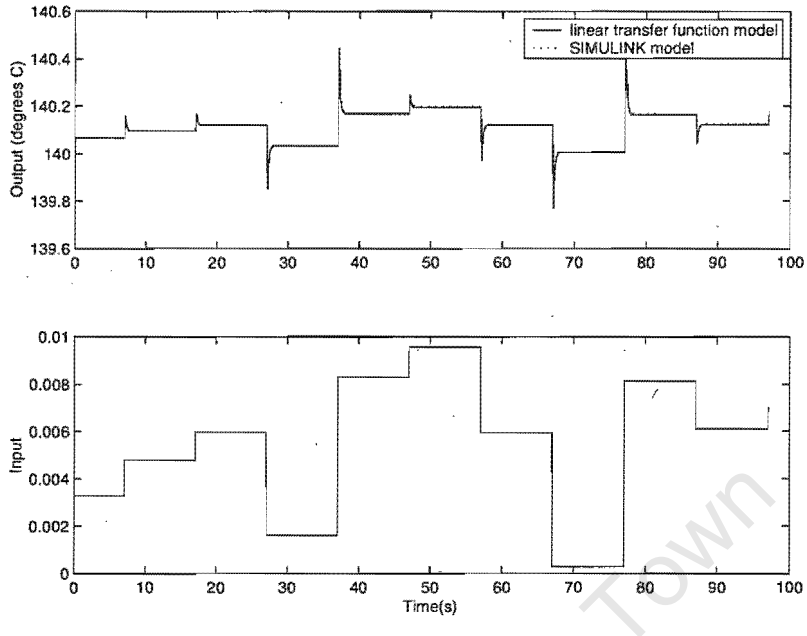

Figure B.3: Comparison of SIMULINK© and identified transfer function output from **input 2** and **output 2**

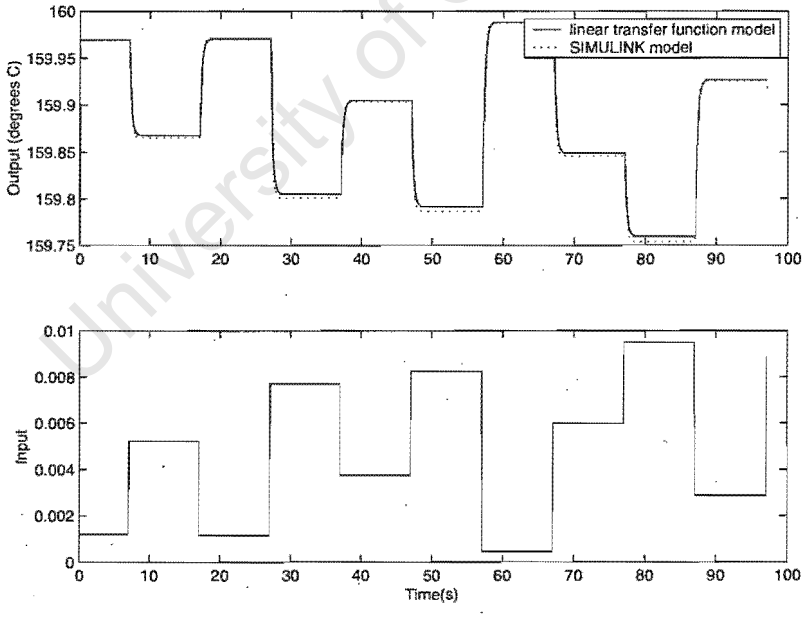

Figure B.4: Comparison of SIMULINK© and identified transfer function output from **input 2** and **output 3**

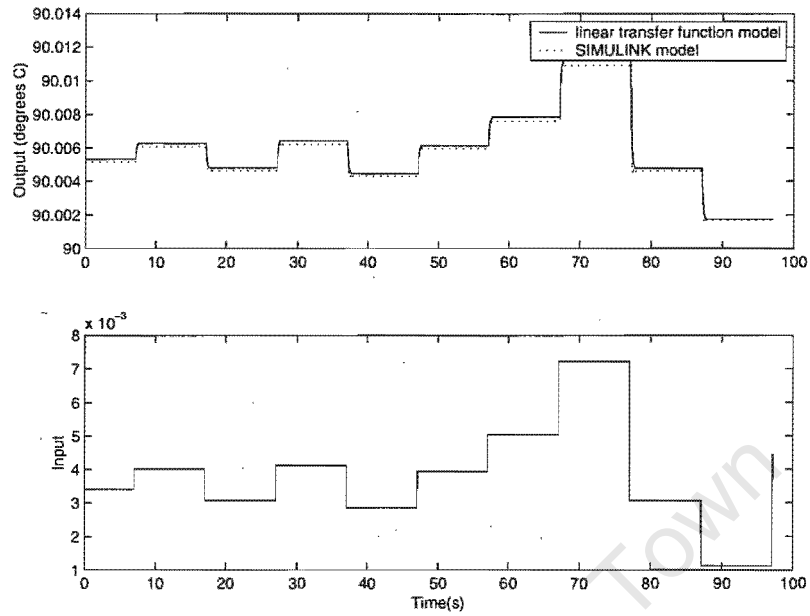

Figure B.5: Comparison of SIMULINK© and identified transfer function output from **input 3** and **output 1**

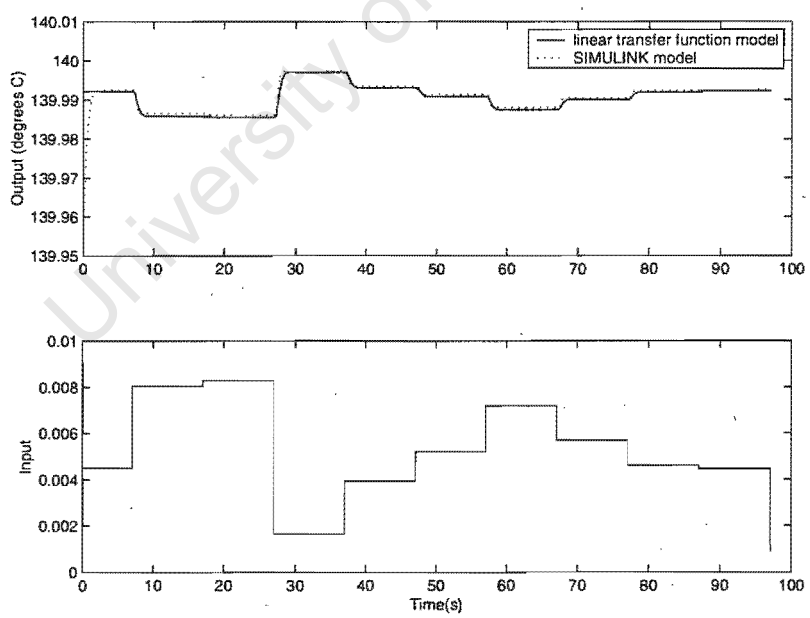

Figure B.6: Comparison of SIMULINK© and identified transfer function output from **input 3** and **output 2**

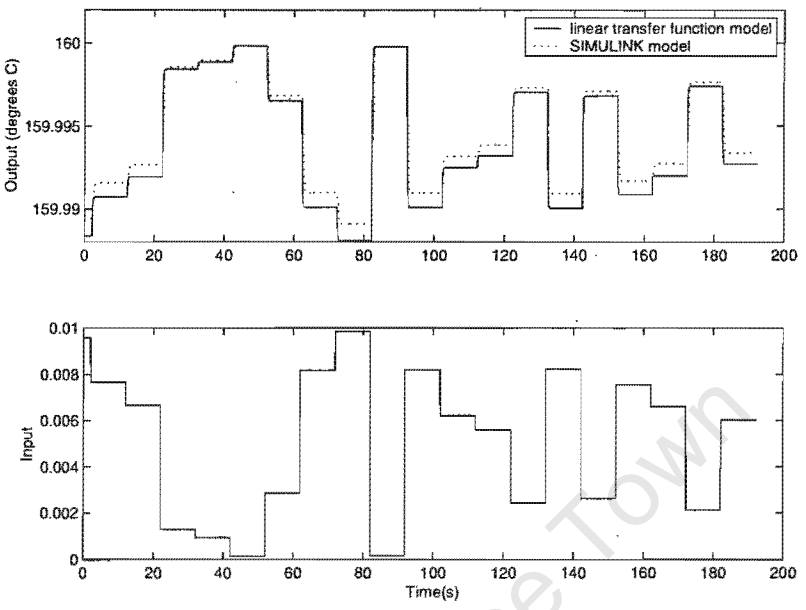

Figure B.7: Comparison of SIMULINK© and identified transfer function output from **input 3** and **output 3**

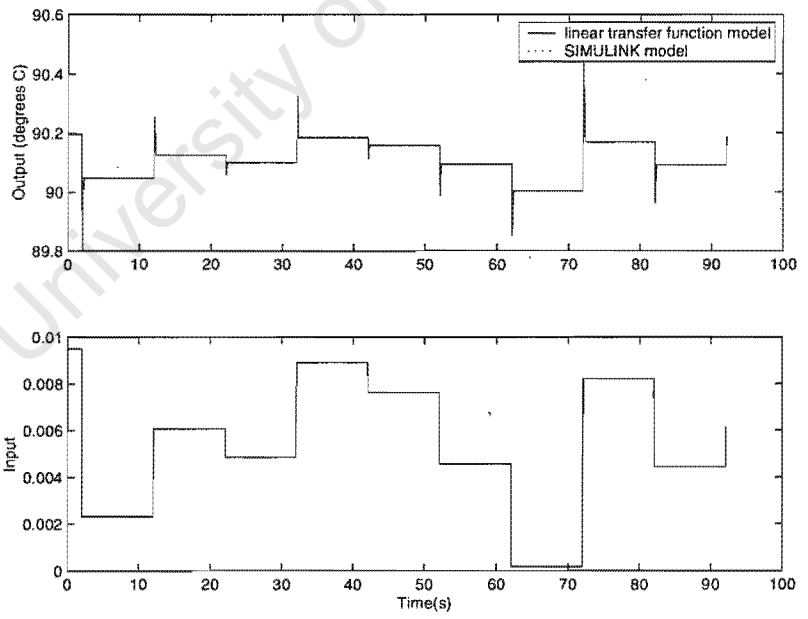

Figure B.8: Comparison of SIMULINK© and identified transfer function output from **input 4** and **output 1**

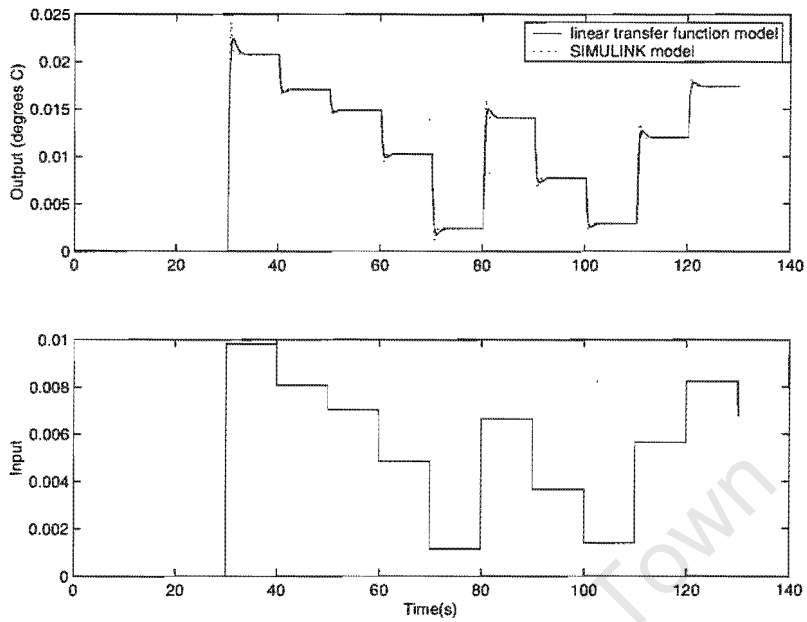

Figure B.9: Comparison of SIMULINK© and identified transfer function output from **input 1** and **output 2**

Identification graphs for operability analysis example 2, design B

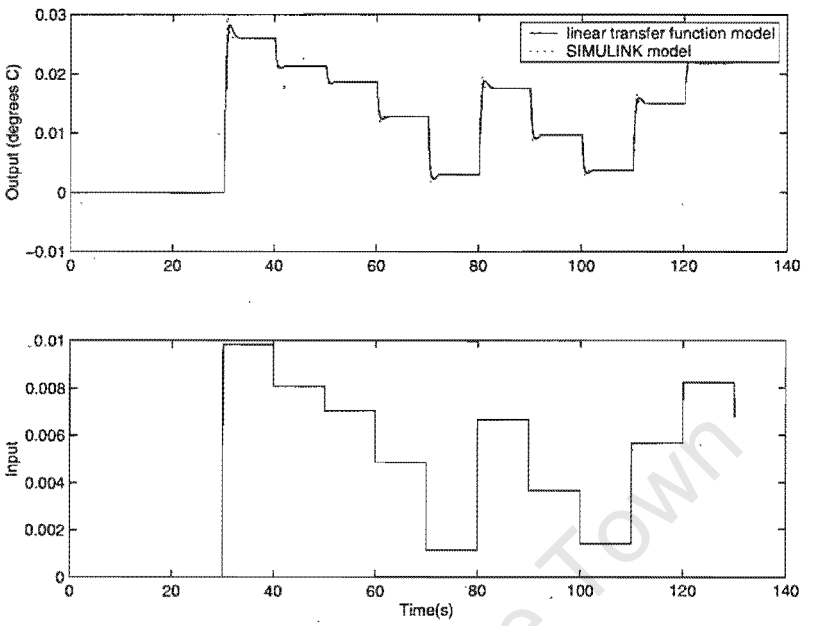

Figure B.10: Comparison of SIMULINK© and identified transfer function output from **input 1** and **output 3**

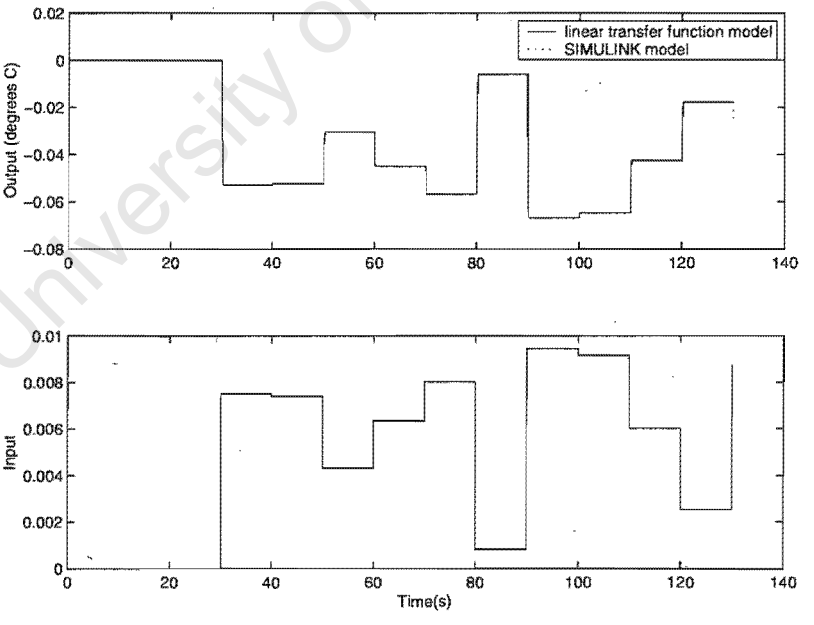

Figure B.11: Comparison of SIMULINK© and identified transfer function output from **input 1** and **output 4**

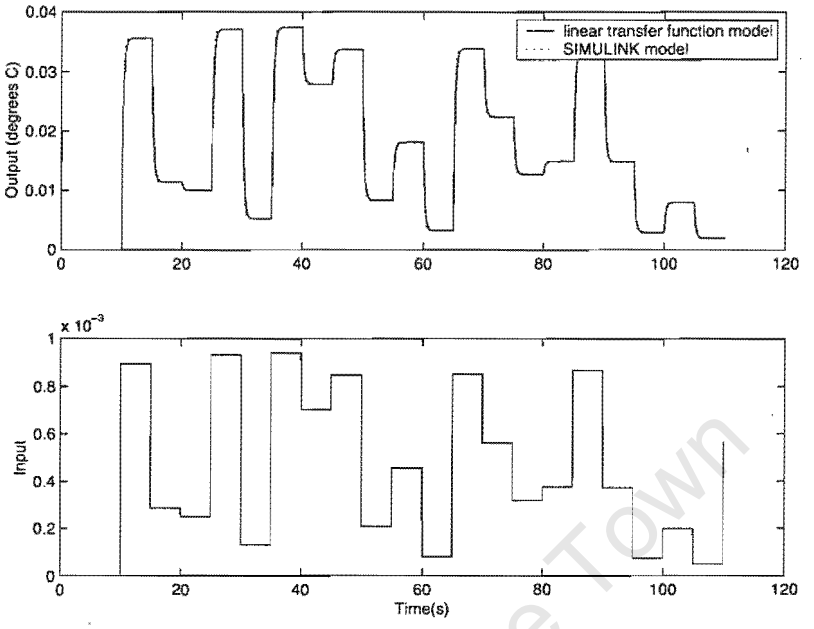

Figure B.12: Comparison of SIMULINK© and identified transfer function output from **input 2** and **output 2**

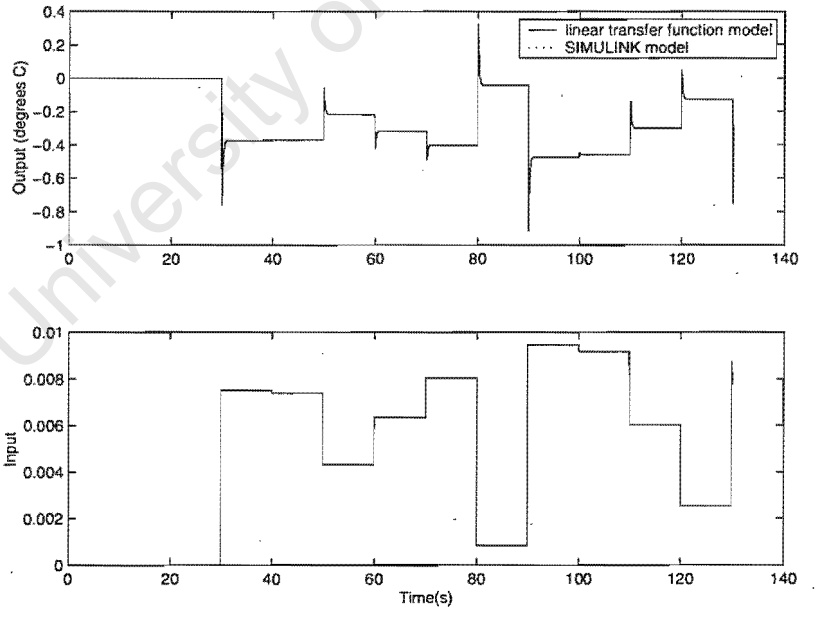

Figure B.13: Comparison of SIMULINK© and identified transfer function output from **input 2** and **output 3**

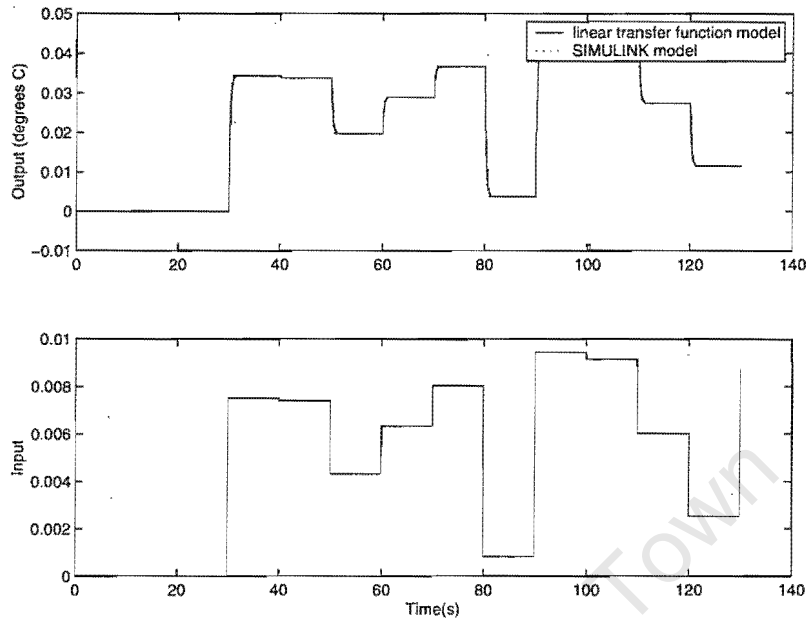

Figure B.14: Comparison of SIMULINK© and identified transfer function output from **input 3** and **output 1**

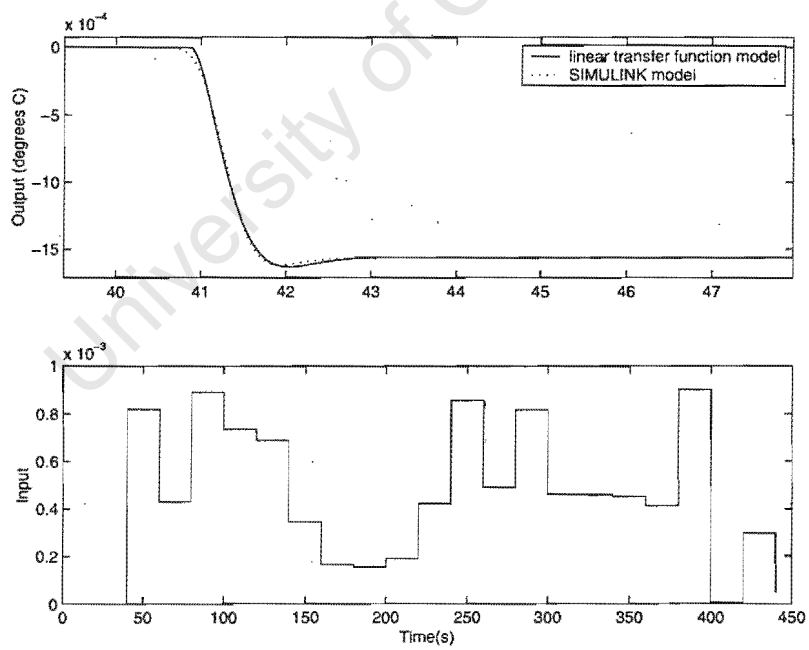

Figure B.15: Comparison of SIMULINK© and identified transfer function output from **input 3** and **output 2**. Here, just one section of the output trajectory has been zoomed so that the quality of the fit is seen more clearly.

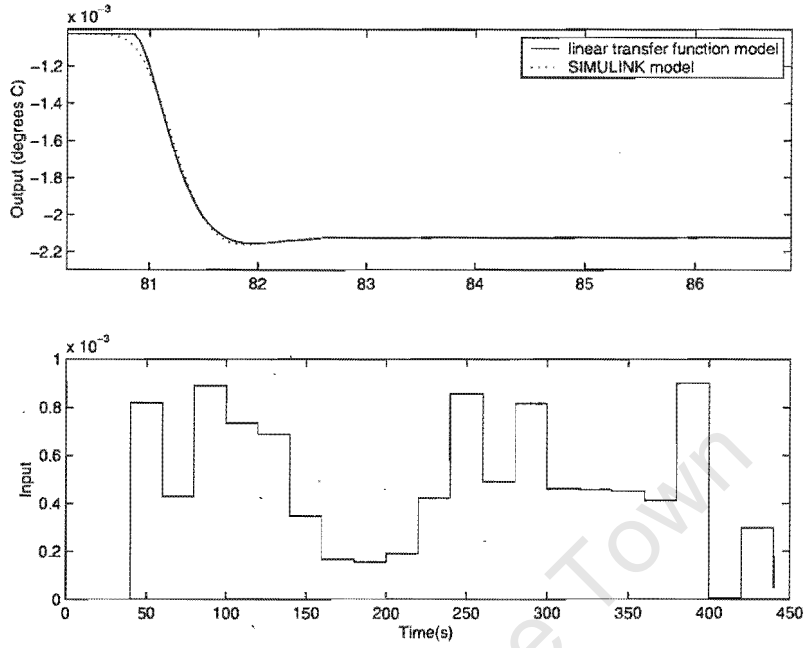

Figure B.16: Comparison of SIMULINK© and identified transfer function output from **input 3** and **output 3**

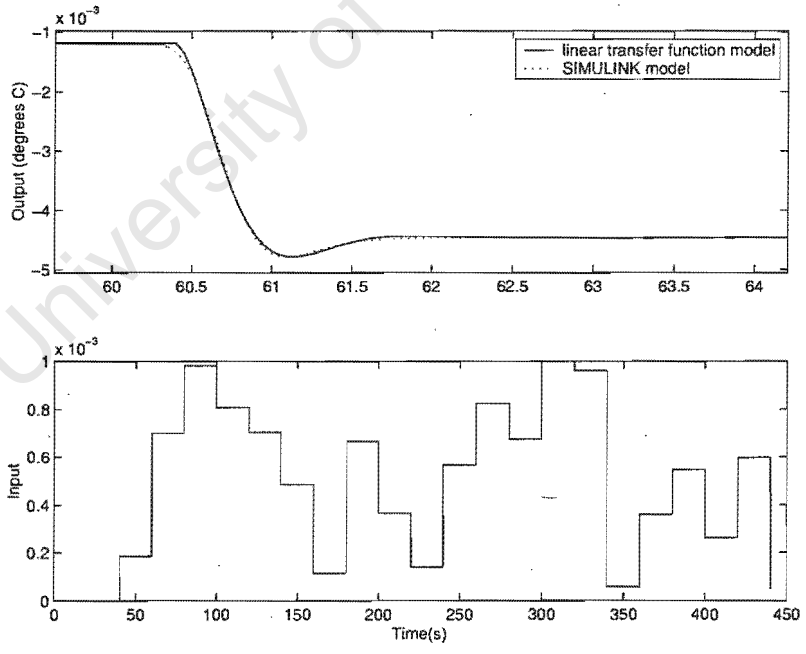

Figure B.17: Comparison of SIMULINK© and identified transfer function output from **input 3** and **output 4**

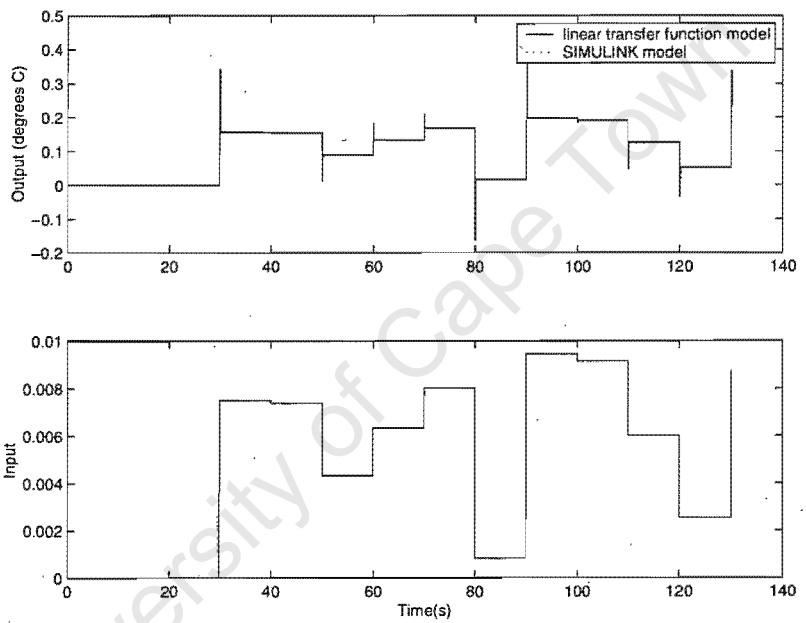

Figure B.18: Comparison of SIMULINK© and identified transfer function output from **input 4** and **output 1**

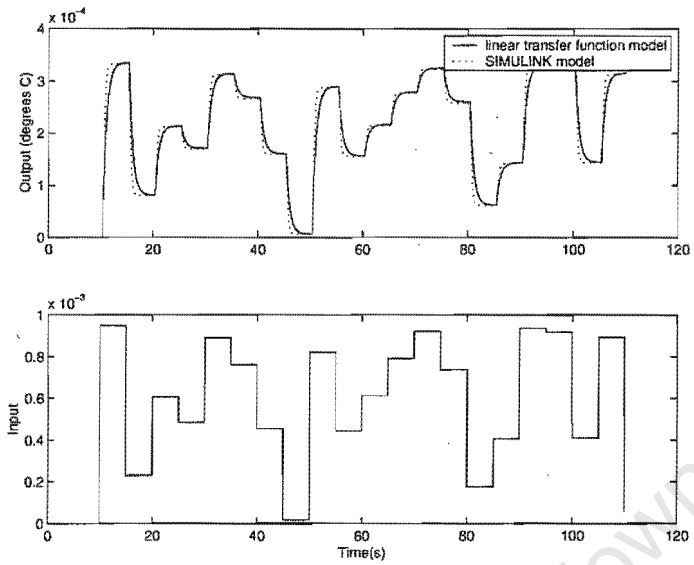

Figure B.19: Comparison of SIMULINK© and identified transfer function output from **disturbance 1** and **output 2**

B.1 Identification of the disturbance transfer function matrix

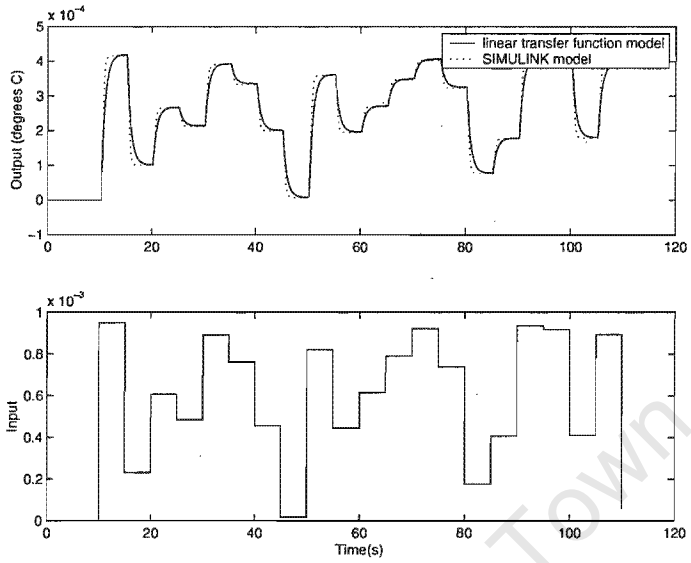

Figure B.20: Comparison of SIMULINK© and identified transfer function output from **disturbance 1** and **output 3**

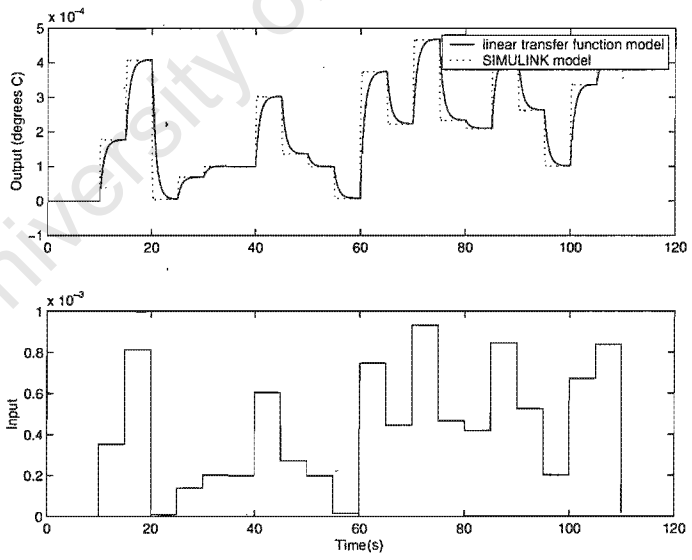

Figure B.21: Comparison of SIMULINK© and identified transfer function output from **disturbance 1** and **output 4**

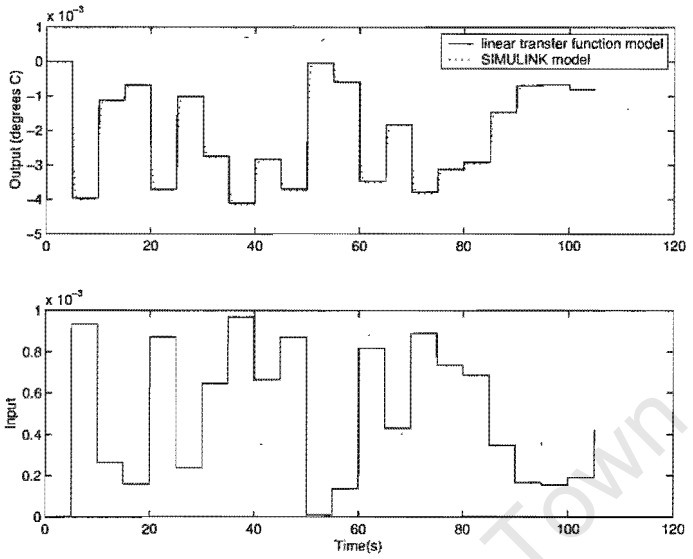

Figure B.22: Comparison of SIMULINK© and identified transfer function output from **disturbance 2** and **output 2**

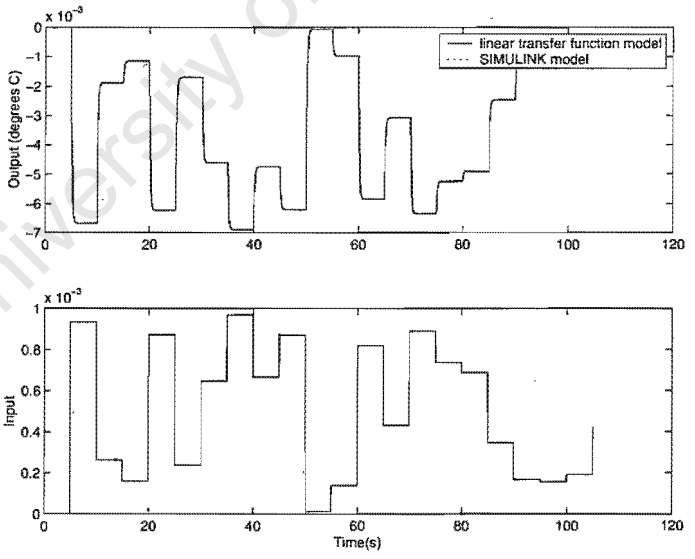

Figure B.23: Comparison of SIMULINK© and identified transfer function output from **disturbance 2** and **output 3**

Appendix C

Solution trajectories for dynamic operability example 2

C.1 Trajectories for design A

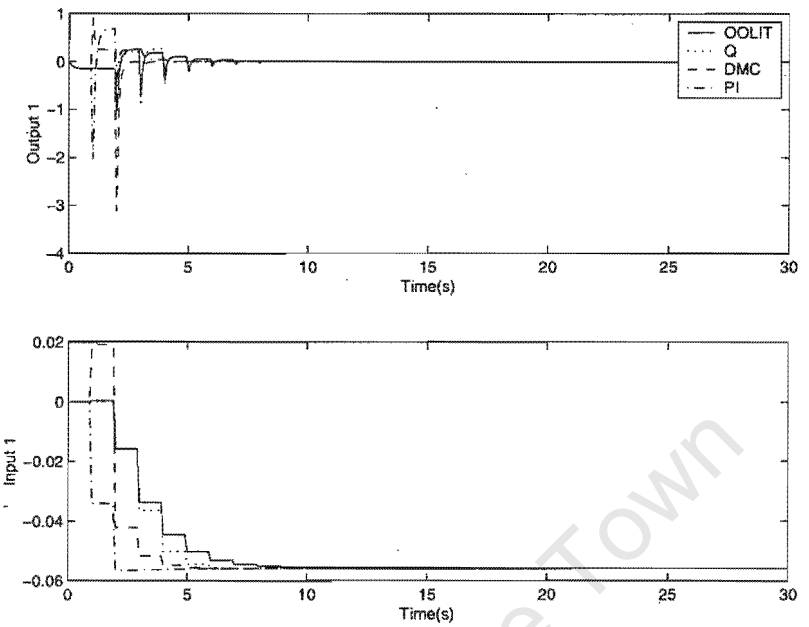

Figure C.1: Solution trajectories for design A: output and input 1 for dynamic operability example 2

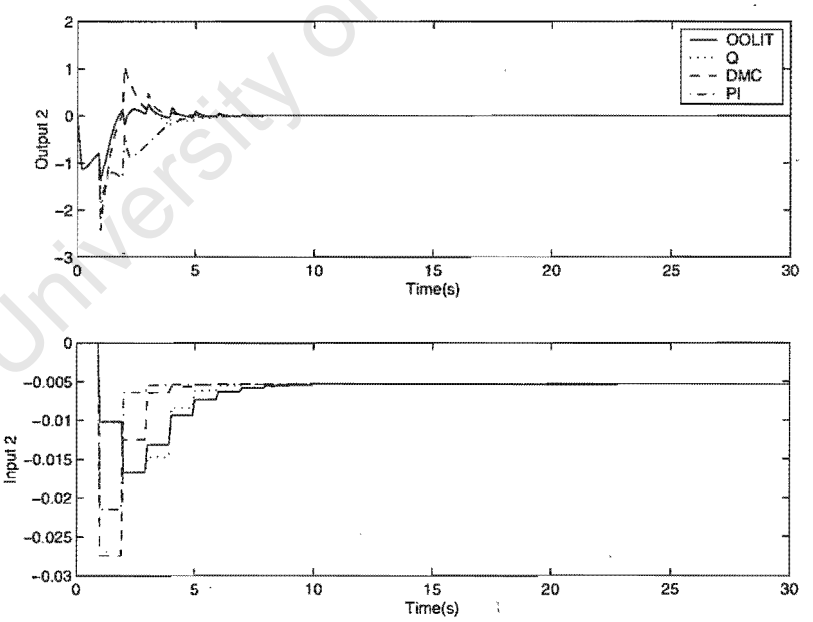

Figure C.2: Solution trajectories for design A: output and input 2 for dynamic operability example 2

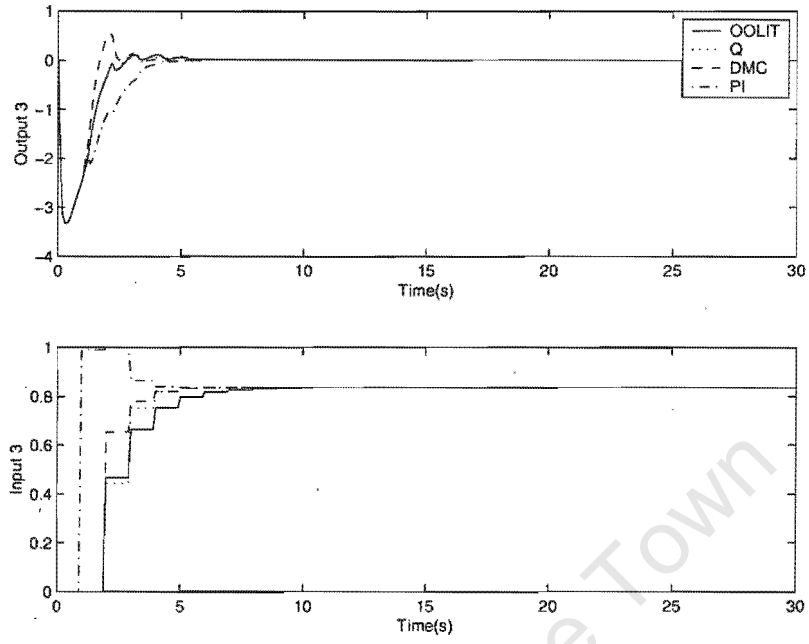

Figure C.3: Solution trajectories for design A: output and input 3 for dynamic operability example 2

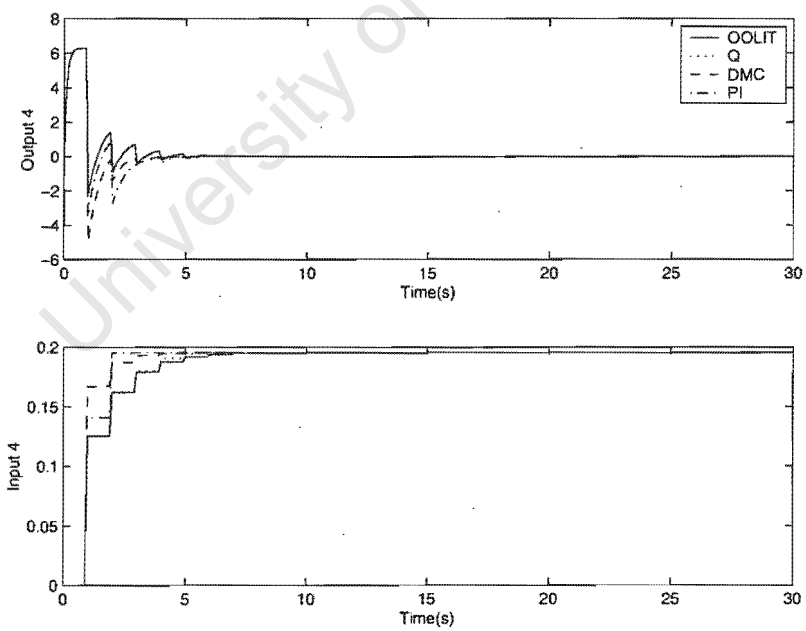

Figure C.4: Solution trajectories for design A: output and input 4 for dynamic operability example 2

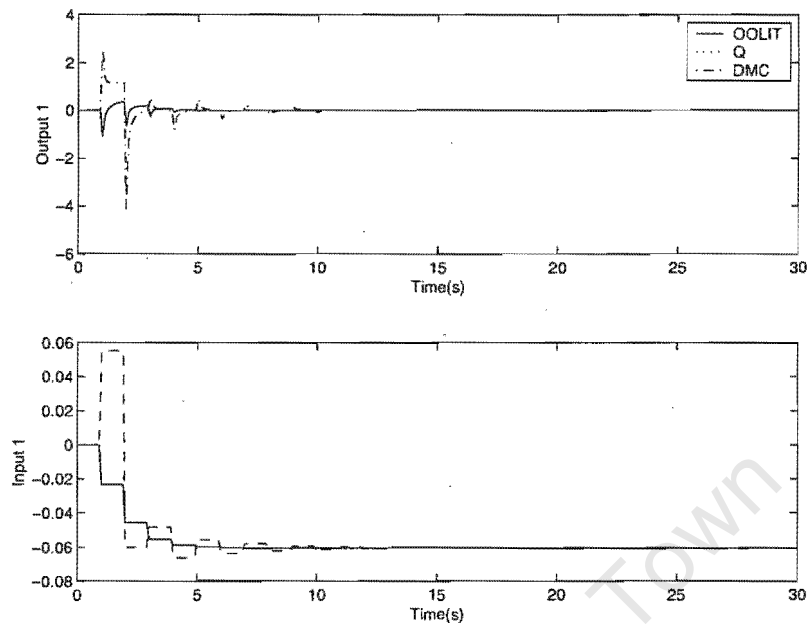

Figure C.5: Solution trajectories for design B: output and input 1 for dynamic operability example 2

C.2 Trajectories for design B

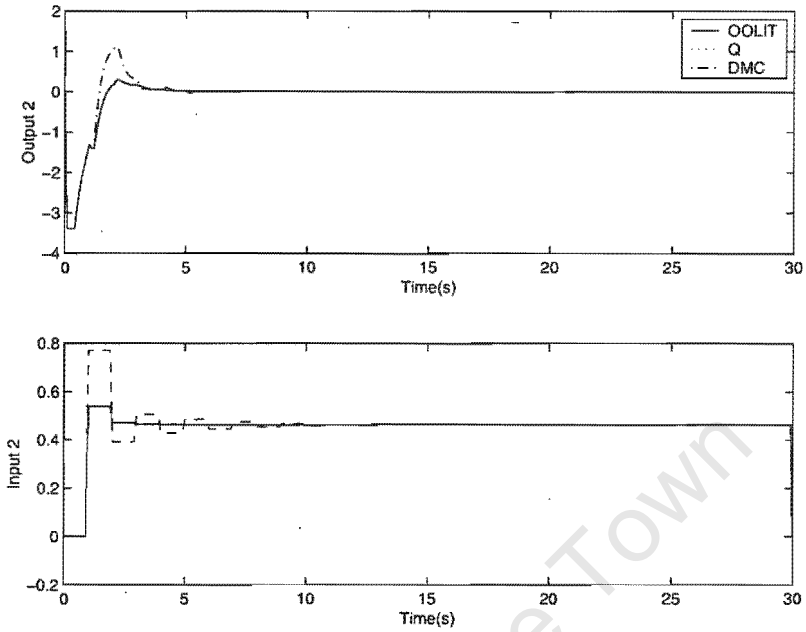

Figure C.6: Solution trajectories for design B: output and input 2 for dynamic operability example 2

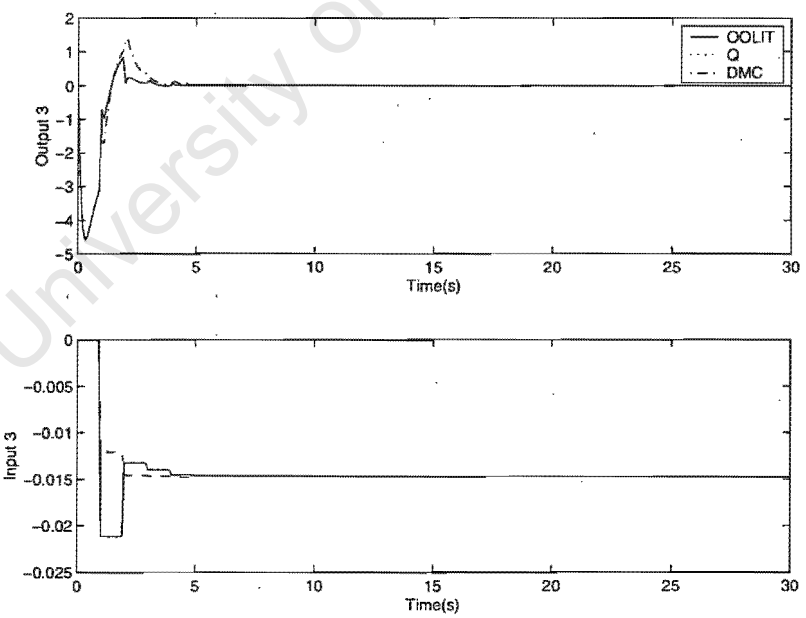

Figure C.7: Solution trajectories for design B: output and input 3 for dynamic operability example 2

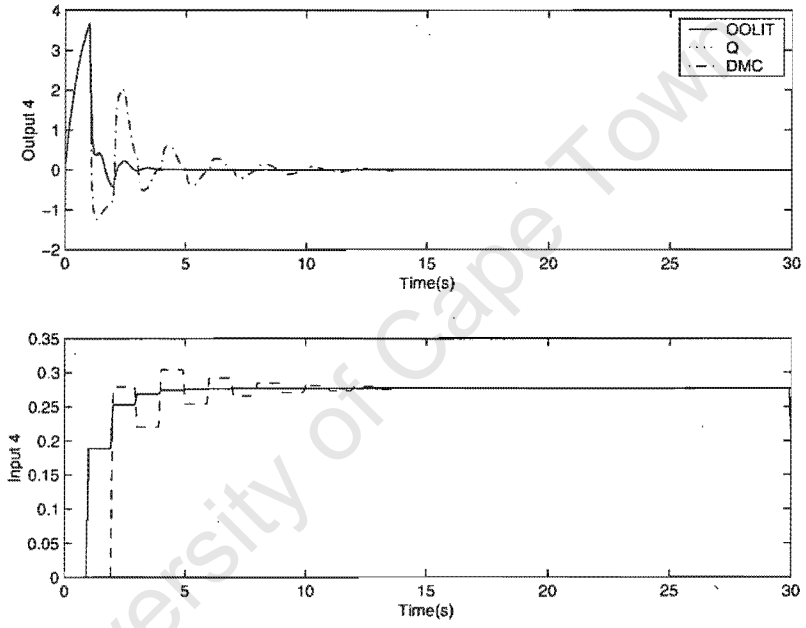

Figure C.8: Solution trajectories for design B: output and input 4 for dynamic operability example 2

Appendix D

Solution trajectories for dynamic operability example 3

D.1 Trajectories for design A

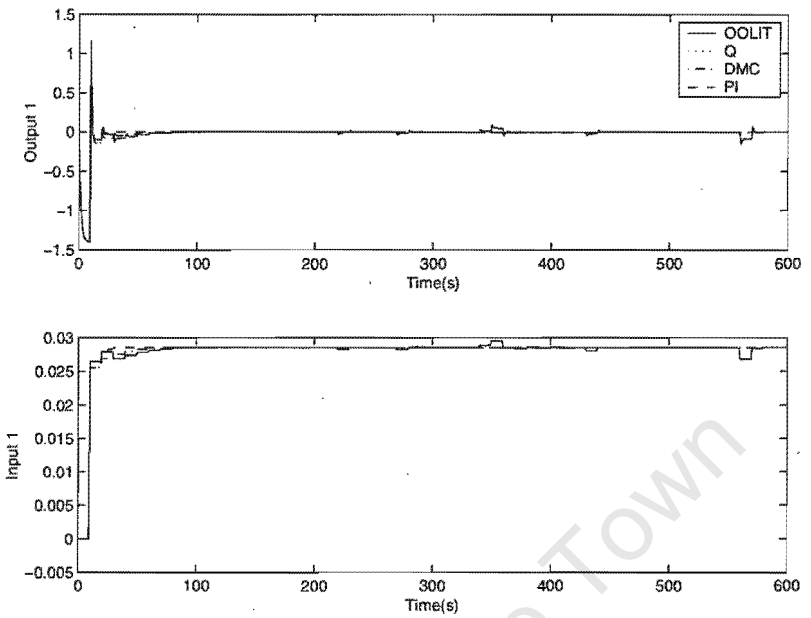

Figure D.1: Solution trajectories for design A: output and input 1 for dynamic operability example 3

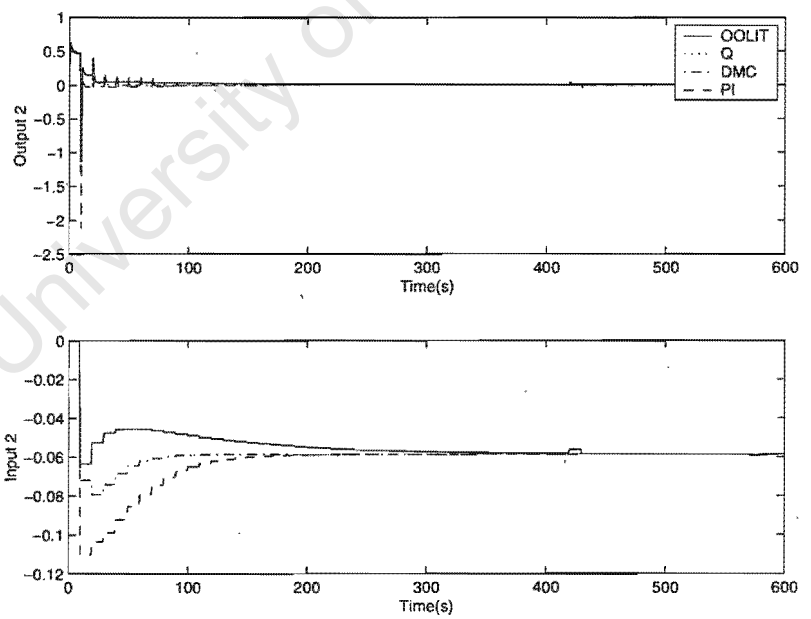

Figure D.2: Solution trajectories for design A: output and input 2 for dynamic operability example 3

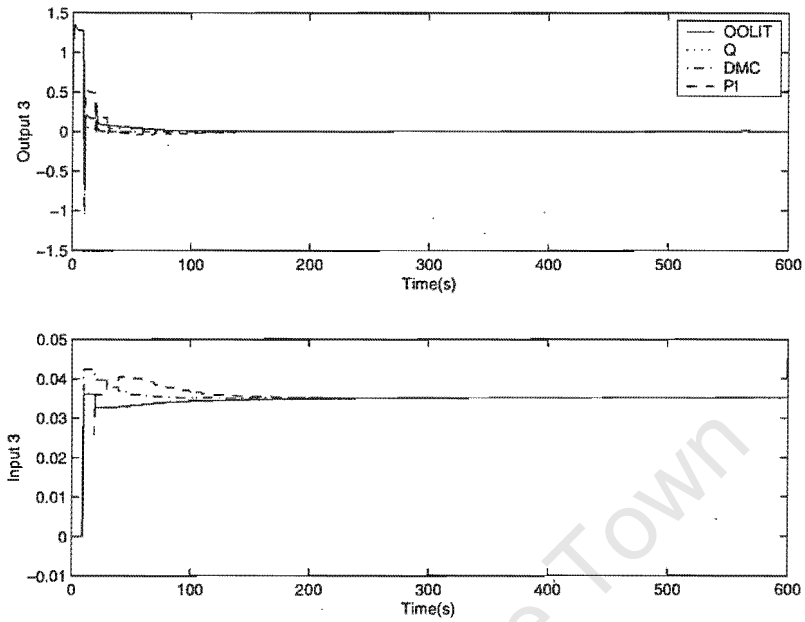

Figure D.3: Solution trajectories for design A: output and input 3 for dynamic operability example 3

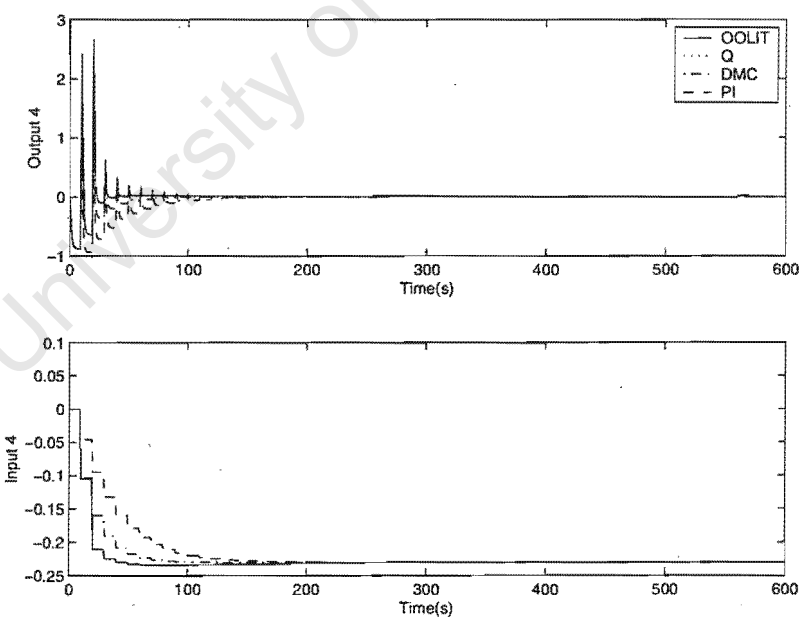

Figure D.4: Solution trajectories for design A: output and input 4 for dynamic operability example 3

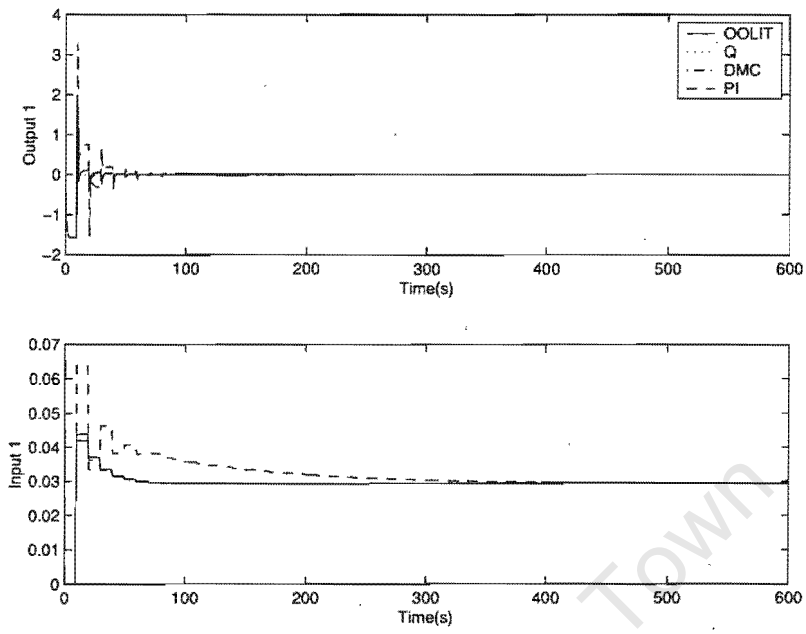

Figure D.5: Solution trajectories for design B: output and input 1 for dynamic operability example 3

D.2 Trajectories for design B

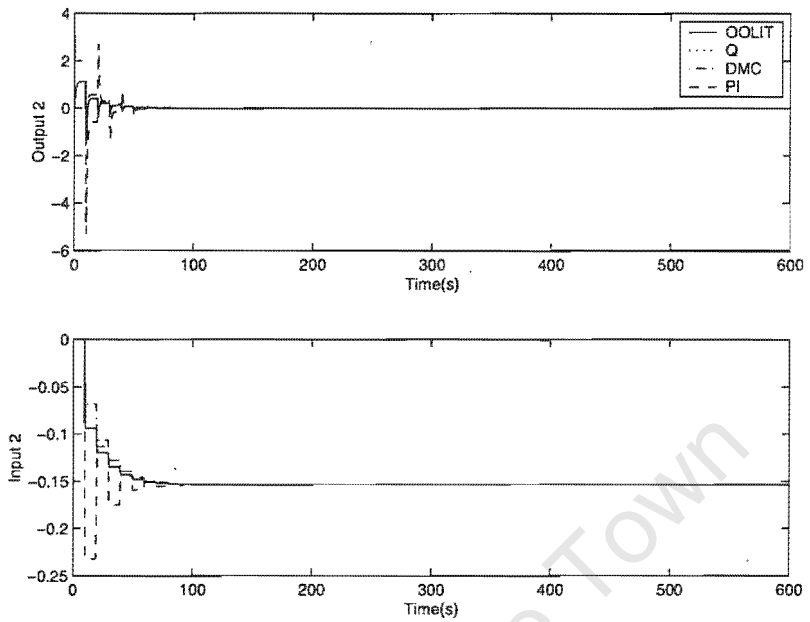

Figure D.6: Solution trajectories for design B: output and input 2 for dynamic operability example 3

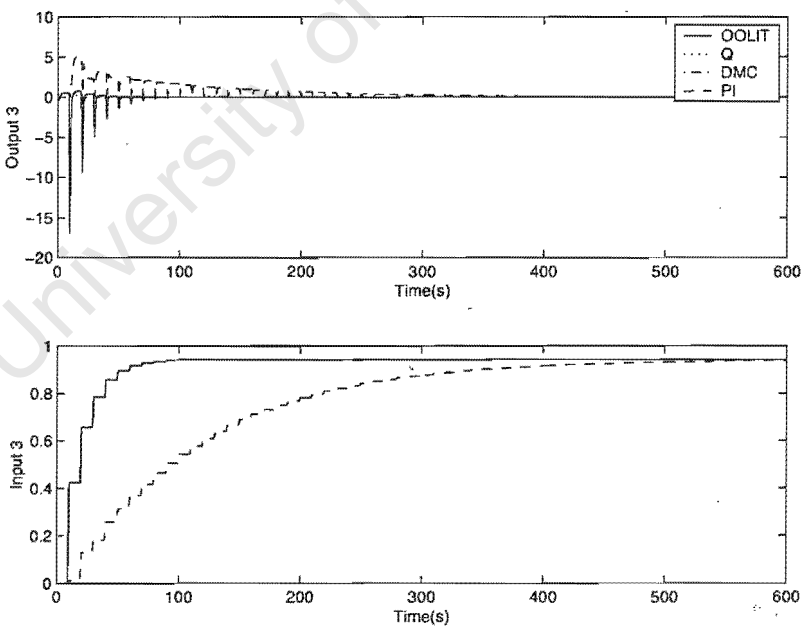

Figure D.7: Solution trajectories for design B: output and input 3 for dynamic operability example 3

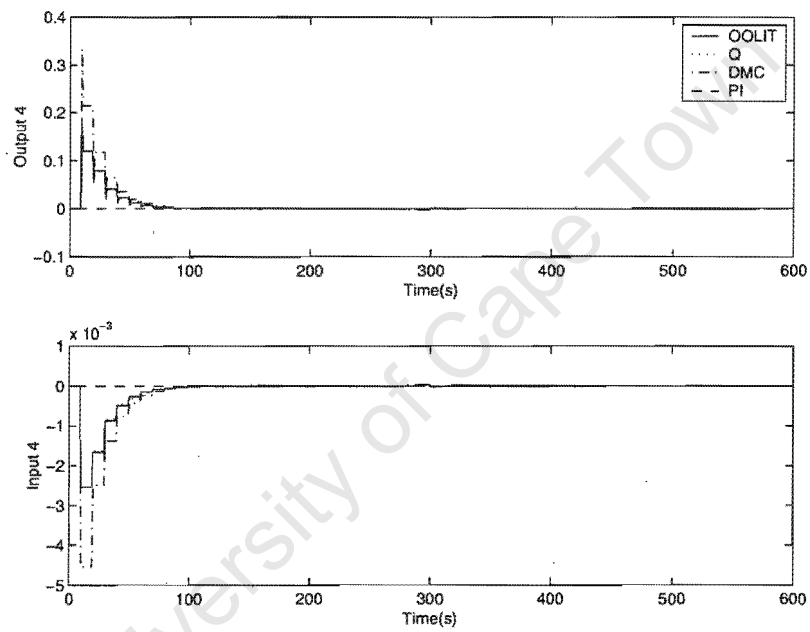

Figure D.8: Solution trajectories for design B: output and input 4 for dynamic operability example 3

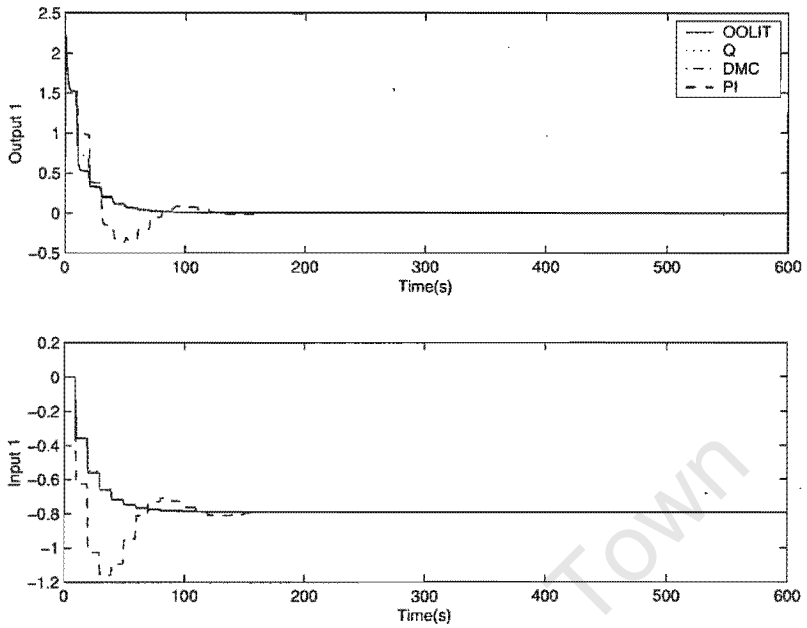

Figure D.9: Solution trajectories for design C: output and input 1 for dynamic operability example 3

D.3 Trajectories for design C

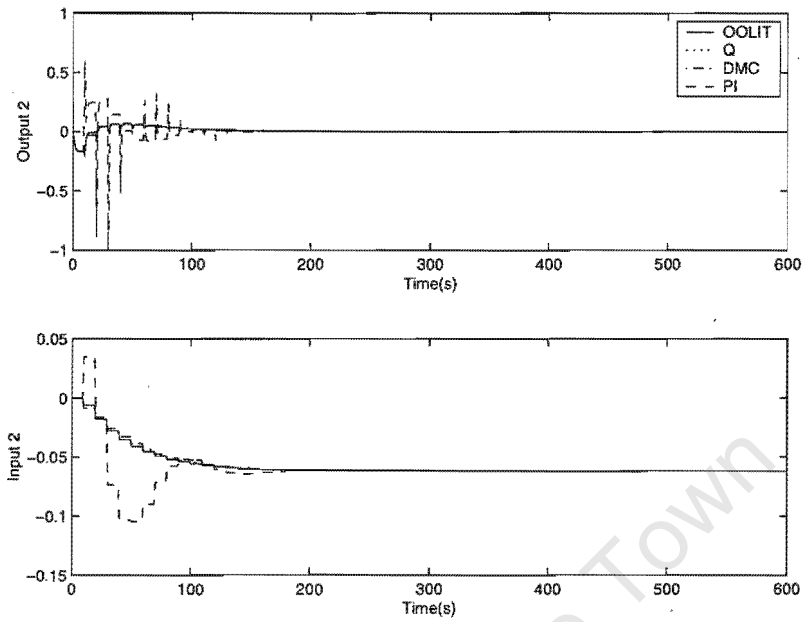

Figure D.10: Solution trajectories for design C: output and input 2 for dynamic operability example 3

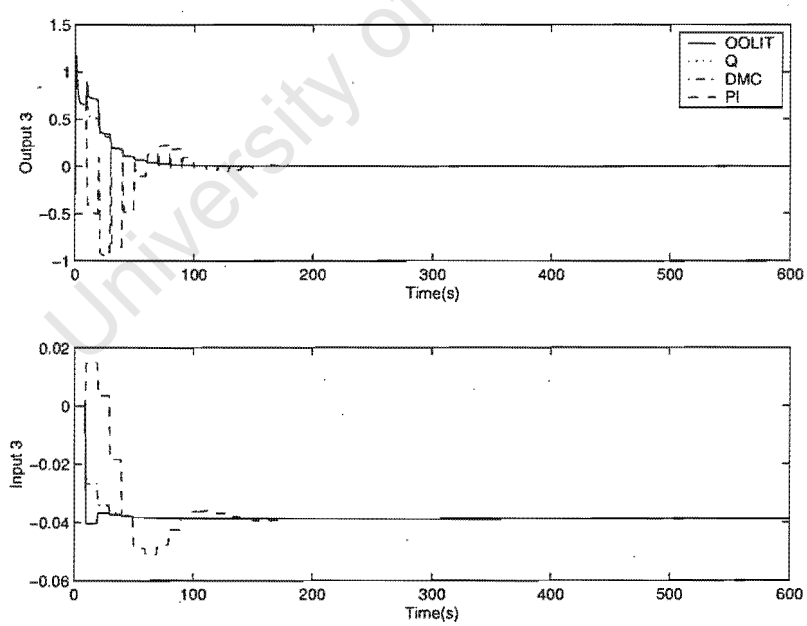

Figure D.11: Solution trajectories for design C: output and input 3 for dynamic operability example 3

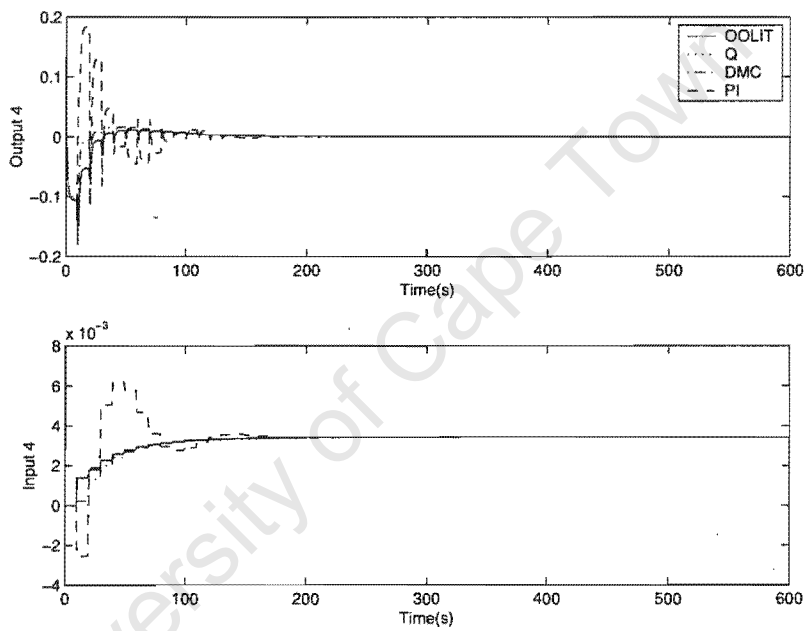

Figure D.12: Solution trajectories for design C: output and input 4 for dynamic operability example 3

Appendix E

Data and results for Operability assessment example 1

State-space linear model representations

The state-space model for design A in terms of matrices a, b, c and d, is

a =

	x1	x2	x3	x4	x5
x1	1.6654	0.33462	-0.33462	0.33462	-0.33462
x2	-2.2414	0.24144	1.7586	-1.7586	1.7586
x3	0.058054	-0.058054	-1.9419	3.9419	-3.9419
x4	1.1205	-1.1205	1.1205	-3.1205	5.1205
x5	-0.77628	0.77628	-0.77628	0.77628	-2.7763
x6	0.1738	-0.1738	0.1738	-0.1738	0.1738
x7	0	0	0	0	0
x8	0	0	0	0	0
x9	0	0	0	0	0
x10	0	0	0	0	0
x11	0	0	0	0	0
x12	0	0	0	0	0
x13	0	0	0	0	0
x14	0	0	0	0	0
x15	0	0	0	0	0
x16	0	0	0	0	0
x17	0	0	0	0	0
x18	0	0	0	0	0
x19	0	0	0	0	0
x20	0	0	0	0	0
x21	0	0	0	0	0

	x6	x7	x8	x9	x10
x1	0.33462	-0.33462	0	0	0
x2	-1.7586	1.7586	0	0	0
x3	3.9419	-3.9419	0	0	0
x4	-5.1205	5.1205	0	0	0
x5	4.7763	-4.7763	0	0	0

x6	-2.1738	4.1738	0	0	0
x7	0	-2	0	0	0
x8	0	0	1.713	0.28698	-0.28698
x9	0	0	-2.653	0.65301	1.347
x10	0	0	1.1657	-1.1657	-0.83429
x11	0	0	-0.22576	0.22576	-0.22576
x12	0	0	0	0	0
x13	0	0	0	0	0
x14	0	0	0	0	0
x15	0	0	0	0	0
x16	0	0	0	0	0
x17	0	0	0	0	0
x18	0	0	0	0	0
x19	0	0	0	0	0
x20	0	0	0	0	0
x21	0	0	0	0	0

	x11	x12	x13	x14	x15
x1	0	0	0	0	0
x2	0	0	0	0	0
x3	0	0	0	0	0
x4	0	0	0	0	0
x5	0	0	0	0	0
x6	0	0	0	0	0
x7	0	0	0	0	0
x8	0.28698	0	0	0	0
x9	-1.347	0	0	0	0
x10	2.8343	0	0	0	0
x11	-1.7742	0	0	0	0
x12	0	1.8236	0.17636	-0.17636	0.17636
x13	0	-3.0847	1.0847	0.91532	-0.91532
x14	0	1.8033	-1.8033	-0.19671	2.1967
x15	0	-0.64827	0.64827	-0.64827	-1.3517
x16	0	0.10597	-0.10597	0.10597	-0.10597
x17	0	0	0	0	0
x18	0	0	0	0	0
x19	0	0	0	0	0
x20	0	0	0	0	0
x21	0	0	0	0	0

	x16	x17	x18	x19	x20
x1	0	0	0	0	0
x2	0	0	0	0	0
x3	0	0	0	0	0
x4	0	0	0	0	0
x5	0	0	0	0	0
x6	0	0	0	0	0
x7	0	0	0	0	0
x8	0	0	0	0	0
x9	0	0	0	0	0
x10	0	0	0	0	0
x11	0	0	0	0	0

x12	-0.17636	0	0	0	0
x13	0.91532	0	0	0	0
x14	-2.1967	0	0	0	0
x15	3.3517	0	0	0	0
x16	-1.894	0	0	0	0
x17	0	0.31987	1.6801	0	0
x18	0	-0.37684	-1.6232	0	0
x19	0	0	0	0.69087	1.3091
x20	0	0	0	-0.71997	-1.28
x21	0	0	0	0	0

	x21
x1	0
x2	0
x3	0
x4	0
x5	0
x6	0
x7	0
x8	0
x9	0
x10	0
x11	0
x12	0
x13	0
x14	0
x15	0
x16	0
x17	0
x18	0
x19	0
x20	0
x21	-10000

b =

	u1	u2	u3
x1	4.8271e-007	0	0
x2	-2.5368e-006	0	0
x3	5.5147e-006	0	0
x4	-7.2141e-006	0	0
x5	6.7172e-006	0	0
x6	-5.8477e-006	0	0
x7	5.5989e-006	0	0
x8	0	-2.5051e-005	0
x9	0	-3.5494e-005	0
x10	0	0.00026155	0
x11	0	0.0021181	0
x12	0.0015792	0	0
x13	-0.002877	0	0
x14	0.0035143	0	0
x15	-0.02311	0	0

x16	-0.0001836	0	0
x17	0	0	1.7496
x18	0	0	2.5867
x19	2.6108	0	0
x20	1.4358	0	0
x21	0	1024	0

c =

	x1	x2	x3	x4	x5
y1	0.16731	-0.16731	0.16731	-0.16731	0.16731
y2	0	0	0	0	0
y3	0	0	0	0	0

	x6	x7	x8	x9	x10
y1	-0.16731	0.16731	0.14349	-0.14349	0.14349
y2	0	0	0	0	0
y3	0	0	0	0	0

	x11	x12	x13	x14	x15
y1	-0.14349	0	0	0	0
y2	0	0	0	0	0
y3	0	0.08818	-0.08818	0.08818	-0.08818

	x16	x17	x18	x19	x20
y1	0	0	0	0	0
y2	0	0	0	0.65457	-0.65457
y3	0.08818	0.84007	-0.84007	0	0

	x21
y1	0
y2	-1302.7
y3	0

d =

	u1	u2	u3
y1	-2.4135e-007	1.2525e-005	0
y2	-1.3054	0	0
y3	-0.00015277	0	-0.87479

The state-space representation for model B is given as

a =

	x1	x2	x3	x4	x5
x1	0.31987	1.6801	0	0	0
x2	-0.37684	-1.6232	0	0	0
x3	0	0	-10000	0	0
x4	0	0	0	0.86225	1.1377
x5	0	0	0	-0.87404	-1.126
x6	0	0	0	0.0005096	-0.0005096
x7	0	0	0	0	0

	x6	x7
x1	0	0
x2	0	0
x3	0	0
x4	-1.1377	0
x5	3.126	0
x6	-1.9995	0
x7	0	-0.054714

b =

	u1	u2	u3
x1	0	0	1.7496
x2	0	0	2.5867
x3	0	1024	0
x4	159.07	0	0
x5	-278.77	0	0
x6	122.72	0	0
x7	0	0.46207	0

c =

	x1	x2	x3	x4	x5
y1	0	0	0	0.56887	-0.56887
y2	0	0	-1302.7	0	0
y3	0.84007	-0.84007	0	0	0

	x6	x7
y1	0.56887	1.0274
y2	0	0
y3	0	0

d =

	u1	u2	u3
y1	-79.534	-0.23103	0
y2	0	0	0
y3	0	0	-0.87479

The state-space representation for the disturbance transfer function is given as

a =

	x1	x2	x3	x4	x5
x1	1.7333	0.26666	-0.26666	0.26666	0
x2	-2.7009	0.70094	1.2991	-1.2991	0
x3	1.2018	-1.2018	-0.79817	2.7982	0
x4	-0.23423	0.23423	-0.23423	-1.7658	0
x5	0	0	0	0	0.63152
x6	0	0	0	0	-0.47903
x7	0	0	0	0	-0.15901
x8	0	0	0	0	0

x9	0	0	0	0	0
x10	0	0	0	0	0
x11	0	0	0	0	0
x12	0	0	0	0	0
x13	0	0	0	0	0
x14	0	0	0	0	0
x15	0	0	0	0	0

	x6	x7	x8	x9	x10
x1	0	0	0	0	0
x2	0	0	0	0	0
x3	0	0	0	0	0
x4	0	0	0	0	0
x5	1.3685	-1.3685	0	0	0
x6	-1.521	3.521	0	0	0
x7	0.15901	-2.159	0	0	0
x8	0	0	1.206	0.79404	-0.79404
x9	0	0	-1.5237	-0.47635	2.4763
x10	0	0	0.36451	-0.36451	-1.6355
x11	0	0	-0.048538	0.048538	-0.048538
x12	0	0	0	0	0
x13	0	0	0	0	0
x14	0	0	0	0	0
x15	0	0	0	0	0

	x11	x12	x13	x14	x15
x1	0	0	0	0	0
x2	0	0	0	0	0
x3	0	0	0	0	0
x4	0	0	0	0	0
x5	0	0	0	0	0
x6	0	0	0	0	0
x7	0	0	0	0	0
x8	0.79404	0	0	0	0
x9	-2.4763	0	0	0	0
x10	3.6355	0	0	0	0
x11	-1.9515	0	0	0	0
x12	0	-0.031537	2.0315	0	0
x13	0	0	-2	0	0
x14	0	0	0	0.93433	1.0657
x15	0	0	0	-0.93642	-1.0636

b =

	u1	u2
x1	-1.3069e-008	0
x2	6.3239e-008	0
x3	-1.3671e-007	0
x4	1.8413e-007	0
x5	0	-0.092551
x6	0	0.23814
x7	0	-0.28136

x8	0.065698	0
x9	-0.061464	0
x10	-0.0057091	0
x11	0.0040212	0
x12	0	-0.008599
x13	0	0.0038995
x14	-0.00031654	0
x15	0.00079379	0

c =

	x1	x2	x3	x4	x5
y1	0.13333	-0.13333	0.13333	-0.13333	0.68424
y2	0	0	0	0	0
y3	0	0	0	0	0

	x6	x7	x8	x9	x10
y1	-0.68424	0.68424	0	0	0
y2	0	0	0.39702	-0.39702	0.39702
y3	0	0	0	0	0

	x11	x12	x13	x14	x15
y1	0	0	0	0	0
y2	-0.39702	1.0158	-1.0158	0	0
y3	0	0	0	0.53284	-0.53284

d =

	u1	u2
y1	6.5345e-009	0.046279
y2	-0.032892	0.0044348
y3	0.00015827	0

Relative Gain Arrays

The RGA for design A is

```
          0.5899    0.4101    0
RGA_A =   0.4101    0.5899    0
          0         0         1.0000
```

The RGA for design B is

```
          1.0000    0         0
RGA_B =   0         1.0000    0
          0         0         1.0000
```

Right-half-plane zeros

The zeros for design A are:

```
  -29693
   6.6403 +     2.6025i
   6.6403 -     2.6025i
  -2.5726 +     2.2715i
  -2.5726 -     2.2715i
  -4.1072
  -1.3906
   1.3757
        2 + 0.00010614i
        2 - 0.00010614i
   2.0001
 -0.13946 +    0.61697i
 -0.13946 -    0.61697i
-0.022591 +    0.07793i
-0.022591 -    0.07793i
-0.096153 +    0.14274i
-0.096153 -    0.14274i
-0.028806 +   0.034315i
-0.028806 -   0.034315i
-0.040352
-0.062914
```

The zeros for design B are

```
 -0.12688 +   0.067344i
 -0.12688 -   0.067344i
  -4.1072
-0.054714
        2
        2
```

Appendix F

MATLAB code

All(!) of the MATLAB© code programmed and subsequently used in this project is provided here, although a software copy may be obtained from author or the Process Control group in the Chemical Engineering Department of the University of Cape Town.

The important sections of this code are described in the body of this thesis, mostly in Chapter 5 on page 59. For scripts and functions which are not described there, there is usually a description in the header of the code itself.

NOTE: *To use this code it is required that the user have access to MATLAB© version 5.3 (Release 11),* ***as well as*** *SIMULINK© version 3.0, the Control Toolbox© , the MPCtools Toolbox© and most importantly, the Optimization Toolbox© .*

F.1 The functions for dynamic operability of HENs

F.1.1 The top level function for dynamic operability of HENs

filename: HENdynopt.m

```
function Result = HENdynopt(filename,cmethod);

% Dynamic Operability studies
```

```
%
% Caleb Hattingh 28 April 1999
%           revised 2 February 2000
%              "    7 March    "
%*******************************************************

% CHOOSE BETWEEN DYN OP METHOD OR OTHER
if cmethod <= 3

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              %                                   %
              % ONE OF THE DYNAMIC OPERABILITY    %
              % METHODS                           %
              %                                   %
              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   Result = optmethods(filename,cmethod);
elseif cmethod == 4

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              %                                   %
              % SIMULATE A NONLINEAR SIMULINK     %
              % MODEL                             %
              %                                   %
              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   Result = HENnonsim(filename);
elseif cmethod == 5

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              %                                   %
              % THE FLEXIBILITY OPTIMIZATION      %
              %                                   %
              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


   % This calls the flexibility program
   Result = flexopt(filename);
elseif cmethod == 6

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              %                                   %
              % THE IDENTIFICATION OF HEN MODEL   %
              %                                   %
              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


   % This calls the flexibility program
   Result = idenG(filename);
else

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
                %                                    %
                % UNDEFINED OPTION FOR CMETHOD       %
                %                                    %
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % This is called if cmethod is not currently identifiable.
   disp('Only methods 0-5 are currently supported.');
   disp(' ');
   disp('type');
   disp(' ');
   disp('    >> help HENdynopt');
   disp(' ');
   disp('for details.');
   disp('Aborting.                                  ');
   return

end
```

F.1.2 Function that sets up and calls one of the four operability assessment optimizations

filename: optmethods.m

```
function Results = optmethods(filename,cmethod);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% This does the dynamic operability optimizations
%
% It is seperate because it looks neater, and the functionality
% therefore much improved.  We can now hanlde other cases of
% cmethod outside this file.
%
% Caleb Hattingh (c) 2000
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% USER ENVIRONMENT SETTINGS
warning off

%-----------------------------------------------------------
% INITIALIZE ALL VARIABLES
run(filename);

%-----------------------------------------------------------
% PRELIMINARY SET-UP
t = [0:delt:tfinal]';       % Make control move timestep vector
newt = [0:deltsim:tfinal]'; % Make simulation timestep vector
n = length(t);              % Number of control moves
```

```
nn = length(newt);          % Number of simulation points
[rows,cols] = size(G);      % Size of Process Model
[rowsd,colsd] = size(Gd);   % Size of Disturbance Process Model
W = eye(rows,cols);         % Output weighting matrix
ssinputs = ssinputvalues(G,Gd,ysp,d)  % Steady state inputs
                                      % under disturbance

%------------------------------------------------------------
% INPUT CONSTRAINTS + ERROR CHECKING
ll = [];
ul = [];
[rlb,clb] = size(lowbnd');
[rub,cub] = size(uppbnd');
if (clb - cols) == 0 & (cub - cols) == 0
   for p = 1:cols
      ll = [ll lowbnd(p)];
      ul = [ul uppbnd(p)];
   end
   ll = ones(n,1)*ll;
   ul = ones(n,1)*ul;
   disp('Manipulated variable bounds assembled');
else
   disp('Dimensions of user defined bounds INCORRECT.');
   disp('Use appropriate "Inf" statements if no bounds present');
   disp('Aborting.                                    ');
   return
end

%------------------------------------------------------------
% THIS IS WHERE WE ENTER THE SECTION THAT SELECTS THE USER'S
% CHOICE OF PARTICULAR OPERABILITY ASSESSMENT METHOD OR
% FUNCTION.

% IT IS FAIRLY LONG AND WILL TAKE A WHILE TO UNDERSTAND
% WELL.

%------------------------------------------------------------
% INITIAL GUESS + ERROR CHECKING
if cmethod == 0

            % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %                                      %
            % PI control - square systems only     %
            %                                      %
            % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   disp('Method = 0: PI control.');

   if (cols-rows)~=0 % If system is not square, then...
      disp('May only use square systems!');
      disp('Program ended.');
      return         % STOP the program
```

```
   else
      % Do nothing
   end

   if isempty(initguess.pi) == 1  % If initial guess not supplied, then...
      u = [0*ones(rows,1)
         10*ones(rows,1)
         0*ones(rows,1)
         10*ones(rows,1)
         0*ones(rows,1)
         10*ones(rows,1)
         0*ones(rows,1)
         10*ones(rows,1)]; % ... make one.
   else                              % Otherwise ...
      [rig,cig] = size(initguess.pi);
      if rig == 2*cols & cig == 1 % ... check that dimensions are correct
         disp('Dimensions of user-defined initial guess are correct.');
         u = initguess.pi;
      else
         disp('Dimensions of user defined initial guess INCORRECT.');
         disp('Aborting.                                  ');
         return
      end
   end

   % These are needed to use the generalized feedback framework
   PIprops.T1 = makeT1(Gd,tfinal,delt,t,n);
   PIprops.T2 = makeT2(G,tfinal,delt,t,n);
   PIprops.T3 = makeT3(Gd,tfinal,delt,t,n);

   % These are bounds on the decision variables

%   ulb = PIprops.ulb*ones(length(u),1);
   ulb = [PIprops.ulb*ones(length(u)/2,1);1e-4*ones(length(u)/2,1)];
%   ulb = [-inf 0 -inf 0 -inf 0]';
   uub = PIprops.uub*ones(length(u),1);

% * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

elseif cmethod == 1

               % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
               %                                     %
               % DMC control - may be non-square     %
               %                                     %
               %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   disp('Method = 1: DMC control.');
   if isempty(initguess.dmc) == 1
      u = 1*ones(cols,1);
   else
      [rig,cig] = size(initguess.dmc);
```

```
   if rig == cols & cig == 1
      disp('Dimensions of user-defined initial guess are correct.');
      u = initguess.dmc;
   else
      disp('Dimensions of user defined initial guess INCORRECT.');
      disp('Aborting.                                    ');
      return
   end
end
sis = 0;
if sis == 1
   g11=poly2tfd(G(1,1).num{:,:},G(1,1).den{:,:},0,G.ioDelayMatrix(1,1));
      plant=tfd2step(tfinal,delt,1,g11);

   gd11=poly2tfd(Gd(1,1).num{:,:},Gd(1,1).den{:,:},0,Gd.ioDelayMatrix(1,1));

   dplant=tfd2step(tfinal,delt,1,gd11);

   DMCprops.plant = plant;   % Save them
   DMCprops.dplant = dplant;
   clear plant;              % Clear spare variables
   clear dplant;
else
% Lets try the tfd route
g11=poly2tfd(G(1,1).num{:,:},G(1,1).den{:,:},0,G.ioDelayMatrix(1,1));
g21=poly2tfd(G(2,1).num{:,:},G(2,1).den{:,:},0,G.ioDelayMatrix(2,1));
g31=poly2tfd(G(3,1).num{:,:},G(3,1).den{:,:},0,G.ioDelayMatrix(3,1));
g41=poly2tfd(G(4,1).num{:,:},G(4,1).den{:,:},0,G.ioDelayMatrix(4,1));

g12=poly2tfd(G(1,2).num{:,:},G(1,2).den{:,:},0,G.ioDelayMatrix(1,2));
g22=poly2tfd(G(2,2).num{:,:},G(2,2).den{:,:},0,G.ioDelayMatrix(2,2));
g32=poly2tfd(G(3,2).num{:,:},G(3,2).den{:,:},0,G.ioDelayMatrix(3,2));
g42=poly2tfd(G(4,2).num{:,:},G(4,2).den{:,:},0,G.ioDelayMatrix(4,2));

g13=poly2tfd(G(1,3).num{:,:},G(1,3).den{:,:},0,G.ioDelayMatrix(1,3));
g23=poly2tfd(G(2,3).num{:,:},G(2,3).den{:,:},0,G.ioDelayMatrix(2,3));
g33=poly2tfd(G(3,3).num{:,:},G(3,3).den{:,:},0,G.ioDelayMatrix(3,3));
g43=poly2tfd(G(4,3).num{:,:},G(4,3).den{:,:},0,G.ioDelayMatrix(4,3));

g14=poly2tfd(G(1,4).num{:,:},G(1,4).den{:,:},0,G.ioDelayMatrix(1,4));
g24=poly2tfd(G(2,4).num{:,:},G(2,4).den{:,:},0,G.ioDelayMatrix(2,4));
g34=poly2tfd(G(3,4).num{:,:},G(3,4).den{:,:},0,G.ioDelayMatrix(3,4));
g44=poly2tfd(G(4,4).num{:,:},G(4,4).den{:,:},0,G.ioDelayMatrix(4,4));

plant=tfd2step(tfinal,delt,4,...
   g11,g21,g31,g41,...
   g12,g22,g32,g42,...
   g13,g23,g33,g43,...
   g14,g24,g34,g44);

gd11=poly2tfd(Gd(1,1).num{:,:},Gd(1,1).den{:,:},0,Gd.ioDelayMatrix(1,1));
gd21=poly2tfd(Gd(2,1).num{:,:},Gd(2,1).den{:,:},0,Gd.ioDelayMatrix(2,1));
```

```
   gd31=poly2tfd(Gd(3,1).num{:,:},Gd(3,1).den{:,:},0,Gd.ioDelayMatrix(3,1));
   gd41=poly2tfd(Gd(4,1).num{:,:},Gd(4,1).den{:,:},0,Gd.ioDelayMatrix(4,1));

   gd12=poly2tfd(Gd(1,2).num{:,:},Gd(1,2).den{:,:},0,Gd.ioDelayMatrix(1,2));
   gd22=poly2tfd(Gd(2,2).num{:,:},Gd(2,2).den{:,:},0,Gd.ioDelayMatrix(2,2));
   gd32=poly2tfd(Gd(3,2).num{:,:},Gd(3,2).den{:,:},0,Gd.ioDelayMatrix(3,2));
   gd42=poly2tfd(Gd(4,2).num{:,:},Gd(4,2).den{:,:},0,Gd.ioDelayMatrix(4,2));

   dplant=tfd2step(tfinal,delt,4,...
      gd11,gd21,gd31,gd41,...
      gd12,gd22,gd32,gd42);

   DMCprops.plant = plant;   % Save them
   DMCprops.dplant = dplant;
   clear plant;              % Clear spare variables
   clear dplant;

   end

   % Bounds on the decision variables
   ulb = DMCprops.ulb*ones(length(u),1);
   uub = DMCprops.uub*ones(length(u),1);


% * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

elseif cmethod == 2

               %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
               %                                   %
               % Q control - may be non-square     %
               %                                   %
               %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   disp('Method = 2: Optimal Linear Control.');
   if isempty(initguess.q) == 1
      u = 0.*ones(Qprops.len*cols.^2,1);
   else
      [rig,cig] = size(initguess.q);
      if rig == Qprops.len*cols.^2 & cig == 1
         disp('Dimensions of user-defined initial guess are correct.');
         u = initguess.q;
      else
         disp('Dimensions of user defined initial guess INCORRECT.');
         disp('Aborting.                                  ');
         return
      end
   end
   % Required for the Generalised feedback framework
   Qprops.T1 = makeT1(Gd,tfinal,delt,t,n);
   Qprops.T2 = makeT2(G,tfinal,delt,t,n);
   Qprops.T3 = makeT3(Gd,tfinal,delt,t,n);
```

```
   % Bounds on the decision variables
   ulb = Qprops.ulb*ones(length(u),1);
   uub = Qprops.uub*ones(length(u),1);

% * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

elseif cmethod == 3

              % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              %                                    %
              % Optimal Open-loop input            %
              % trajectory optimization            %
              %                                    %
              % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


   % OOLIT - may be ANYTHING
   disp('Method = 3: Optimal Input Trajectory Search.');
   if isempty(initguess.OOLIT) == 1
      u = 2000.*reshape(  [zeros(1,cols);ones(n-1,1)*(ssinputs(:,1)')] ,cols*n,1);
   else
      [rig,cig] = size(initguess.OOLIT)
      if rig == n*cols & cig == 1
         disp('Dimensions of user-defined initial guess are correct.');
         u = initguess.OOLIT;
      else
         disp('Dimensions of user defined initial guess INCORRECT.');
         disp('Rows of given guess are:');
         rig
         disp('Rows should be');
         n*cols
         disp('Aborting.                                    ');
         return
      end
   end

   % Bounds on the decision variables
   ulb = OOLITprops.ulb*ones(length(u),1);
   uub = OOLITprops.uub*ones(length(u),1);

% * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

else

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              %                                    %
              % UNDEFINED OPTION FOR CMETHOD       %
              %                                    %
              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
   % This is called if cmethod is not currently identifiable.
   disp('Only control methods 0-3 are currently supported.');
   disp(' ');
   disp('type');
   disp(' ');
   disp('    >> help HENdynopt');
   disp(' ');
   disp('for details.');
   disp('Aborting.                                    ');
   return
end

% * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
% SETPOINT SETUP + ERROR CHECKING
[rysp,cysp] = size(ysp);
if isempty(ysp) == 1
   ysp = ones(nn,rows);
elseif rysp == 1
   ysp = ones(nn,1)*ysp;
elseif rysp == nn
   disp('Dimensions of user defined setpoint trajectory are correct.');
else
   disp('Dimensions of user defined setpoint trajectory INCORRECT.');
   disp('Aborting.                                    ');
   return
end

%-----------------------------------------------------------

% OPTIMIZATION CALLING ROUTINE
[usol,mincost,exitflag,output] = fmincon('PMeasure',u,...
   [],[],[],[],ulb,uub,...
   'Conset',...
   Options,...
   t,newt,n,nn,W,ysp,G,Gd,ll,ul,d,delt,deltsim,...
   rows,cols,rowsd,colsd,cmethod,PIprops,DMCprops,Qprops,...
   OOLITprops,Constrprops,ssinputs,typeISE);

% Optimization solver format:
% X=FMINCON(FUN,X0,A,B,Aeq,Beq,LB,UB,NONLCON,OPTIONS)

%-----------------------------------------------------------
%-----------------------------------------------------------
% ALERT THAT PROGRAM HAS FINISHED - play a sound file
try
   [sndvec,Fs,bits] = wavread('loop1');
   sound(sndvec,Fs,bits);
   clear sndvec Fs bits
catch
   disp('No sound hardware detected')
end
```

```
%-------------------------------------------------------------
% RESULTS RETRIEVAL
% I reckon we put some stuff on the screen,
% put up a graph or two, export all results
% as objects in 'results', and get the
% @#$%^%& outta' here.

disp(' ');
disp('The minimized performance measure was');
mincost
disp('for the conditions');
tfinal
delt
deltsim
disp('and the values of the step disturbance vector were');
d
disp('and the values the solution is');
usol
disp(' ');

%-------------------------------------------------------------
% SELECTION OF CONTROL ALGORITHM METHOD FOR GRAPH
BSflag = 1;
figure
if cmethod == 0
   % Using digital PI control
%   [ytraj,utraj] = CalcyPI(usol,t,n,G,Gd,ysp,...
%      d,ll,ul,newt,delt,deltsim,PIprops,BSflag,rows,cols,colsd);
    [ytraj,utraj] = digCalcyPI(u,t,n,G,Gd,ysp,...
    d,ll,ul,newt,delt,deltsim,PIprops,BSflag,rows,cols,colsd);

   title('Input-Output Trajectories Using Optimal PI Control');
elseif cmethod == 1
   % Using DMC control
   [ytraj,utraj] = CalcyDMC(usol,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,DMCprops,BSflag,rows,cols);
   title('Input-Output Trajectories Using Optimal DMC Control');
elseif cmethod == 2
   % Using Q-parametrization
   [ytraj,utraj] = CalcyQ(usol,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,Qprops,BSflag,rows,cols);
   title('Input-Output Trajectories Using Optimal Linear Control (Q-parametrization)');
elseif cmethod == 3
   % Using OOLIT
   [ytraj,utraj] = CalcyOOLIT(usol,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,OOLITprops,BSflag,rows,cols);
   title('Input-Output Trajectories Using Optimal Open-Loop Input Trajectory (OOLIT)');
else
   disp('This is freakin'' weird.');
   disp('Aborting.                              ');
   return
```

```
end

%-------------------------------------------------------
% SAVE RESULTS FOR EXPORT IN A 'RESULTS' STRUCTURE
Results.ise = mincost;
Results.tfinal = tfinal;
Results.delt = delt;
Results.deltsim = deltsim;
Results.usol = usol;
Results.ytraj = ytraj;
Results.utraj = utraj;

% Now we have trajectories - lets plot the graphs
ylabsA = cellstr(['y1';'y2';'y3';'y4']);
ylabsB = cellstr(['u1';'u2';'u3';'u4']);

for i = 1:rows
   subplot(2,rows,i), plot(newt,ytraj(:,i),'k-');
   ylabel(ylabsA(i));xlabel('time(s)');
   subplot(2,rows,rows+i), plot(newt,utraj(:,i),'k-');
   ylabel(ylabsB(i));xlabel('time(s)');
end
```

F.1.3 The cost function (performance measure) for the operability optimizations

filename: Pmeasure.m

```
function J = PMeasure(u,t,newt,n,nn,W,ysp,G,Gd,ll,ul,d,delt,deltsim,...
   rows,cols,rowsd,colsd,cmethod,PIprops,DMCprops,Qprops,...
   OOLITprops,Constrprops,ssinputs,typeISE);

BSflag = 1; % Want a big trajectory vector

% DO THE SIMULATION

% SELECTION OF CONTROL ALGORITHM METHOD
if cmethod == 0
   % Using digital PI control
   [ytraj,utraj] = digCalcyPI(u,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,PIprops,BSflag,rows,cols,colsd);
elseif cmethod == 1
   % Using DMC control
   [ytraj,utraj] = CalcyDMC(u,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,DMCprops,BSflag,rows,cols);
elseif cmethod == 2
   % Using Q-parametrization
   [ytraj,utraj] = CalcyQ(u,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,Qprops,BSflag,rows,cols);
elseif cmethod == 3
```

```
   % Using OOLIT
   [ytraj,utraj] = CalcyOOLIT(u,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,OOLITprops,BSflag,rows,cols);
else
   disp('This is weird.');
   disp('Aborting.');
   stop
end

y = ytraj;

% CONTINUE WITH OBJECTIVE CALCULATION
J = 0;
if typeISE == 0
   % sum of setpoint error
   % weigthing matrix not included
   for q = 1:nn
      term = sum(abs(y(q,:)-ysp(q,:)));
      J = J + (term);
   end
elseif typeISE == 1
   % sum of square setpoint error
   % weighting matrix included
   J = sum(sum(100*((y-ysp).^2)));
elseif typeISE == 2
   % time-weighted sum of setpoint error
   for q = 1:nn
      term = newt(q).*sum(abs(y(q,:)-ysp(q,:)));
      J = J + (term);
   end
elseif typeISE == 3
   % time-weighted sum of square setpoint error
   J = sum(sum(t'*100*((y-ysp).^2)));
end
```

F.1.4 The constraints function for the operability optimizations

filename: conset.m

```
function [g,geq] = conset(u,t,newt,n,nn,W,ysp,G,Gd,ll,ul,d,...
   delt,deltsim,rows,cols,rowsd,colsd,cmethod,PIprops,DMCprops,Qprops,...
   OOLITprops,Constrprops,ssinputs,typeISE);

BSflag = 0;

% SELECTION OF CONTROL ALGORITHM METHOD
if cmethod == 0
   % Using digital PI control
   %[ytraj,utraj] = CalcyPI(u,t,n,G,Gd,ysp,...
```

```
   %   d,ll,ul,newt,delt,deltsim,PIprops,BSflag,rows,cols,colsd);

   [ytraj,utraj] = digCalcyPI(u,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,PIprops,BSflag,rows,cols,colsd);

elseif cmethod == 1
   % Using DMC control
   [ytraj,utraj] = CalcyDMC(u,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,DMCprops,BSflag,rows,cols);
elseif cmethod == 2
   % Using Q-parametrization
   [ytraj,utraj] = CalcyQ(u,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,Qprops,BSflag,rows,cols);
elseif cmethod == 3
   % Using OOLIT
   [ytraj,utraj] = CalcyOOLIT(u,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,OOLITprops,BSflag,rows,cols);
else
   disp('This is freakin'' weird.');
   disp('Aborting.                              Loser.');
   return
end

% INITIALIZE CONSTRAINT VECTORS
g = [];    % Initialize inequality constraint residual vector
geq = [];  % Initialize equality constraint residual vector
a = 0;     % a 'length of g' counter
b = 0;     % a 'length of geq' counter

ysp = ones(n,1)*ysp(1,:);

% SPECIFICATION OF EQUALITY CONSTRAINTS
% 1) Setpoint satisfied at end of simulation
if Constrprops.setpoint == 1
   geq(b+1:b+(Constrprops.numsetpoint+1)*rows) = ...
      reshape(ytraj(n-Constrprops.numsetpoint:n,1:rows)...
      - ysp(n-Constrprops.numsetpoint:n,1:rows),...
      (Constrprops.numsetpoint+1)*rows,1);
   b = length(geq);
elseif Constrprops.setpoint == 0
   % Leave out setpoint constraints
else
   disp('Unrecognized setting for setpoint constraint.');
   disp('Aborting.');
   return
end

% 1a) Constrain initial move to zero ONLY FOR OOLIT
if cmethod == 3
   geq(b+1:b+rows) = utraj(1,1:rows);
%   for p = 1:rows
%      geq(b+p) = utraj(1,p);
```

```
%   end
else
   % Nothing
end
b = length(geq);

% 2) Input gradient at final time (guarentee settling)
if Constrprops.inputgrad == 1
   geq(b+1:b+(Constrprops.numinputgrad+1)*rows) = ...
      reshape(utraj(n-Constrprops.numinputgrad:n,1:rows)...
      - utraj(n-Constrprops.numinputgrad-1:n-1,1:rows),...
      (Constrprops.numinputgrad+1)*rows,1);
   b = length(geq);
elseif Constrprops.inputgrad == 0
   % Leave out input constraints
else
   disp('Unrecognized setting for input gradient constraint.');
   disp('Aborting.');
   return
end

% 3) Ensure final input values are correct at final time
if Constrprops.finalpoint == 1
      geq(b+1:b+(Constrprops.numfinalpoint+1)*rows) = ...
      reshape(utraj(n-Constrprops.numfinalpoint:n,1:cols)...
      - ones(Constrprops.numfinalpoint+1,1)*(ssinputs(1:cols)'),...
      (Constrprops.numfinalpoint+1)*rows,1);
   b = length(geq);
elseif Constrprops.finalpoint == 0
   % Leave out final point constraints
else
   disp('Unrecognized setting for final point constraint.');
   disp('Aborting.');
   return
end

if cmethod == 20
   onetf = tf(1,1,delt);
   I = onetf*eye(rows);
   Gz = c2d(G,delt,'tustin');
   Q = makeQ(u,n,rows,cols,delt,2,G);
   K = inv(I+Q*Gz)*Q;
   for i =1:rows
      for j = 1:cols
         if i == j
            %nothing
         else
            vec = K(i,j).num{:,:};
            geq(b+1) = sum(abs(vec));
         end
         b = length(geq);
      end
```

```
   end
end

% SPECIFICATION OF INEQUALITY CONSTRAINTS
% 1) Constraints on manipulated variables
if Constrprops.manipvar == 1
      for q = 1:cols
         g(a+1:a+n) = ll(:,q) - utraj(:,q);
         a = length(g);
         g(a+1:a+n) = utraj(:,q) - ul(:,q); % upper limits
         a = length(g);
      end
      a = length(g);
elseif Constrprops.manipvar == 0
   % Leave out final point constraints
else
   disp('Unrecognized setting for manipulated variable constraint.');
   disp('Aborting.');
   return
end
```

F.1.5 Output trajectory calculation function of PI control

filename: digcalcypi.m

```
function [ytraj,utraj] = digCalcyPI(u,t,n,G,Gd,ysp,...
   d,lowbnd,uppbnd,newt,delt,deltsim,PIprops,BSflag,rows,cols,colsd);

%function [youtvec,inp] = digcalcypi(u,t,n,G,ysp,Gd1,Gd2,...
%   distss1,distss2,lowbnd,uppbnd,newt,deltsim);

u = u/5000; % optimization scaling

% Notes:
%
% ysp must NOT be a time vector - just the setpoint values
% initial values ???
%
% Basically, we generate the whole trajectory using the
% velocity form of the digital PI controller.  'Nuf said.
%
% Note that input constraints are passed here.
% Also, there appear to be 2 options:  saturate input constraints
% here, or put them in as constraints in the optimization
% routine
Plant = [G Gd];

Plantss = ss(Plant);
Plantss = pade(Plantss,4);
delt = t(2) - t(1);
```

```
Kc = u(1:cols,1);
Ti = u(cols+1:2*cols,1);
Plantns = size(Plantss,'order');
statePlantss = zeros(1,Plantns);
ebef = zeros(cols,1);%[0;0;0;0];     % error of previous timestep (steady state
y = zeros(cols,1);%[0;0;0;0];  % y values of previous timestep
dp = zeros(cols,1);%[0;0;0;0];
pbef = zeros(cols,1);%[0;0;0;0];
pn = zeros(cols,1);%[0;0;0;0];
for p = 1:n
   % some freedom in the time vector here - can use more points than Gc does
%   [y,jnk,states] = lsim(Plantss,...
%      [ones(length([0:deltsim:delt]),1)*pn' ones(length([0:deltsim:delt]),1)*d],...
%      [0:deltsim:delt],statePlantss);

   %[yd,jnk,statesd] = lsim(Gd,...
   %  [ones(length([0:deltsim:delt]),1)*d],[0:deltsim:delt],stateGdbef);

   %y = y + yd;  % add the disturbance in

   %-----------------------------------------------------------------

   % y comes out in a 2-row matrix - we take the FIRST row only
   %                                 (to find initial input)

   if p == 1
      y = zeros(1,rows);
   else
      y = y(end,:);
   end

   en = ysp(1,:) - y;  % calculate error of the first row

   %-------------------------------------------------------
   % DIGITAL PI CONTROLLER
   for hh = 1:cols
      if abs(en(hh)-ebef(hh)) < 1e-8
         if p < 5
            dp(hh) = Kc(hh)*((en(hh) - ebef(hh)) + (delt*en(hh))/Ti(hh));
         else
            dp(hh) = 0;
         end
      else
         dp(hh) = Kc(hh)*((en(hh) - ebef(hh)) + (delt*en(hh))/Ti(hh));
      end
   end
   dp;
   %caleb
   pn = pbef + dp;
   % ---------Now we have the input move--------------------------
   % ----------APPLY INPUT CONSTRAINTS----------------------------
   % option here to apply input constraints !
```

```
   for q = 1:rows
      if pn(q) > uppbnd(q)
         pn(q) = uppbnd(q);
      elseif pn(q) < lowbnd(q)
         pn(q) = lowbnd(q);
      else
         % Do nothing
      end
   end

   % ---------we do the REAL simulation-------------------------
%   [y,jnk,states] = lsim(G,...
%      [ones(length([0:deltsim:delt]),1)*pn'],[0:deltsim:delt],stateGbef);
%   [rst,cst] = size(states);
%   stateGbef = states(rst,:);  % store final states to use next time
   pbef = pn;                  % pass input back

%  [yd,jnk,statesd] = lsim(Gd,...
%     [ones(length([0:deltsim:delt]),1)*d],[0:deltsim:delt],stateGdbef);
%   [rstd,cstd] = size(statesd);
%   stateGdbef = statesd(rstd,:);  % store final states to use next time

   %   y = y + yd;  % add the disturbance in

   [y,jnk,states] = lsim(Plantss,...
      [ones(length([0:deltsim:delt]),1)*pn' ones(length([0:deltsim:delt]),1)*d],...
      [0:deltsim:delt],statePlantss);

   [ry,cy] = size(y);
   statePlantss = states(ry,:);  % store final states to use next time

%   [ry,cy] = size(y);
%   if p == length(t)
%      for pp = 1:1 %(delt/deltsim)
%         for q = 1:cy
%            youtvec(p*(delt/deltsim)-(delt/deltsim)+pp,q) = y(pp,q);  %save the y
%         end
%      end
%   else
%      for pp = 1:(delt/deltsim)
%         for q = 1:cy
%            youtvec(p*(delt/deltsim)-(delt/deltsim)+pp,q) = y(pp,q);  %save the y
%         end
%      end
%   end

   % -------Just do bits and pieces before next run----------------
   ebef = en; % pass new value back
   utraj(p,:) = pn(:)';
   ytraj(p,:) = y(end,:);

   % -----------------END OF MAIN SIMULATION---------------------------------
```

```
end

% We must do the following to make u and y consistent
ytraj = [zeros(1,rows);ytraj(1:end-1,:)];

if (BSflag == 0) | (delt == deltsim)
   % Do nothing - vectors utraj and ytraj are already ok
else
   bigu = [];
   a = 0;
   smallu = utraj;  % u is already in columns, the purpose of the above for-for
   for q = 1:n
      if q == n
         bigu(a+1,:) = smallu(q,:);
      else
         for p = 1:(delt/deltsim)
            bigu(a+p,:) = smallu(q,:);
         end
      end
      [a,jnk] = size(bigu);
   end

   try
      ytraj = lsim(Plantss,[bigu ones(length(newt),1)*d],newt);
   catch
      ytraj = stprespMIMO(Plantss,[bigu ones(length(newt),1)*d],newt);
   end

   utraj = bigu;
end
```

F.1.6 Output trajectory calculation function for MPC control

filename: calcydmc.m

```
function [ytraj,utraj] = calcyDMC(lambda,t,n,G,Gd,ysp,...
   d,ll,ul,newt,delt,deltsim,DMCprops,BSflag,rows,cols);

% Example of DMC controller tuning
% By Caleb Hattingh 1999

% INITITALIZATION VARIABLES
tfinal = t(n);
M = DMCprops.M;
P = DMCprops.P;
plant = DMCprops.plant;
dplant = DMCprops.dplant;

% CALCULATE THE MPC CONTROLLER GAIN MATRIX
```

```
model = plant;                              % no plant/model mismatch
%ywt = eye(rows);uwt = diag([lambda]);      % output wgt 1, inp wgt 10
ywt = [];uwt = lambda';
Kmpc1 = mpccon(model,ywt,uwt,M,P);          % calculate gain matrix


% SIMULATION FOR MEASURED STEP DISTURBANCE THROUGH DPLANT
tend = tfinal;
r = ysp(1,:);
usat = [ll(1,:) ul(1,:) Inf.*ones(1,cols)];
tfilter = [];
dmodel = [];%dplant;    % disturbance is MEASURED - affects whether
  % inputs = 0 at t=0 or not ??
[ytraj,utraj] = mpcsim(plant,model,Kmpc1,tend,r,usat,tfilter,...
   dplant,dmodel,d);

% CHOOSING CONTROL OR SIMULATION TRAJECTORIES
if BSflag == 0
   % We want the trajectories sampled only at control move points

   % This is what we already have, so do nothing!
elseif BSflag == 1
   % We want full simulation trajectories
   newu = [];
   a = 0;
   smallu = utraj;
   for q = 1:length(t)
      if q == length(t)
         newu(a+1,:) = utraj(q,:);
      else
         for p = 1:(delt/deltsim)
            newu(a+p,:) = utraj(q,:);
         end
      end
      [a,jnk] = size(newu);
   end

   try
      ytraj = lsim([G Gd],[newu ones(length(newt),1)*d],newt);
   catch
      ytraj = stprespMIMO([G Gd],[newu ones(length(newt),1)*d],newt);
   end


%   ytraj = lsim([G Gd],[newu ones(length(newt),1)*d],newt);
   utraj = newu;
else
   disp('BSflag is not correctly defined!');
   disp('Aborting.');
   stop
end
```

F.1.7 Output trajectory calculation function for optimal linear control (via Q-parametrization)

filename: calcyQ.m

```
function [ytraj,utraj] = CalcyQ(q,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,Qprops,BSflag,rows,cols);

q = q/500;   % optimization scaling

% Program to simulate a closed-loop system using a
% Q-parametrized controller
%
% Caleb Hattingh (c) 2000

% INITIALIZE VARIABLES AND SETTINGS
T1 = Qprops.T1;  % See Q-parametrization theory
T2 = Qprops.T2;  % See Q-parametrization theory
T3 = Qprops.T3;  % See Q-parametrization theory

Q = makeQ(q,n,rows,cols,delt,2);  % Refer to function 'makeQ'

% ASSEMBLE THE CLOSED LOOP TRANSFER FUNCTION
Hzw = T1 + T2*Q*T3;  % See Q-parametrization theory
Hinputs = ones(n,1)*[ysp(1,:) d];         % Inputs to H
Houtputs = lsim(Hzw,Hinputs,t);       % Simulation !!

ytraj = Houtputs(:,1:rows);           % Out pops the output trajectory
utraj = Houtputs(:,rows+1:rows+cols); % AND the manip var. trajectory!

% CHOOSING CONTROL OR SIMULATION TRAJECTORIES
if BSflag == 0
   % We want the trajectories sampled only at control move points

   % This is what we already have, so do nothing!
elseif BSflag == 1
   bigu = [];
   a = 0;
   smallu = utraj;  % u is already in columns, the purpose of the above for-for
   for q = 1:n
      if q == n
         bigu(a+1,:) = smallu(q,:);
      else
         for p = 1:(delt/deltsim)
            bigu(a+p,:) = smallu(q,:);
         end
      end
      [a,jnk] = size(bigu);
   end

   try
```

```
      ytraj = lsim([G Gd],[bigu ones(length(newt),1)*d],newt);
   catch
      ytraj = stprespMIMO([G Gd],[bigu ones(length(newt),1)*d],newt);
   end

   utraj = bigu;
else
   disp('BSflag is not correctly defined!');
   disp('Aborting.');
   stop
end
```

Function for creating T_1, T_2, T_3 and Q

filename: makeT1.m

```
function T1 = makeT1(Gd,tfinal,delt,t,L);

% The structure of T1 is precalculated

[rowsd,colsd] = size(Gd);

Gd = c2d(Gd,delt);

zerotf = tf(0,1);
zerotf = c2d(zerotf,delt);

for p = 1:rowsd
   for k = 1:rowsd
      zerotfmat1(p,k) = zerotf;
   end
end

for p = 1:rowsd
   for k = 1:colsd
      zerotfmat2(p,k) = zerotf;
   end
end

T1 = [zerotfmat1 Gd;zerotfmat1 zerotfmat2];
```

filename: makeT2.m

```
function T2 = makeT2(Gp,tfinal,delt,t,L);

% The structure of T2 is precalculated

[rows,cols] = size(Gp);
```

```
Gp = c2d(Gp,delt,'tustin');

posonetf = tf(1,1);
posonetf = c2d(posonetf,delt);

zerotf = tf(0,1);
zerotf = c2d(zerotf,delt,'tustin');

for p = 1:rows
   for k = 1:cols
      if p == k
         posonetfmat(p,k) = posonetf;
      else
         posonetfmat(p,k) = zerotf;
      end
   end
end

T2 = [Gp;posonetfmat];
```

filename: makeT3.m

```
function T3 = makeT3(Gd,tfinal,delt,t,L);

% The structure of T3 is precalculated

[rowsd,colsd] = size(Gd);

Gd = c2d(Gd,delt);

negonetf = tf(-1,1);
negonetf = c2d(negonetf,delt);

zerotf = tf(0,1);
zerotf = c2d(zerotf,delt);

for p = 1:rowsd
   for k = 1:rowsd
      if p == k
         negonetfmat(p,k) = negonetf;
      else
         negonetfmat(p,k) = zerotf;
      end
   end
end

T3 = [negonetfmat Gd];
```

filename: makeQ.m

```
function Q = makeQ(q,n,rows,cols,delt,method,G)

% This function puts the vector q which contains all the coefficients
% for each controller, into the Q transfer function
%
% Caleb Hattingh (c) 2000

if method == 0
   Gz = c2d(G,delt,'tustin');
   numq = length(q)/rows;
   a = 0;
   for p = 1:rows
      for k = 1:cols
         if p == k
            a = a + 1;
            Q(p,k) = tf(q((a-1)*numq+1:a*numq)',1,delt,'Variable','z^-1');
         else
            Q(p,k) = Gz(p,k);
         end
      end
   end
elseif method == 2
   numq = length(q)/(rows*cols);
   a = 0;
   for p = 1:rows
      for k = 1:cols
         a = a+1;
         Q(p,k) = tf(q((a-1)*numq+1:a*numq)',1,delt,'Variable','z^-1');
      end
   end
end
```

F.1.8 Output trajectory calculation function for optimal input trajectory control

filename: calcyOOLIT.m

```
function [ytraj,utraj] = CalcyOOLIT(uin,t,n,G,Gd,ysp,...
      d,ll,ul,newt,delt,deltsim,OOLITprops,BSflag,rows,cols);

% Program to find the output trajectory of plant with a step disturbance
% given an input trajectory
%
% caleb Hattingh (c) 2000

if delt == deltsim
    BSflag = 0;
else
    %nothing
end
```

```
uin = uin/2000;
%G = ss(G);
%Gd = ss(Gd);
%G = pade(G,2);
%Gd = pade(Gd,2);

% SELECT CORRECT RESOLUTION OF TRAJECTORIES
if BSflag == 0
   utraj = reshape(uin,n,cols);
   try
      ytraj = lsim([G Gd],[utraj ones(n,1)*d],t);
   catch
      ytraj = stprespMIMO([G Gd],[utraj ones(n,1)*d],t);
   end
elseif BSflag == 1
   % We want the full simulation trajectory
   utraj = [];
   a = 0;
   eyemat = neweye(delt,deltsim,n);
   utraj = eyemat*reshape(uin,n,cols);
   try
      ytraj = lsim([G Gd],[utraj ones(length(newt),1)*d],newt);
   catch
      ytraj = stprespMIMO([G Gd],[utraj ones(length(newt),1)*d],newt);
   end
else
   disp('BSflag was incorrectly set.');
   disp('Aborting.');
   stop
end
```

F.2 Flexible design optimization for HENs

F.2.1 Flexible design optimization function for HENs

filename: flexopt.m

```
function Flexresult = flexopt(HENfile)

% THIS CODE SETS UP AND SOLVES A FLEXIBILITY OPTIMIZATION PROGRAM
% FOR HEAT EXCHANGER NETWORKS.  IT IS DESIGNED TO BE EMBEDDED INTO
% THE HENdynopt FILE TO BE CALLED VIA A CERTAIN FUNCTION.
%
% The returned argument is a *structure* with the following fields:
% .decide - set of flexibility decision variables
% etc
% etc
% etc
remdup = 0;
```

```
run(HENfile); % Lets hope this works

% ERROR CHECKING AND SETTING number of vertices
if isempty(thet) == 1 & isempty(thetd) == 1
   % Theta is empty and so there are no SS uncertainties
   disp('No steady-state uncertainties defined:');
   disp('Nominal steady state optimization');
   nVERTICES = 0;
elseif isempty(thet) == 1 & isempty(thetd) ~= 1
   % No SS uncertainties but there are dynamic uncertainties
   disp('Only dynamic disturbances found.');
   nVERTICES = 0;
elseif isempty(thet) ~= 1 & isempty(thetd) == 1
   % SS uncertainties present but no dynamic uncertainties
   disp('No dynamic uncertainties; Only SS uncertainties present.');
   [r,c] = size(thet);
   nVERTICES = 2^r;
   fullset = getvertexset(thet,remdup);
elseif isempty(thet) ~= 1 & isempty(thetd) ~= 1
   % The full deal - both SS and dynamic uncertainties present
   disp('Both dynamic and steady-state uncertainties present');
   [r,c] = size(thet);
   nVERTICES = 2^r;
   fullset = getvertexset(thet,remdup);
else
   % This should never be called
   disp('An error was caught when checking theta settings');
   disp('in the flexibility program.');
   disp(' ');
   disp('Program terminated.');
   return
end

[rd,jnk] = size(thetd);
fullsetd = thetd;
nVERTICESD = 1;

% THIS IS WHERE WE HAVE THE CALCULATION FOR THE NUMBER OF DECISION
% VARIABLES IN THE OPTIMIZATION PROBLEM.
%
% NOTE THAT THIS NUMBER WILL BE RADICALLY DIFFERENT FROM THE
% NUMBER OF ARGUMENTS PASSED TO THE SIMULATION PROGRAMS - WE WILL
% SIMPLY EXTRACT FROM THE SET BELOW, THE NECESSARY VARIABLES
% TO USE FOR THE SIMULATION.
nDECISIONVARIABLES = nHEXareas + nHEATareas + nCOOLareas + ...
   nHEATflows + nCOOLflows + nMANIPVARS + nMANIPVARS + ...
   nVERTICES.*(nHEATflows+nCOOLflows+nMANIPVARS+nMANIPVARS);

% INITIAL GUESS AND LOWER AND UPPER BOUNDS
if isempty(u) == 1
   % Setup initguess - will be massive because of all the variables
   u(1:nHEXareas,1) = 4000.*ones(nHEXareas,1);
```

```
lb(1:nHEXareas,1) = 0.1.*ones(nHEXareas,1);
ub(1:nHEXareas,1) = Inf.*ones(nHEXareas,1);

a = length(u);
u(a+1:a+nHEATareas,1) = 5000.*ones(nHEATareas,1);
lb(a+1:a+nHEATareas,1) = 0.1.*ones(nHEATareas,1);
ub(a+1:a+nHEATareas,1) = Inf.*ones(nHEATareas,1);

a = length(u);
u(a+1:a+nCOOLareas,1) = [30.0790220972896];%12.*ones(nCOOLareas,1);
lb(a+1:a+nCOOLareas,1) = 0.1.*ones(nCOOLareas,1);
ub(a+1:a+nCOOLareas,1) = Inf.*ones(nCOOLareas,1);

a = length(u);
u(a+1:a+nHEATflows,1) = 20.*ones(nHEATflows,1);
lb(a+1:a+nHEATflows,1) = 0.1.*ones(nHEATflows,1);
ub(a+1:a+nHEATflows,1) = Inf.*ones(nHEATflows,1);

a = length(u);
u(a+1:a+nCOOLflows,1) = 40.*ones(nCOOLflows,1);
lb(a+1:a+nCOOLflows,1) = 0.1.*ones(nCOOLflows,1);
ub(a+1:a+nCOOLflows,1) = Inf.*ones(nCOOLflows,1);

a = length(u);
u(a+1:a+nMANIPVARS,1) = 0.5.*ones(nMANIPVARS,1);
lb(a+1:a+nMANIPVARS,1) = 0.0.*ones(nMANIPVARS,1);
ub(a+1:a+nMANIPVARS,1) = 1.0.*ones(nMANIPVARS,1);

a = length(u); % THIS SET IS FOR DISTURBANCES ONLY
u(a+1:a+nMANIPVARS,1) = 0.5.*ones(nMANIPVARS,1);
lb(a+1:a+nMANIPVARS,1) = 0.0.*ones(nMANIPVARS,1);
ub(a+1:a+nMANIPVARS,1) = 1.0.*ones(nMANIPVARS,1);

a = length(u);

for i = 1:nVERTICES
      a = length(u);
u(a+1:a+nHEATflows,1) = 40.*ones(nHEATflows,1);
lb(a+1:a+nHEATflows,1) = 0.1.*ones(nHEATflows,1);
ub(a+1:a+nHEATflows,1) = Inf.*ones(nHEATflows,1);

   u(a+1:a+nCOOLflows,1) = 40.*ones(nCOOLflows,1);
   lb(a+1:a+nCOOLflows,1) = 0.1.*ones(nCOOLflows,1);
   ub(a+1:a+nCOOLflows,1) = Inf.*ones(nCOOLflows,1);
   a = length(u);

   u(a+1:a+nMANIPVARS,1) = 0.5*ones(nMANIPVARS,1);
   lb(a+1:a+nMANIPVARS,1) = 0.0.*ones(nMANIPVARS,1);
   ub(a+1:a+nMANIPVARS,1) = 1.0.*ones(nMANIPVARS,1);
   a = length(u);

   u(a+1:a+nMANIPVARS,1) = 0.5*ones(nMANIPVARS,1);
```

```
      lb(a+1:a+nMANIPVARS,1) = 0.0.*ones(nMANIPVARS,1);
      ub(a+1:a+nMANIPVARS,1) = 1.0.*ones(nMANIPVARS,1);
      a = length(u);

   end
else
   % There was a user-defined initial guess
   if length(u) ~= nDECISIONVARIABLES
      disp('The dimension of your initial guess is')
      disp('NOT correct.')
      disp(' ');
      disp(sprintf('The length of your initial guess was %d',length(u)));
      disp(sprintf('but it should have been %d',nDECISIONVARIABLES));
      disp(' ');
      disp('Thus, the program is terminated.');
      return
   else
      % Initial guess checks out fine
         % Setup initguess - will be massive because of all the variables
   lb(1:nHEXareas,1) = 0.1.*ones(nHEXareas,1);
   ub(1:nHEXareas,1) = Inf.*ones(nHEXareas,1);

   a = length(lb);
   lb(a+1:a+nHEATareas,1) = 0.1.*ones(nHEATareas,1);
   ub(a+1:a+nHEATareas,1) = Inf.*ones(nHEATareas,1);

   a = length(lb);
   lb(a+1:a+nCOOLareas,1) = 0.1.*ones(nCOOLareas,1);
   ub(a+1:a+nCOOLareas,1) = Inf.*ones(nCOOLareas,1);

   a = length(lb);
   lb(a+1:a+nHEATflows,1) = 0.1.*ones(nHEATflows,1);
   ub(a+1:a+nHEATflows,1) = Inf.*ones(nHEATflows,1);

   a = length(lb);
   lb(a+1:a+nCOOLflows,1) = 0.1.*ones(nCOOLflows,1);
   ub(a+1:a+nCOOLflows,1) = Inf.*ones(nCOOLflows,1);

   a = length(lb);
   lb(a+1:a+nMANIPVARS,1) = 0.0.*ones(nMANIPVARS,1);
   ub(a+1:a+nMANIPVARS,1) = 1.0.*ones(nMANIPVARS,1);

   a = length(lb); % THIS SET IS FOR DISTURBANCES ONLY
   lb(a+1:a+nMANIPVARS,1) = 0.0.*ones(nMANIPVARS,1);
   ub(a+1:a+nMANIPVARS,1) = 1.0.*ones(nMANIPVARS,1);

   a = length(lb);

   for i = 1:nVERTICES
      a = length(lb);
      lb(a+1:a+nHEATflows,1) = 0.*ones(nHEATflows,1);
      ub(a+1:a+nHEATflows,1) = Inf.*ones(nHEATflows,1);
```

```
        a = length(lb);
        lb(a+1:a+nCOOLflows,1) = 0.*ones(nCOOLflows,1);
        ub(a+1:a+nCOOLflows,1) = Inf.*ones(nCOOLflows,1);

        a = length(lb);
        lb(a+1:a+nMANIPVARS,1) = 0.0.*ones(nMANIPVARS,1);
        ub(a+1:a+nMANIPVARS,1) = 1.0.*ones(nMANIPVARS,1);

        a = length(lb);
        lb(a+1:a+nMANIPVARS,1) = 0.0.*ones(nMANIPVARS,1);
        ub(a+1:a+nMANIPVARS,1) = 1.0.*ones(nMANIPVARS,1);
    end
    end
end


%-------------Start of flexibility optimization-------------------------
[usol,mincost,exitflag,output] = fmincon('calcJflex',...
   u,...
   [],[],[],[],...
   lb,ub,...
   'flexconm',...
   Options,...
   nDECISIONVARIABLES,nHEXareas,nHEATareas,...
   nCOOLareas,nHEATflows,nCOOLflows,nMANIPVARS,nVERTICES,...
   nVERTICESD,r,rd,ssfile,COOLflowTO,HEATflowTO,nyout,...
   fullset,fullsetd,yset);
%-------------End of flexibility optimization-------------------------

Flexresult.decision = usol;      % The solution
Flexresult.minnomcost = mincost; % Optimal cost
Flexresult.ssfile = ssfile;      % The file to calculate SS outputs
Flexresult.theta = thet;     % The steady-state disturbance set
Flexresult.thetad = thetd;   % The dynamic disturbances set
```

F.2.2 Objective function (design & operating cost) for flexibility optimization

filename: calcJflex.m

```
function [J,Hotduty,Coldduty] = calcJflex(u,nDECISIONVARIABLES,nHEXAREAS,nHEATAREAS,...
   nCOOLAREAS,nHEATFLOWS,nCOOLFLOWS,nMANIPVARS,nVERTICES,...
   nVERTICESD,numtheta,numthetad,ssfile,COOLflowTO,HEATflowTO,nyout,...
   fullset,fullsetd,ysp);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Flexibility Cost optimization function
%
% Notice that you can select for a seperate vertex by specifying
```

```
% 'vnum' and 'vdnum'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%disp('begin calcJflex')

% RETRIEVE THE CORRECT VARIABLE SET FOR SIMULATION
u = formvar(u,nHEXAREAS,nHEATAREAS,nCOOLAREAS, ...
   nHEATFLOWS,nCOOLFLOWS,nMANIPVARS,nVERTICES,nVERTICESD,...
   0,0);

% CALCULATE THE STEADY-STATE OUTPUTS OF THE HEN
yout = feval(ssfile,u,[0 0 0 0]',[0 0]');

% SELECT THE CORRECT CASE DEPENDING ON WHETHER THERE ARE
% HEATING OR COOLING UTILITY EXCHANGERS PRESENT
if nHEATFLOWS == 0 & nCOOLFLOWS == 0
   % No duties to calculate
elseif nHEATFLOWS > 0 and nCOOLFLOWS == 0
   % Only hot duties to be calculated
   Hotduty = ( ...
      (yout(nHEXAREAS+1:nHEXAREAS+nHEATFLOWS) - HEATflowTO).*...
      u(nHEXAREAS+nHEATAREAS+nCOOLAREAS+1:...
        nHEXAREAS+nHEATAREAS+nCOOLAREAS+nHEATFLOWS)');
   Coldduty = 0;
elseif nHEATFLOWS == 0 & nCOOLFLOWS > 0
   % Only cold duties to be calculated
   Coldduty = ( ...
      (yout(nHEXAREAS+1:nHEXAREAS+nCOOLFLOWS) - COOLflowTO).*...
      u(nHEXAREAS+nHEATAREAS+nCOOLAREAS+1:...
        nHEXAREAS+nHEATAREAS+nCOOLAREAS+nCOOLFLOWS)');
   Hotduty = 0;
elseif nHEATFLOWS > 0 & nCOOLFLOWS > 0
   % BOTH hot and cold duties to be calculated
   Hotduty = ( ...
      (yout(nHEXAREAS+1:nHEXAREAS+nHEATFLOWS) - HEATflowTO).*...
      u(nHEXAREAS+nHEATAREAS+nCOOLAREAS+1:...
        nHEXAREAS+nHEATAREAS+nCOOLAREAS+nHEATFLOWS)');
   Coldduty = ( ...
      (yout(nHEXAREAS+nHEATFLOWS+1:nHEXAREAS+nHEATFLOWS+nCOOLFLOWS) - COOLflowTO).*...
      u(nHEXAREAS+nHEATAREAS+nCOOLAREAS+nHEATFLOWS+1:...
        nHEXAREAS+nHEATAREAS+nCOOLAREAS+nHEATFLOWS+nCOOLFLOWS)');
else
   % Can't be anything here
   disp('Mistake !!!')
end

%  HERE WE CALCULATE THE UTILITY COSTS
Coldcost = 3600*0.003*sum(Coldduty);
Hotcost = 3600*0.03*sum(Hotduty);

% HERE WE CALCULATE THE COST OF THE AREAS
Areacost = sum(8600+670.*u(1:nHEXAREAS+nHEATAREAS+nCOOLAREAS).^0.83);
```

```
% TOTAL NETWORK COST FOR THE DESIRED VERTEX OR NOMINAL CONDITION
J = (Areacost+Coldcost+Hotcost);
```

F.2.3 Constraint function for flexibility optimization

filename: flexconm.m

```
function [g,geq] = flexconm(u,nDECISIONVARIABLES,nHEXAREAS,nHEATAREAS,...
   nCOOLAREAS,nHEATFLOWS,nCOOLFLOWS,nMANIPVARS,nVERTICES,...
   nVERTICESD,numtheta,numthetad,ssfile,COOLflowTO,HEATflowTO,nyout,...
   fullset,fullsetd,ysp);

% INITIALIZE VECTOR OF INEQUALITY & EQUALITY CONSTRAINTS
g = [];
geq = [];
a = 0; % initialise 'length of g' variable
b = 0; % initialise 'length of geq' variable

%i=0;

%disp('begin flexconm')
% LOOP THROUGH THE NOMINAL AND ALL OTHER SETS OF INPUT DATA
for i = 0:nVERTICES       % For the nominal case and all SS uncertainties

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %  WITHOUT DISTURBANCE
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % 1) Retrieve current data
   uthisset = formvar(u,nHEXAREAS,nHEATAREAS,nCOOLAREAS, ...
      nHEATFLOWS,nCOOLFLOWS,nMANIPVARS,nVERTICES,nVERTICESD,...
      i,0);

   % 2) Calculate SS outputs for these conditions
   if i == 0
      yout = feval(ssfile,uthisset,zeros(numtheta,1),zeros(numthetad,1));
   else
      yout = feval(ssfile,uthisset,fullset(i,:),zeros(numthetad,1));
   end

   % 3) LETS APPLY SOME CONSTRAINTS

   % a) Constrain the cooling utility output temperature to be
   %    40 degrees C. There is still a degree of freedom in the
   %    selection of cooling utility flowrate.

   g(a+1:a+nCOOLFLOWS) = yout(nyout+nHEATFLOWS+1:...
      nyout+nHEATFLOWS+nCOOLFLOWS) - 40;
   a = length(g);
```

F.3.1 Function to compare linear transfer function resulting from idenification and output from a SIMULINK© model.

filename: idensimmarsA.m

```
cd \
cd d:\matlab\uct\general\print

load marsd1_G;    % Plant
load marsd1_Gd;   % Disturbance Plant

inputnum = 4;
outputnum = 1;

delt = 10;
deltsim = 0.1;
tfinal=100;

t = [0:delt:tfinal]';
n = length(t);
newt = [0:deltsim:tfinal]';
randvec = 0.01*rand(length(t),1);
tmpu = [0*ones(n,1) 0*ones(n,1) 0*ones(n,1) randvec];

% We want the full simulation trajectory
utraj = [];
a = 0;
for q = 1:n
   if q == n
      utraj(a+1,:) = randvec(q,:);
   else
      for p = 1:(delt/deltsim)
         utraj(a+p,:) = tmpu(q,:);
      end
   end
   [a,jnk] = size(utraj);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%555

runtype = 0;
dynfile = 'd1_new';
tsim = newt;
ua = [0
     0.147550363750453
     0
     0.128277706530432];

usima = [19.4275412163383
```

```
        35.7750606681826
        6.16193328506885
        16.0964772396389
        34
        0
        0.147550363750453
        0
        0.128277706530432
        19.4275412163383/80
        35.7750606681826/80
        6.16193328506885/80
        16.0964772396389/80
        0.4
        0.4
        0.4
        0.4
        0
        0];

usim = ones(length(tsim),1)*(usima');

usim(:,5+inputnum) = usima(5+inputnum,1)*ones(length(newt),1)+utraj(:,inputnum);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%555

UT = [tsim usim];
[tout,statevec,youtsim] = sim(dynfile,tsim,[],UT);



Gij = G(outputnum,inputnum)
Gdij = Gd(outputnum,:);

outvec = [90 140 160 200];

d = [0 0];

youtlsim = lsim([Gij Gdij],[utraj(:,inputnum) ones(length(newt),1)*d],newt) + outvec(outputnum)*ones(length(newt),1)

newt=newt(1:end-79,:);
youtlsim=youtlsim(80:end,:);
youtsim=youtsim(80:end,:);

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

figure(1)
title('Comparison of linear model approximation of nonlinear SIMULINK model');

subplot(2,1,1)

h = plot(newt,youtlsim,newt,youtsim(:,outputnum));

set(h,'LineWidth',0.5,{'LineStyle'},{'-';':'});
```

```
set(h,{'Color'},{'k';'k'})

xlabel('');
ylabel('Output (degrees C)');
legend(h,'linear transfer function model','SIMULINK model');

subplot(2,1,2)

h2 = plot(newt,utraj(80:end,inputnum));

set(h2,'LineWidth',0.5,{'LineStyle'},{'-'});
set(h2,{'Color'},{'k'});

xlabel('Time(s)');
ylabel('Input');

print -deps -r300 marsd1u4y1iden
```

F.3.2 Function to create a vector of inputs for a specific vertex for steady-state simulation given a vector containing decision variables for all vertices

filename: formvar.m

```
function uset = formvar(u,nHEXAREAS,nHEATAREAS,nCOOLAREAS, ...
   nHEATFLOWS,nCOOLFLOWS,nMANIPVARS,nVERTICES,nVERTICESD,...
   vnum,vdnum);

% THIS FUNCTION IS A WORK OF ART.  IT EXTRACTS THE CORRECT SET
% OF VALUES REQUIRED TO RUN THE HEN SIMULATION,
% EITHER FOR STEADY-STATE OR DYNAMIC.
setlength = nHEATFLOWS+nCOOLFLOWS+2*nMANIPVARS; %length for 1 vertex

a = 0; % Initialize a counter
uxarea = u(1:nHEXAREAS);
a = a + length(uxarea);
uharea = u(a+1:a+nHEATAREAS);
a = a + length(uharea);
ucarea = u(a+1:a+nCOOLAREAS);
a = a + length(ucarea);
% Finished with areas

if vnum == 0
   uhflow = u(a+1:a+nHEATFLOWS);
   a = a + length(uhflow);
   ucflow = u(a+1:a+nCOOLFLOWS);
   a = a + length(ucflow);
   if vdnum == 0
      ubyp = u(a+1:a+nMANIPVARS);
```

```
    else
       ubyp = u(a+nMANIPVARS+1:a+nMANIPVARS+nMANIPVARS);
    end
else
    a = a + (vnum)*setlength; % shift a by correct amount
    uhflow = u(a+1:a+nHEATFLOWS);
    a = a + length(uhflow);
    ucflow = u(a+1:a+nCOOLFLOWS);
    a = a + length(ucflow);
    if vdnum == 0
       ubyp = u(a+1:a+nMANIPVARS);
    else
       ubyp = u(a+nMANIPVARS+1:a+nMANIPVARS+nMANIPVARS);
    end
end
uset = [uxarea;uharea;ucarea;uhflow;ucflow;ubyp];
```

F.3.3 Function to calculate the outputs of a heat exchanger with a user-defined number of well-mixed cells

filename: calcHEX.m

```
function [Thout,Tcout] = calcHEX(cph,cpc,h,A,Thin,Tcin,uh,uc);

% Steady state HEN solver - mean, not log mean temperature
%                           driving force

if A == 0
    Thout = Thin;
    Tcout = Tcin;
elseif A < 0
    Thout = Thin;
    Tcout = Tcin;
else
    N = 4;
A = A./N;  %(make up for using multiple cells)
aa = -0.5*h*A+(1-uh)*cph;
b = -0.5*h*A-(1-uh)*cph;
c = 0.5*h*A;
e = +(1-uc)*cpc-0.5*h*A;
f = -(1-uc)*cpc-0.5*h*A;

for i = 1:N
    hotrows(i,:) = [zeros(1,i-1) aa b zeros(1,N-i) zeros(1,i-1) c c zeros(1,N-i)];
    coldrows(i,:) = [zeros(1,i-1) c c zeros(1,N-i) zeros(1,i-1) f e zeros(1,N-i)];
end

bigA = [hotrows;coldrows;[1 zeros(1,2*N+1)];[zeros(1,2*N+1) 1]];
bigb = [zeros(2*N,1);Thin;Tcin];
```

```
sol = bigA\bigb;

Thout = (1-uh)*sol(N+1) + uh*Thin;
Tcout = (1-uc)*sol(N+2) + uc*Tcin;
```

F.3.4 Function to transform a lower-upper uncertainty matrix into a matrix with each uncertainty vertex on a row

filename: getvertexset.m

```
function [fullset,nvertices] = getvertexset(theta,remdup);

% This function has the calling sequence
%
% >> fullset = getvertexset(theta)
%
% and, when given a theta matrix of upper and lower bounds
% for a set of variables, returns a new matrix where
% every row is a unique vertex of the binary set.
%
% NOTE: The binary matrix theta MUST always have 2 columns, i.e.
%       lower bounds in column 1 and uppers in columns 2, OR vice versa.
%
% (c) 1999. Caleb Hattingh

theta = theta';
[r,c] = size(theta);

fullset = zeros(2^c,c); % Initialise where the answers will
                        % be placed.
for i = 1:c
   tmpvec = [];
   repvec = [theta(1,i)*ones((2^c)/2^i,1);theta(2,i)*ones((2^c)/2^i,1)];
   for j = 1:(2^(i-1))
      tmpvec = [tmpvec;repvec];
   end
   fullset(:,i) = tmpvec;
end

% There may be some rule-based system for
% special cases of these variables, i.e.
% where there is only a discrete variable
% and not a range.  This can be further
% investigated.

if remdup == 0
   % Do nothing - leave duplicates in.
else
```

```
    gdrow = [];
    idflag = 0;
    for jj = 1:2^c
     for k = 1:2^c
    %disp(sprintf('for jj=%d , k = %d',jj,k));
    %disp(sprintf('vector 1 = %g\n',fullset(k,:)));
    %disp(sprintf('vector 2 = %g\n',fullset(jj,:)));
    %disp(sprintf('similarity = %d',veccompare(fullset(k,:),fullset(jj,:))));
    %disp(' ');
    if veccompare(fullset(k,:),fullset(jj,:)) == 1 & k > jj
        % There is a match
idflag = 1;
    else
% There was no match
    end
     end
if idflag == 0  % If there was no match
   disp('came in here?');
   gdrow = [gdrow;jj];
else
   % Nothing
end
idflag = 0;
    end
    fullset = fullset(gdrow,:);
end
```

F.3.5 Function to simulate a SIMULINK© HEN

filename: HENnonsim.m

```
function Result = HENnonsim(filename);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% All this function does is simulate a nonlinear
% SIMULINK HEN file, based on parameters in the
% data file provided by <filename>
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %                                   %
            % JUST SIMULATE NONLINEAR MODEL     %
            %                                   %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% THE INPUT FILE THAT SPECIFIES ALL THE OPTIONS
run(filename);

if runtype == 0
   % Single run, input must be specified
   Result = HENsimonce(usim,tsim,dynfile);
elseif runtype == 1
   % Multiple runs, inputs specified
   Result = HENsimmany(Usim,Tsim,dynfile); % U and T are STRUCTURES
else
   % Invalid or no option selected
   disp('No valid run type selected.  Aborting...');
   return
end
```

F.3.6 Function to modify multicolumn input trajectory matrix into a vector with consecutive input trajectories stacked

filename: truncinp.m

```
function utraj = truncinp(lb,ub,utraj);

% lb and ub in columns
% utraj has a column for each input, and as many rows in the trajectory

[r,c]=size(utraj);

for p=1:r
   for q=1:c
      if utraj(p,q) < lb(q)
         utraj(p,q) = lb(q);
      elseif utraj(p,q) > ub(q)
         utraj(p,q) = ub(q);
      else
         % nothing
      end
   end
end
```

F.3.7 Function to calculate the steady-state input values required for zero setpoint error under a disturbance influence

filename: ssinputvalues.m

```
function uss = ssinputvalues(G,Gd,ysp,d);

% Calculates the steady-state final input values
% for use with the optimal DMC tuning parameter
% program.
%
% G = a LTI model representing the plant.
% Gd = a LTI model representing the disturbance plant.
% ysp = a setpoint vector.
% d = a disturbance vector.
%
% uss = the steady state inputs required to satisfy setpoints
%       under disturbances.
%
% format:
%       >> uss = ssinputvalues(G,Gd,ysp,d);
%
% Caleb Hattingh 1999.

[mag,phase,omega] = bode(G,0); % Calculates magnitude at zero frequency
G0 = cos(phase*2*pi/360).*mag; % Using phase to get signs right

if isempty(d) == 1          % If no disturbances
   uss = inv(G0)*ysp';
else
   [magd,phased,omegad] = bode(Gd,0); % Calculates magnitude at zero frequency
   Gd0 = cos(phased*2*pi/360).*magd;  % Using phase to get signs right
   uss = inv(G0)*(ysp'-Gd0*d');
end
```

F.3.8 Function to calculate the output trajectory of a SISO transfer function resuling from time-varying input use - generates and uses step-response model internally

filename: stpresp.m

```
function y = stpresp(ysrp,u,t);

% Use a step response model for simulation
% This function is SISO only.

y = zeros(length(u),1);

y = ysrp.*u(1);
for i = 2:length(u)
   if u(i,1) == u(i-1,1)
      % Nothing
   else
      y = y + [zeros(i-1,1);ysrp(1:length(u)-i+1,1).*(u(i)-u(i-1))];
```

```
   end
end
```

F.3.9 Function to calculate MIMO response using internal step response model

filename: stprespMIMO.m

```
function ytraj = stprespMIMO(G,u,t);

% This calculates the MIMO outputs using step response models

[rows,cols] = size(G);  % This G includes the disturbance matrix

ytraj = zeros(length(u),rows);
for i = 1:rows
   for j = 1:cols
      ystp = step(G(i,j),t);
      ytraj(:,i) = ytraj(:,i) + stpresp(ystp,u(:,j),t);
   end
end
```

F.3.10 Function that extracts data from another vector at discrete intervals

filename: mksmall.m

```
function ansvec = mksmall(u,delt,deltsim,tfinal);

t = [0:delt:tfinal]';
newt = [0:deltsim:tfinal]';
colvec = ones(delt/deltsim,1);
A = [];
for j = 1:length(t)
   A((delt/deltsim)*(j-1)+1:(delt/deltsim)*j,j) = colvec;
end
tt = A*t;
r = 0;
% Short function to get only unique data points out of a vector
for i = 1:length(newt)
   if newt(i) == tt(i)
      ansvec(r+1,:) = u(i,:);
      [r,c] = size(ansvec);
   else
      % nothing
   end
end
```

F.3.11 Input file for dynamic operability study

filename: gund2.m

```
%-----------------------------------------------------------------------
%
%  This is a template file for use with the Dynamic Operability Toolbox.
%
%  Name: Joe Spud
%  Date: 17 May 99
%  Source of Example: Marcelle et al (198?)
%
%-----------------------------------------------------------------------

warning off

load gund2_G;    % Plant
load gund2_Gd;   % Disturbance Plant

%G(3,2).num{:,:} = 4*G(3,2).num{:,:}

G.iodelaymatrix;

col1 = G(:,2);
col2 = G(:,4);
col3 = G(:,3);
col4 = G(:,1);

G = [col1 col2 col3 col4];

rga(G)

G.ioDelayMatrix;

% Note - G still in tf format

d = [-5 5];     % Disturbance step values
%lowbnd = [-0.054003;-0.21747;0;0];   % Lower M.V. bound
%uppbnd = [1-0.054003;1-0.21747;1;1]; % Upper M.V. bound
lowbnd = [-5;-5;-5;-5];   % Lower M.V. bound
uppbnd = [5;5;5;5]; % Upper M.V. bound

ysp = [0 0 0 0];  % Setpoint values for whole trajectory

% SIMULATION PARAMETERS
tfinal = 2000;         % Final Time
delt = 10;            % Control Move Timestep
deltsim = 10;       % Simulation Timestep0.5
typeISE = 1;

% PI-SPECIFIC PARAMETERS
```

```
PIprops = [];
initguess.pi = [    -0.6383
   -0.0030
    0.2492
    0.0470
    0.0385
    0.0380
    0.0217
    0.0094];
PIprops.ulb = -Inf;%[-Inf -Inf -Inf -Inf 0 0 0 0]';
PIprops.uub = Inf;

% DMC-SPECIFIC PARAMETERS
DMCprops.M = 40; % Control move horizon
DMCprops.P = 40; % Prediction Horizon
initguess.dmc = [10.627
       15.643
       12.736
    0.0005002];
DMCprops.ulb = 0;
DMCprops.uub = 10000;

% Q-PARAMETRIZATION-SPECIFIC PARAMETERS
Qprops.len = 10; % Number of q coefficients per IO model
initguess.q = [];
Qprops.ulb = -Inf;
Qprops.uub = Inf;

% OOLIT-SPECIFIC PARAMETERS
OOLITprops = [];
initguess.OOLIT = [];
OOLITprops.ulb = -Inf;
OOLITprops.uub =  Inf;

% OPTIMIZATION PARAMETERS
Options.Display = 'iter';
Options.MaxIter = 5000;
Options.TolFun = 1e-4;
Options.TolCon = 1e-6;
Options.MaxFunEvals = 1e8;
Options.TolX = 1e-4;
Options.LargeScale = 'off';
Options.Diagnostics = 'on';
%Options.DiffMaxChange = 1e-1;
%Options.DiffMinChange = 1e-8;

% CONSTRAINT PROPERTIES

% a) Equality Constraints
Constrprops.setpoint = 0;
Constrprops.numsetpoint = 1;
```

```
Constrprops.inputgrad = 0;
Constrprops.numinputgrad = 1;

Constrprops.finalpoint = 0;
Constrprops.numfinalpoint = 1;

% b) Inequality Constraints
Constrprops.manipvar = 0;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Flexibility optimization Options
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Design options
nHEXareas = 3;   % Number of process heat exchangers
nHEATareas = 0;  % Number of utility heaters
nCOOLareas = 1;  % Number of utility coolers
nHEATflows = 0;  % Number of hot utility flows
nCOOLflows = 1;  % Number of cold utility flows
nMANIPVARS = 4;  % Number of bypass manipulations for dynamic control.
nyout = 4;
vols = [0;0;0;0];

% Utilities settings
HEATflowT0 = [];     % Temperature of Hot utility
COOLflowT0 = [10];  % Temperature of Cold utility

% Setpoint information (operating point)
yset = [60 60 70 90];

% Uncertainty information
thet = [];%-0.005 0.005
        %-0.005 0.005
        %-0.005 0.005];          % Steady state uncertainties
        %-0.005 0.005];

thetd = [-5;5];           % Dynamic disturbances
                    % (note identical entries left and right).

% Filename of SS HEN sim file
ssfile = 'calcygund2';

% Flexibility optimization parameters
%Options.Display = 'iter';
%Options.MaxIter = 100;
%Options.TolFun = 1e-6;
%Options.TolCon = 1e-8;
%Options.TolX = 1e-4;
%Options.LargeScale = 'off';
```

```
%Options.Diagnostics = 'on';
%Options.DiffMaxChange = 1e-4;

% Initial guess - use empty matrix for none
u = [        352.36
       369.75
       321.34
        348.4
       51.667
 -7.5123e-026
     0.054003
  5.7899e-027
      0.21747
      0.18586
  1.0051e-026
     0.028541
    1.06e-025];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Identification Options
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

idenprops.dynfile = 'gund2sim';
idenprops.tfinal = 400; % Final time
idenprops.delt = 20;        % timestep for input changes
idenprops.deltsim = 0.1; % timestep for simulation
idenprops.ttrunc = 3; % number of initial points to truncate

% Set max range for randomized input trajectory
idenprops.maxrange = 0.1;

idenprops.inputnum = 4; % Number of inputs
idenprops.outputnum = 4; % number of outputs

% Nominal steady-state input
usima = [          3.523588562500329e+002
    3.697511442758766e+002
    3.213391172508431e+002
    3.483975865818115e+002
    5.166666666679294e+001
    0
    5.400262908650718e-002
    0
    2.174723649356071e-001
         3.523588562500329e+002/80
    3.697511442758766e+002/80
    3.213391172508431e+002/80
    3.483975865818115e+002/80
         0.1
         0.1
```

```
        0.1
        0.1
        0
        0];
idenprops.outvec = [60 60 70 90];
idenprops.d = [0 0];
```

F.3.12 MATLAB© code for calculating open-loop indicators

Relative gain array

filename: rga.m

```
function RGA = rga(G)

% This function calculates the RGA

[mag,phase,omega] = bode(G,0); % Calculates magnitude at zero frequency
%G0 = cos(phase*2*pi/360).*mag; % Using phase to get signs right
G0 = mag;
G0it = (inv(G0))';
RGA = G0.*G0it;
```

Performance relative gain array

filename: prga.m

```
function PRGA = prga(G)

% This function calculates the RGA

[mag,phase,omega] = bode(G,0); % Calculates magnitude at zero frequency
%G0 = cos(phase*2*pi/360).*mag; % Using phase to get signs right
G0 = mag;
G0it = (inv(G0))';

PRGA = diag(diag(G0))*G0it;
```

Condition number

filename: cn.m

```
function [cn,w] = cn(G)

% This function calculates the RGA
w = logspace(-3,5,100)';
n = length(w);
```

```
for i = 1:n
   [mag,phase,omega] = bode(G,w(i)); % Calculates magnitude at zero frequency
   %G0 = cos(phase*2*pi/360).*mag; % Using phase to get signs right
   G0 = mag;
   maxsvG = max(svd(G0));
   minsvG = min(svd(G0));

   cn(i) = maxsvG/minsvG;
end
```

Disturbance condition number

filename: dcn.m

```
function [dcn,w] = dcn(G,Gd)

% This function calculates the RGA
w = logspace(-3,5,100)';
n = length(w);

for i = 1:n
   [mag,phase,omega] = bode(G,w(i)); % Calculates magnitude at zero frequency
   %G0 = cos(phase*2*pi/360).*mag; % Using phase to get signs right
   G0 = mag;
   [magd,phased,omegad] = bode(Gd,w(i)); % Calculates magnitude at zero frequency
   %Gd0 = cos(phase*2*pi/360).*magd; % Using phase to get signs right
   Gd0 = magd;
   svG = max(svd(G0));

   dcn(i) = norm(inv(G0)*Gd0,2)*svG/norm(Gd0,2);
end
```

Closed-loop disturbance gain

filename: cldg.m

```
function [cldg,w] = cldg(G,Gd,w)

% This function calculates the RGA
%w = logspace(-3,5,100)';
%n = length(w);

[mag,phase,omega] = bode(G,w); % Calculates magnitude at zero frequency
%G0 = cos(phase*2*pi/360).*mag; % Using phase to get signs right
G0 = mag;
[magd,phased,omegad] = bode(Gd,w); % Calculates magnitude at zero frequency
%Gd0 = cos(phase*2*pi/360).*magd; % Using phase to get signs right
Gd0 = magd;
```

```
Gdiag = diag(diag(G0));
Ginv = inv(G0);

cldg = Gdiag*Ginv*Gd0;
```

Relative disturbance gains

filename: rdg.m

```
function [rdg0,Gd0] = rdg0(G,Gd)

% This function calculates the RGA
%w = logspace(-3,5,100)';
%n = length(w);
w = 0;
[mag,phase,omega] = bode(G,w); % Calculates magnitude at zero frequency
%G0 = cos(phase*2*pi/360).*mag; % Using phase to get signs right
G0 = mag;
[magd,phased,omegad] = bode(Gd,w); % Calculates magnitude at zero frequency
%Gd0 = cos(phase*2*pi/360).*magd; % Using phase to get signs right
Gd0 = magd;

Gdiag = diag(diag(G0));
Ginv = inv(G0);

[r,c] = size(Gd);
for i = 1:c
   rdg0(:,i) = Gdiag*Ginv*Gd0(:,i)./Gd0(:,i);
end
```

Morari resiliency index

filename: mri.m

```
function [MRI,w] = MRI(G)

% This function calculates the RGA
w = logspace(-3,5,100)';
n = length(w);

for i = 1:n
   [mag,phase,omega] = bode(G,w(i)); % Calculates magnitude at zero frequency
   %G0 = cos(phase*2*pi/360).*mag; % Using phase to get signs right
   G0 = mag;

   MRI(i) = min(svd(G0));
end
```

F.3.13 MATLAB© code for the identification of HENs as linear transer functions with time delay

Main Identification Program

filename: IdenG.m

```
function Result = idenG(filename);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% This function identifies a simple linear transfer
% function model of nonlinear SIMULINK-modelled
% heat exchanger network.
%
% However, it may be used to identify ANY process,
% since all that is for this function is input
% trajectory information and time information.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Intro
disp(' ');
disp('-------------------------------------------------------------');
disp('         H E N s - I d e n t i f i c a t i o n');
disp(' ');
disp('                      Caleb Hattingh');
disp(' ');
disp('-------------------------------------------------------------');
disp(' ');

% Run datafile to get access to variables
run(filename);
disp('loading datafile...');
% Get simulink file
dynfile = idenprops.dynfile;

% Get timedata
% t = idenprops.t;
tfinal = idenprops.tfinal; % Final time
delt = idenprops.delt;     % timestep for input changes
deltsim = idenprops.deltsim; % timestep for simulation
ttrunc = idenprops.ttrunc; % number of initial points to truncate

idenprops.initguess = [];

outvec = idenprops.outvec; % ss outputs from simulink
d = idenprops.d;

% Set max range for randomized input trajectory
maxrange = idenprops.maxrange;

% Transfer function settings
%sizepolynum = idenprops.sizepolynum;
%sizepolyden = idenprops.sizepolyden;

inputnumtot = idenprops.inputnum;
outputnumtot = idenprops.outputnum;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% b u i l d   t i m e   v e c t o r
% b u i l d   r a n d o m   v e c t o r
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('building simulation vectors...');
t = [0:delt:tfinal]';n = length(t);
newt = [0:deltsim:tfinal+ttrunc*delt]';nn = length(newt);

randvec = [0.01*ones(ttrunc,1);zeros(ttrunc-1,1);zeros(1,1);maxrange*rand(n,1)];

tmpu = zeros(length(randvec),inputnumtot);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% l o o p   t h r o u g h   a l l   i n p u t s   a n d   o u t p u t s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp(' ');

for inputnum = 1:inputnumtot

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % b u i l d   i n p u t   v e c t o r
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   disp('building random step input vector...');
   tmpu(:,inputnum) = randvec; % replace correct input with rand step
   % Here we fill out the input vector for finer resolution
   utraj = [];
   a = 0;
   for q = 1:n+ttrunc+ttrunc
      if q == n+ttrunc+ttrunc
         utraj(a+1,:) = randvec(q,:);
      else
         for p = 1:(delt/deltsim)
            utraj(a+p,:) = tmpu(q,:);
         end
      end
      [a,jnk] = size(utraj);
   end

   [rutraj,cutraj] = size(utraj);

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % n o n l i n e a r   s i m u l a t i o n   s e t u p
   % note: must iterate through each input and output, and provide a graph
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Fill out SIMULINK trajectory appropriately
   usim = ones(rutraj,1)*(usima'); %usima defined in input file

   % Now replace input trajectory with the random one
   usim(:,6+inputnum) = usima(6+inputnum,1)*ones(rutraj,1)+utraj(:,inputnum);

   % New time matrix for longer simulation
   specialt = [0:deltsim:tfinal+ttrunc*delt+ttrunc*delt]';

%   size(usim)
%   size(specialt)

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % n o n l i n e a r   s i m u l a t i o n
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   disp('running SIMULINK simulation (this may take a while)...');
```

```
UT = [specialt usim]; % build simulink input
[tout,statevec,youtsim] = sim(dynfile,specialt,[],UT); % Do simulink sim

disp('...done.')
disp(' ');
disp(' ');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% s h o r t e n  l e n g t h s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

innon = utraj(ttrunc*delt/deltsim+1:rutraj,inputnum);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% d o  o u t p u t s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for outputnum = 1:outputnumtot

   outnon = youtsim(ttrunc*delt/deltsim+1:rutraj,outputnum) - outvec(outputnum)*ones(nn,1);
%     size(outnon)

   disp(sprintf('For input %d and output %d:\n',inputnum,outputnum));
   disp(' ');
   figure(1)
   subplot(2,1,1)
   plot(newt,outnon);
   subplot(2,1,2)
   plot(newt,innon);

   dothisone = input('Do you want to identify this IO pair (y/n) ? ','s');
   disp(' ');
   if dothisone == 'n'
      % Do nothing
   elseif dothisone == 'y'
      doagain = 1;
      while doagain == 1
         theta = input('Value of dead-time to use: ');
         sizepolynum = input('Number of numerator terms in TF: ');
         sizepolyden = input('Number of denominator terms in TF: ') - 1;
         disp(' ');

         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         % l i n e a r  s i m u l a t i o n
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%        youtlsim = lsim([Gij Gdij],[utraj(:,inputnum) ones(length(newt),1)*d],newt) + outvec(outputnum)*ones(lengt

         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         % INITIAL GUESS CHECKING AND PROVIDING
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

         if isempty(idenprops.initguess) == 1
            u = [0.0001*ones(sizepolynum,1);10*ones(sizepolyden,1)];
         else
            if length(u) == sizepolynum + sizepolyden
               disp('Dimension of initial guess for identification is CORRECT.')
            else
               disp('INCORRECT dimension of initial guess.  Aborting...')
               disp(' ');
               disp(' ');
               return
```

```
   end
end

choice = input('Do you want to specify initial guess? ','s');
if choice == 'y'
   msg001 = sprintf('Column vector, %d elements: ',sizepolynum+sizepolyden);
   u = input(msg001);
else
   % Nothing
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% UPPER AND LOWER BOUNDS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lb = -10000.*ones(length(u),1);
lb(sizepolynum+1) = 1e-3;
ub = 10000.*ones(length(u),1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HERE WE CALL THE IDENTIFICATION PROCEDURE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Options.Display = 'iter';
Options.MaxIter = 50;
Options.TolFun = 1e-8;
Options.TolCon = 1e-8;
Options.TolX = 1e-8;
Options.LargeScale = 'off';
Options.Diagnostics = 'on';
%         Options.LineSearchType = 'cubicpoly';
Options.DiffMaxChange = 1e-12;

[usol,mincost,exitflag,output] = fmincon(...
   'idenfunc',...
   u,...
   [],[],[],[],...
   lb,ub,...
   [],...
   Options,...
   newt,sizepolynum,sizepolyden,...
   innon,outnon,theta);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ANSWER RETRIEVAL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

num = usol(1:sizepolynum)';
den = [usol(sizepolynum+1:sizepolynum+sizepolyden)' 1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FORM TRANSFER FUNCTION MODEL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

G = tf(num,den);
G.ioDelayMatrix = theta % Yes, we must not forget the dead time

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROVIDE SOME GRAPHS TO CHECK ACCURACY OF APPROXIMATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[yout,tout] = lsim(G,innon,newt); % Quick simulation
Gbig(outputnum,inputnum) = G;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        %p l o t t i n g   o u t p u t
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        figure(1);
        title('Comparison of linear model approximation of nonlinear SIMULINK model');
        subplot(2,1,1)
        h = plot(newt,yout,newt,outnon);
        set(h,'LineWidth',0.5,{'LineStyle'},{'-';':'});
        set(h,{'Color'},{'k';'k'})

        xlabel('');
        ylabel('Output (degrees C)');
        legend(h,'linear transfer function model','SIMULINK model');
        subplot(2,1,2)
        h2 = plot(newt,innon);
        set(h2,'LineWidth',0.5,{'LineStyle'},{'-'});
        set(h2,{'Color'},{'k'});
        xlabel('Time(s)');
        ylabel('Input');

        ans = input('Try again (y/n) ? ','s');
        if ans == 'y'
           % dont change the doagain variable
        elseif ans == 'n'
           doagain = 0;
        else
           % bad variable
           return
        end
     end

     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     %p r i n t   f i l e   o f   g r a p h
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     pyn = input('Print plot ? ','s');
     if pyn == 'y'
        pfname = sprintf('identu%dy%d',inputnum,outputnum);
        print('-deps','-r300',pfname)
        %          print -deps -r300 identgph
        disp('EPS file written')
        disp(' ');
     elseif pyn == 'n'
        % Do nothing
     else
        disp('Must press y or n!')
     end
  else
     % Bad Answer
     disp('You answered incorrectly: should be either y or n.')
     disp('Aborting...');
     disp(' ');
     return
  end
 end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SETUP ANSWERS FOR EXPORT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Result.G = Gbig;
Result.deadtime = theta;
```

```
Result.match = mincost;   % We must format mincost as
                          % a cumulative percent change
                          % summmation
```

Objective function used in identification optimization strategy

filename: idenfunc.m

```
function J = idenfunc(u,t,sizepolynum,sizepolyden,...
   innon,outnon,theta);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% THIS FUNCTION CALCULATES THE DIFFERENCE BETWEEN THE
% ACTUAL AND THE MODEL OUTPUTS FOR THE SAME u INPUT
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FIRST, BUILD THE TRANSFER FUNCTION BASED ON THE
% DECISION VARIABLES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
G = []; % Initialize the variable

num = u(1:sizepolynum)';  % Numerator
den = [u(sizepolynum+1:sizepolynum+sizepolyden)' 1]; % Denominator

G = tf(num,den); % Build it
G.ioDelayMatrix = theta; % DONT forget the dead time

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% HERE IS A QUICK SIMULATION
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[yout,tout] = lsim(G,innon,t);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% THIS TRANSFORMATION IS RECOMMENDED WHENEVER YOU USE
% THE lsim COMMAND, TO UNSURE THE CORRECT NUMBER OF
% DATA POINTS
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
d = round(length(tout)/length(t)); % diff in num. of pts
y = yout(1:d:end,:);               % shorten y appropriately


%y = y.*100;
%outnon = outnon.*100;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% THIS IS THE ACTUAL OBJECTIVE FUNCTION, WHICH IS THE
% DIFFERENCE BETWEEN THE NONLINEAR OUTPUT AND THE
% OUTPUT THAT HAS JUST BEEN GENERATED ABOVE
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
absdiff = (abs(y - outnon)).^2; %  the absolute difference at every point
J = sum(absdiff);              % sum all the differences.
```

Index