ABSTRACT

SCOTT, JASON RODERICK. Fault Detection in Differential Algebraic Equations. (Under the direction of Stephen L. Campbell.)

Fault detection and identification (FDI) is important in almost all real systems. Fault detection is the supervision of technical processes aimed at detecting undesired or unpermitted states (faults) and taking appropriate actions to avoid dangerous situations, or to ensure efficiency in a system. This dissertation develops and extends fault detection techniques for systems modeled by differential algebraic equations (DAEs).

First, a passive, observer-based approach is developed and linear filters are constructed to identify faults by filtering residual information. The method presented here uses the least squares completion to compute an ordinary differential equation (ODE) that contains the solution of the DAE and applies the observer directly to this ODE. While observers have been applied to ODE models for the purpose of fault detection in the past, the use of observers on completions of DAEs is a new idea. Moreover, the resulting residuals are modified requiring additional analysis. Robustness with respect to disturbances is also addressed by a novel frequency filtering technique.

Active detection, as opposed to passive detection where outputs are passively monitored, allows the injection of an auxiliary control signal to test the system. These algorithms compute an auxiliary input signal guaranteeing fault detection, assuming bounded noise. In the second part of this dissertation, a novel active detection approach for DAE models is developed by taking linear transformations of the DAEs and solving a bi-layer optimization problem. An efficient real-time detection algorithm is also provided, as is the extension to model uncertainty. The existence of a class of problems where the algorithm breaks down is revealed and an alternative algorithm that finds a nearly minimal auxiliary signal is presented. Finally, asynchronous signal design, that is, applying the test signal on a different interval than the observation window, is explored and discussed.

Fault Detection in Differential Algebraic Equations

by
Jason Roderick Scott

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina

2015

APPROVED BY:

Hien T. Tran	Ralph C. Smith
Negash G. Medhin	Stephen L. Campbell Chair of Advisory Committee

DEDICATION

In honor and memory of my parents.

BIOGRAPHY

Jason R. Scott was born in Erie, PA; grew up in Jamestown, NY; and graduated from Geneseo University in Geneseo, NY, in 2009 with a BA in Mathematics. He moved to Raleigh, NC that same year to pursue a doctorate in mathematics. He received an MS in Applied Mathematics from North Carolina State University in 2011. He spent summers interning at Sandia National Laboratories in Albuquerque, NM and M&T Bank in Buffalo, NY.

ACKNOWLEDGEMENTS

I would like to acknowledge

- Dr. Stephen L. Campbell, with whom it was a pleasure working with;
- Dr. Robert Martin, for great guidance and very enjoyable chats;
- my committee members and my graduate school representative for their valuable input and flexible scheduling;
- Nichole, for her encouragement and support;
- and all of my friends and colleagues that helped me along the way.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Fault detection and identification (FDI) are important in almost all real systems. Fault detection is the supervision of technical processes aimed at detecting undesired or unpermitted states and taking appropriate actions to avoid dangerous situations, or to ensure efficiency in a system. Deviations from nominal behavior that are large enough to cause undesired states, even in robustly controlled systems, are known as faults. The task of fault identification is to determine various characteristics of the fault, such as its type, size, or location.

The simplest and most common form of fault detection is known as limit checking [43]. Measured variables are checked with regard to tolerances, and alarms are generated for the operator. Tolerances are determined from compromises between detection size and unnecessary alarms. The simplicity of limit checking makes it an attractive option in many systems. Limit checking is effective in systems that operate approximately in a steady state, but may be inaccurate if the system is rapidly changing. Limit checking may also be inadequate in the case of a gradually increasing (incipient) fault or in systems under closed loop feedback because reaction occurs only after a relatively large change. In either case, faults may be masked until the situation is catastrophic.

Another option is the use of hardware redundancy, where measurements from multiple sensors are compared with each other and a voting mechanism is employed to detect faults. However, some applications may not permit the use of redundant sensors due to the extra cost, weight, volume, etc., they require. In other situations, such as with actuators, direct measurements are often not possible or are prohibitively expensive, either practically or financially. Therefore, advanced methods of fault detection have been developed with the goal of early, reliable detection.

This thesis focuses on model-based approaches, i.e., approaches where mathematical models are assumed to capture the nominal behavior of the system. The use of parity equations [41, 59] and parameter estimation [4, 38] for this purpose are common approaches. However, this

work investigates observer-based approaches [33, 42, 70] and multiple-model detection [20].

Model-free approaches also exist, but are not the subject of this work [48]. Model-free approaches are generally data driven and depend on statistical techniques to determine if a fault has occurred. For example, anomaly detection uses data to find patterns that do not conform to expected behavior [23]. Descriptive statistics are estimated from data generated by a system operating nominally. Then, given new data from a similar system whose operation is being tested, it is possible to calculate the probability of the new data coming from a nominal system. One advantage to statistically driven approaches is the ease with which they can deal with stochastic noise as opposed to the deterministic noise in this work. However, statistical approaches often require further investigation once an anomaly is flagged. Model-based approaches, especially those in this work, often yield much more information upon detection of a fault. Some sophisticated fault detection approaches use a combination of model-based and model-free approaches. For a comprehensive overview of these approaches, both model-based and model-free, see [44].

Every FDI method can be classified as either passive or active. In the passive approach, measurements from the system are continuously monitored and compared to the normal behavior of the system in some way. Most work in FDI is based in this class of methods. The observer-based approach in Chapter 3 is passive. Passive approaches are particularly useful for systems where auxiliary signals may pose a safety risk. The main disadvantage is that robust control schemes may hide failures during normal operation.

Active methods, on the other hand, interact with the system to improve detection. Usually, a test signal, constructed to highlight faults, is fed into the system, either on a test interval, or on a periodic basis. We call this test signal the auxiliary signal and its construction is the subject of the early sections in Chapter 4. An elementary example of an auxiliary signal is checking the brakes of a car while driving down a road, before they are needed. The auxiliary signal is usually determined in advance, based on the given system, and is constructed with the specific intent of exposing faults. In some scenarios, a sequence of test signals is applied, each one aimed at exposing a certain fault, or group of faults.

1.1 Differential Algebraic Equations

This dissertation develops new, or in some cases extends, fault detection techniques for systems modeled by differential algebraic equations (DAEs). A system $F(x'(t), x(t), u(t), t) = 0$ is, in general, a system of DAEs. If $F_{x'(t)}$ is nonsingular, then it is a system of ordinary differential equations (ODEs), a class of equations subsumed by the class of DAEs. The methods and approaches herein can be applied toward ODEs; however, they were developed with more complicated DAEs, known as higher index, DAEs in mind. The notion of index as it pertains to a DAE will be discussed later in this introduction. To avoid an incessant use of "higher index,"

from this point forward we will use the term "DAE" in place of "higher index DAE" to indicate a set of differential equations which are not an ODE.

DAEs arise naturally in many applications, especially in applications with a need for fault detection and identification. Some examples include electrical circuits, trajectory prescribed path control, systems of rigid bodies, problems in constrained mechanics, electrical networks and chemical reactions [10]. A classical example of a DAE arising from a constrained mechanical system with position x, velocity v , kinetic energy $T(x, v)$, external force $f(x, v, t)$ and constraint $\phi(x) = 0$ is

$$\frac{\partial^2 T}{\partial v^2} v' = g(x, v, t) + G^T \lambda \tag{1.1a}$$

$$x' = v \tag{1.1b}$$

$$0 = \phi(x), \tag{1.1c}$$

where $G = \frac{\partial \phi}{\partial x}$, and λ is the Lagrange multiplier. The constraints (1.1c) make this system a DAE, not an ODE, even if $\frac{\partial^2 T}{\partial v^2}$ is invertible. If it is invertible, multiplication of (1.1a) by $\frac{\partial^2 T}{\partial v^2}^{-1}$ converts (1.1) into a semi-explicit DAE.

DAEs differ from ODEs in several key aspects. The singularity conditions on $F_{x'}$ mean DAEs always contain pure algebraic equations called constraints like equation (1.1c). In terms of fault detection, this will be advantageous because it will allow additional residual information to be considered when looking for faults. However, not all constraints are given explicitly and some are even hidden constraints. Hidden constraints are only visible after differentiating the given equations. For example, consider the following semi-explicit DAE,

$$x_1' + x_3 = f_1 \tag{1.2a}$$

$$x_2' + x_1 = f_2 \tag{1.2b}$$

$$x_2 = f_3 \tag{1.2c}$$

where x_i are the unknowns and f_i are external forcing terms. The f_i may even be faults or disturbances in the problem. Equation (1.2c) is the only explicit algebraic constraint. However, a few differentiations and substitutions result in finding the only solution to this DAE,

$$x_1 = f_2 - f_3' \tag{1.3a}$$

$$x_2 = f_3 \tag{1.3b}$$

$$x_3 = f_1 - f_2' + f_3'', \tag{1.3c}$$

because there are two additional implicit algebraic constraints in the original DAE. Note the derivatives of the faults or forcing terms appear in the solution of the DAE. This is unlike the situation for ODEs and will have consequences for fault detection as we will see later (Section 3.6.2). Depending on the specific situation, the presence of fault derivatives can make the fault harder or easier to detect than in the analogous ODE case. If the f_i represent unavoidable disturbances as a result of normal operation of the system, (1.3) exposes the full impact that disturbances will have on the system. To be specific, even small disturbances will have large impacts on the system if their derivatives are large.

There are also numerous numerical difficulties associated with working with DAEs that are not present with ODEs [10]. The presence of algebraic constraints cause the solutions of a DAE to form a manifold called the solution manifold. Only the initial conditions that lie on the solution manifold accept a solution to the DAE. These are called consistent initial conditions. This characteristic of DAEs suggests that a they can be thought of as an ODE defined on a solution manifold. Therefore, numerically solving a DAE amounts to solving an ODE with constraints whether they be implicitly or explicitly defined.

1.2 Observer-Based Approach

In Chapter 3, we develop a passive FDI approach for systems modeled by DAEs based on the use of observers to estimate the true state of the system. The observer is used to generate an output error, also known as the residual, and faults are detected if the output error is far enough from zero. To illustrate, let us begin with the linear time invariant (LTI) ordinary differential equation (ODE) model with output

$$\begin{aligned} x'(t) &= Ax(t) + Bu(t) + f_p(t) \\ y &= Cx(t) + f_s(t) \end{aligned}$$

where $u(t)$ is the control, $f_p(t)$ is a process fault, and $f_s(t)$ is a sensor fault. If $f_p = f_s = 0$, then the system is operating normally. Suppose we design an observer, e.g., the standard Luenberger observer

$$\begin{aligned} \hat{x}'(t) &= A\hat{x}(t) + L(y(t) - \hat{y}(t)) + Bu(t) \\ \hat{y}(t) &= C\hat{x}(t), \end{aligned}$$

such that the dynamics of the state estimation error, $e(t) = x(t) - \hat{x}(t)$, are asymptotically stable. The dynamics of the state estimation error are

$$e'(t) = (A - LC)e + f_p(t) + Lf_s(t),$$

so in the LTI case, this simply means $A - LC$ is asymptotically stable. Then, in the absence of faults, the state error vanishes asymptotically

$$\lim_{t \to \infty} e(t) = 0.$$

Define the output error, or residual, to be $r(t) = y(t) - \hat{y}(t)$. $y(t)$ are the measured outputs from the system and $\hat{y}(t)$ are the outputs generated by the observer. Hence, both are available for our purposes. If there are no faults, $r(t)$ approaches zero as the state estimation error goes to zero. One way to make a decision on faults, is to create a problem dependent threshold, τ, and compare τ to $r(t)$. τ could be a vector of the same length as $r(t)$ or a positive number. If τ is a number one might say a fault is detected if $\|r(t)\| > \tau$. An example of a decision-making comparison when τ is a vector is the following: if $r_i(t) > \tau_i$, in any component i, then a fault is detected. The approach in Chapter 3 is a modified version of this approach, so that it is applicable to DAE systems.

Other previous work in observer-based fault detection in ODEs has also included the design of unknown input observers [24], dynamically extended observers [70], descriptor observers [33], and sliding mode observers [22, 47]. These works are applicable to linear ODE models. On the other hand, the authors of [34] and [73] construct observers for nonlinear systems, but a formal method of applying these observers to fault detection is absent.

1.3 Fault Detection in DAEs

Previous work on FDI in DAEs has included the idea of model-based fault diagnosis for discrete-time descriptor linear parameter varying (LPV) systems. LPV systems are a special case of LTV systems. The authors of [3] provide sufficient conditions to ensure the existence and the stability of the proposed observer by using a combined Lyapunov analysis based on a linear matrix inequalities formulation. While the LPV assumption may simplify the process of FDI in the DAE systems that meet this assumption, it does not lend itself to developing general methods. Our work on observer construction for FDI, while initially developed for LTI systems, is potentially applicable to general DAE systems including the general LTV and nonlinear cases.

Previous work, first in [71] and later refined in [26] (also developed independently and in a slightly different manner in [72]) constructs unknown input observers using a sliding mode approach for fault detection in DAE. In contrast to the approach in this work, observers are

directly applied to the DAE without any reduction or reformulation. This requires certain assumptions on the matrices of the LTI DAE that are different than our assumptions for the construction of an observer. In general, unknown input observers require the regularity of a certain matrix pencil [10], computation of more matrices than the one gain matrix commonly found in the design of observers for LTI ODE, and rely on eigenvalue placement techniques. These dependencies make it difficult to generalize these observers for LTV and nonlinear DAE systems. In addition, the authors of [30] introduce a descriptor unknown input observer with sliding modes. This introduces the added complexity of simulating a descriptor system in order to obtain state estimates.

However, an advantage of the sliding mode approach, in addition to the simplicity of not requiring any reformulation, is the ability to follow the behavior of the faulty system. The mechanism of detection is based on monitoring abnormal deviations of the controlled outputs from their the set point trajectories and deviations of the estimated parameters from their nominal values. Moreover, disturbances are easily decoupled with unknown input observers by assuming the fault and disturbance enter the system with different distribution matrices. In Section 3.4 we give an alternative approach based on the discrete Fourier transform that does not require this assumption.

Articles [39], [40], and [45] develop techniques for parameter estimation in DAE. Paired with a decision making threshold for the difference between parameter estimations and nominal values, parameter estimation can be made into a passive fault detection approach. While the estimates provide a powerful FDI technique, the approach in [40] and [45] also requires the solution of a nonlinear programming problem, certainly a nontrivial task, and potentially computationally expensive. While appropriate for some applications, the time-sensitive nature of fault detection may prohibit these methods. In general, parameter estimation for DAE has yet to be formalized for the specific purpose of FDI.

A large percentage of this thesis concerns itself with active detection in DAEs. Since previous work in active detection has only been developed for systems modeled by ODEs, the related methods in this thesis are pioneering ideas in this field. Due to the absence of competing methods, in the following we summarize previous active detection work for ODE systems.

In some ways, the auxiliary design methods developed in this thesis can be seen as the DAE analogs to the methods developed by the authors of [20] and [54] for ODEs. These works employ a multi-model approach, meaning there are at least two models, one for the nominally operating system, and the others for the faulty systems. In these works, dynamic optimization is used to find the smallest auxiliary signal that guarantees fault detection by constructing a small signal that forces the output sets to be disjoint, even in the presence of bounded noise. It is often desirable to find the smallest auxiliary signal, in the norm sense, so that the system is minimally affected during the test interval. The smallest auxiliary signal, which is the solution to

an optimal control problem, is computed offline, and then applied to the system. The measured outputs which, by construction, can only come from one of the models, are monitored during the test interval and a decision is made, possibly before the end of the test period. The approach has since been extended for detection of incipient faults [53], to include a priori information about the initial condition [52], the presence of model uncertainty [1], and for nonlinear systems [2], [68].

The authors of [61] develop an alternative method of constructing an auxiliary signal that is based on a moving horizon for discrete time systems. That is, instead of pre-computing the test signal, online measurements during the test interval are used to refine an open-loop input, initially computed offline. In general, moving horizon approaches have the potential to produce much more conservative inputs (e.g. reduced duration, norm, etc.). In [61], the initial optimal input is computed by solving an expensive mixed-integer quadratic program. Then, at each time step, the optimal input is recalculated by solving this problem using updated information from system measurements. The authors note the computation time required for this method makes it impractical for many applications of interest. To address this, they develop a much more efficient method by computing all possible reachable sets offline and storing them. An optimal input strategy is computed for each possible trajectory, stored in a table, and at each time step, the optimal signal for that time step is selected. The main drawback to this approach is that it requires very strict assumptions. The authors assume the matrix coefficient on the state in the output for each model is invertible and they restrict the set of possible noises. Even the basic method, which is too computationally expensive for online usage, requires restrictive assumptions including the following: the control lies in a convex polytope, the noises are zero-centered zonotopes, the initial states lie in zonotopes, and the faults are time invariant.

In [50], the author extends fault detection results from closed loop systems to open loop systems, as well as closed loop systems with feedback controllers. The open loop system is derived from the closed loop system by removing the feedback controller. A transfer function matrix is derived between the input and the residual vector that is equivalent to the fault signature matrix in the closed loop case. This shows that it is possible to use the same active fault detection method on the closed and open loop systems. The only restriction being the maximal number of faults occurring simultaneously is bounded by the number of control signals minus one.

1.4 Thesis Outline

Although work on FDI in systems modeled by DAE is not new–the approaches in previous paragraphs are such examples–there are relatively few works in the area, compared to ODE detection. In this work, we present FDI methods that have never before been applied to DAE.

In Chapter 2 we reformulate the DAE problem using the completion technique–described later in Section 2.2–and derive some properties related to it in Section 2.4. In Chapter 3 we use the completion to construct an observer, compute outputs based on the observer, and construct residuals to make decisions on fault detection. We also develop linear filters to identify faults and address disturbance attenuation using the discrete Fourier transform. We apply this method to a DAE circuit model and illustrate its benefit in Section 3.6.

In Chapter 4, we turn to the task of active detection. Analogous work on ODE models is done in [20]. Since DAE are singular, the standard techniques for FDI in the ODE case are not directly applicable to the DAE case. A reformulation of a bi-level optimization problem with DAE constraints is necessary to construct a problem that can be implemented using modern software. A novel theorem is presented to identify the active model in real-time. We also investigate some shortcomings of the method in Chapter 4 and present a suboptimal modification. Finally, we investigate a related topic in auxiliary signal design known as asynchronous active detection, a strategy where the output is watched on a different time interval than the interval the auxiliary signal is applied on. We show the result is a cheaper test signal and sometimes a shorter test interval.

Chapter 5 concludes with a list of the author's presentations, publications, and contributions. Areas of future work are also introduced.

Chapter 2

Background

In this chapter, we summarize the theory of completions, originally introduced in [14], and later matured in [13], [56], [57], [17], [18], and [55]. A completion of a DAE is an ODE whose solutions include those of the DAE. The specific type of completion studied here, and in the references herein, is known as the least squares completion, so named because after constructing a large set of equations, known as the derivative array, the state is solved in the least squares sense. The least squares completion has been developed for general DAEs with no necessary structural assumptions, originally as part of a general numerical solution technique. The generality of the method makes our FDI approach potentially applicable to systems modeled by general DAEs.

The completion procedure generates an ODE containing the solutions of the DAE. The extra solutions, known as the additional dynamics, have been shown to be stable, provided some care is taken [55]. In that case, the completion is known as the stabilized completion. As shown in Section 3.1, this enables us to construct a Luenberger observer for the completion and rely on its state estimates to generate residuals for fault diagnosis.

2.1 DAE Preliminaries

Recall, we are assuming the systems under consideration may be approximated by models of the form

$$F(x'(t), x(t), u(t), t) = 0, \tag{2.1}$$

where x is the state, u is the control, both F and x are vector valued, and $F_{x'}(x'(t), x(t), u(t), t)$ is singular. In what follows, we are concerned with the case where the solutions exist and are uniquely defined on the interval of interest. Unlike ODEs, not all initial values for x admit a smooth solution; those that do, are called consistent initial conditions. Intuitively, a system of DAEs is solvable if the system's solution is determined by a consistent initial condition. A more formal definition of solvability is the following:

Definition 2.1.1. *Let I be an open subset of $\mathcal{R}$, Ω a connected open subset of $\mathcal{R}^{2m+1}$, and F a differentiable function for Ω to $\mathcal{R}^m$. Then the DAE (2.1) is solvable on I in Ω if there is an r-dimensional family of solutions $\phi(t,c)$ defined on a connected open set $I \times \tilde{\Omega}$, $\tilde{\Omega} \subset \mathcal{R}^r$, such that*

1. *$\phi(t,c)$ is defined on all of I for each $c \in \tilde{\Omega}$*
2. *$(\phi_t(t,c), \phi(t,c), t) \in \Omega$ for $(t,c) \in I \times \tilde{\Omega}$*
3. *If $\psi(t)$ is any solution with $(\psi'(t), \psi(t), t) \in \Omega$, then $\psi(t) = \phi(t,c)$ for some $c \in \tilde{\Omega}$*
4. *The graph of ϕ as a function of (t,c) is an $(r+1)$-dimensional manifold.*

The definition says that there is a local, r-dimensional family of continuous solutions with no bifurcations, on the interval I. The definitions and theorems in this section, including the previous definition, can be found in [10].

A useful property when discussing completions is the index of the DAE. The index plays a key role in the classification and behavior of DAEs. In order to motivate the definition that follows, consider the special case of a semi-explicit DAE

$$x' = f(x,y,t) \tag{2.2a}$$
$$0 = g(x,y,t). \tag{2.2b}$$

If we differentiate the constraint equation (2.2b) with respect to t we get

$$x' = f(x,y,t) \tag{2.3a}$$
$$g_x(x,y,t)x' + g_y(x,y,t)y' = -g_t(x,y,t). \tag{2.3b}$$

If g_y is nonsingular, the system (2.3) is an implicit ODE and we say that (2.2) has index one. If this is not the case, suppose with coordinate changes we can rewrite (2.3) as (2.2), but with different x, y. Then, we repeat the differentiation and coordinate changes until we can get to an ODE. The number of differentiations required in this procedure is known as the index. For example, a DAE of index zero is an ODE.

Definition 2.1.2. *The index, k, of a system of DAEs is the minimum number of times the system is differentiated with respect to t in order to uniquely determine the first derivative of the state vector as a continuous function of the state vector and t.*

Suppose we differentiate (2.2) with respect to t, k times, where k is the index. Then we have the $k+1$ equations

$$F = 0 \tag{2.4a}$$

$$\frac{d}{dt}F = 0 \tag{2.4b}$$

$$\vdots$$

$$\frac{d^k}{dt^k}F = 0, \tag{2.4c}$$

which together are known as the derivative array and are denoted by $G(z, x, t)$, where

$$z(t) = \begin{bmatrix} x'(t) \\ \vdots \\ x^{(k+1)}(t) \end{bmatrix}$$

For a linear DAE, such as

$$E(t)x'(t) + F(t)x(t) = B(t)u, \tag{2.5}$$

the derivative array is

$$\mathcal{E}(t)z(t) + \mathcal{F}(t)x(t) = \mathcal{B}(t)\bar{u}(t), \tag{2.6}$$

where

$$\mathcal{E} = \begin{bmatrix} E & 0 & 0 & \dots & 0 \\ E' + F & E & 0 & \dots & 0 \\ E'' + 2F' & 2E' + F & E & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ . & . & . & \dots & . \end{bmatrix}, \quad \mathcal{F} = \begin{bmatrix} F \\ F' \\ F'' \\ \vdots \\ F^{(k)} \end{bmatrix}.$$

$\mathcal{B} = \text{diag}(B, B, \dots, B)$ and $\bar{u}$ defined in an analogous way to $\mathcal{F}$, except for vectors. Upon consideration of the derivative array, we have the following convenient definition:

Definition 2.1.3. *The index, k, of a linear system of DAEs $E(t)x'(t) + F(t)x(t) = B(t)u(t)$ is the minimum number of times the system is differentiated with respect to t so that the coefficient matrices $\mathcal{E}(t)$, $\mathcal{F}(t)$ from the derivative array equations $\mathcal{E}(t)w(t) + \mathcal{F}(t)x(t) = \mathcal{B}(t)\bar{u}(t)$ meet the following conditions:*

1. $\begin{bmatrix} \mathcal{E}(t) & \mathcal{F}(t) \end{bmatrix}$ *is full row rank for all t;*
2. $\mathcal{E}(t)$ *has constant rank; and*
3. *if $x(t)$ is n dimensional, then $\mathcal{E}(t)b(t) = 0$ for some vector $b(t)$ implies the first n entries for $b(t)$ are zero for all t.*

Definition 2.1.3, a specific version of Definition 2.1.2 for linear systems, includes checks easily implemented in an index-determining algorithm for time-invariant systems.

2.2 Completion Theory

A completion of

$$E(t)x'(t) + F(t)x(t) = B(t)u(t) + f_p(t), \tag{2.7}$$

a linear system of DAEs, with possible faults f_p, is a linear system of ODEs

$$x'(t) = \hat{A}(t)x(t) + \hat{B}(t)\bar{u}(t) + \hat{G}(t)\bar{f}_p \tag{2.8}$$

designed so the solutions of (2.8) contain those of (2.7). For any vector function q, we use $\bar{q}$ to denote a vector containing q and the first $k-1$ derivatives of q with respect to t. As discussed in the introduction, for an n-dimensional system of linear DAEs, solutions are defined on the less then n-dimensional solution manifold. Since (2.8) is an n-dimensional ODE, there exists an n-dimensional family of solutions. Therefore, there are extra solutions whose behavior will be discussed in what follows.

Recall we denote the derivative array, given by (2.4), as $G(z, x, t)$. We will need the following definition [12]:

Definition 2.2.1. *A system of algebraic equations*

$$A \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = b \tag{2.9}$$

is called 1*-full with respect to* x_1*, if* x_1 *is uniquely determined for any consistent* b*.*

Suppose the following assumptions are satisfied for both k and $k+1$ in some neighborhood:

I. Sufficient smoothness of $G = 0$.

II. $G = 0$ is consistent as an algebraic equation.

III. G_z is 1-full and has constant rank independent of (z, x, t).

IV. $\begin{bmatrix} G_z & G_x \end{bmatrix}$ has full row rank independent of (z, x, t).

When these assumptions are met, then multiplication by the Moore-Penrose inverse [19], $\mathcal{E}^\dagger$, solves uniquely for x'

$$x'(t) = -\mathcal{E}^\dagger \mathcal{F} x + \mathcal{E}^\dagger \mathcal{B}\bar{u} + \mathcal{E}^\dagger \bar{f}_p$$

and determines the least squares completion (2.8) where $\hat{A}$, $\hat{B}$, and $\hat{G}$ are the first block row of $-\mathcal{E}^\dagger\mathcal{F}$, $\mathcal{E}^\dagger\mathcal{B}$, and $\mathcal{E}^\dagger$, respectively.

Assumptions I.-IV. also imply the geometric solvability of the linear DAE [16], a notion related to the solvability definition found in this thesis. Having no need for geometric solvability here, for our purposes these assumptions are only required for the calculation of the completion.

Thus far, to form the derivative array, we differentiate the system of DAEs k times, where k is the index of the DAE. However, it has been shown in [56] the additional dynamics may be unstable, leading to numerical inaccuracies. Therefore, we modify the approach by applying the differential polynomial $\mathcal{D} = \frac{d}{dt} + \Lambda$ instead, where $\mathrm{Re}(s) > 0$ for all eigenvalues s of the matrix Λ. The authors of [17] have shown this modification stabilizes the additional dynamics and guarantees no repeated eigenvalues in the additional dynamics. Repeated eigenvalues make construction of observers problematic unless there are sufficient outputs.

As a first step in fault detection, this thesis considers LTI DAEs

$$Ex' + Fx = Bu + f_p. \tag{2.10}$$

Therefore, in the following, we demonstrate the construction of the derivative array and calculation of the completion using $\mathcal{D}$ on LTI systems. Applying $\mathcal{D}$ to (2.10) k times results in the modified derivative array

$$D_j \mathcal{E} \begin{bmatrix} x' \\ \omega \end{bmatrix} = -D_j \mathcal{F} x + D_j \mathcal{B} \bar{u} + D_j \bar{f}_p \tag{2.11}$$

where

$$\mathcal{E} = \begin{bmatrix} E & 0 & 0 & \dots & 0 \\ F & E & 0 & \dots & 0 \\ 0 & F & E & \dots & 0 \\ \vdots & & & & \\ 0 & 0 & \dots & F & E \end{bmatrix}, \; D_j = \begin{bmatrix} I & 0 & 0 & \dots & 0 \\ \Lambda & I & 0 & \dots & 0 \\ \Lambda^2 & 2\Lambda & I & \dots & 0 \\ \vdots & & & \ddots & \\ \Lambda^k & k\Lambda^{k-1} & & \dots & I \end{bmatrix},$$

$$\mathcal{F} = \begin{bmatrix} F^T & 0 & 0 & \dots & 0 \end{bmatrix}^T, \text{ and } \mathcal{B} = \mathrm{diag}\{B, B, \dots, B\}.$$

$D_j \mathcal{E}$ is rank deficient. For many classes of DAEs, such as the Hessenberg DAEs of mechanics, $\hat{B}\bar{u}$ involves none or one derivative of u.

Suppose assumptions I.-IV. are satisfied for both k and $k+1$ in some neighborhood. Then solving (2.11) in the least squares sense we get

$$\hat{z} = -\mathcal{E}^\dagger D_j \mathcal{F} x + \mathcal{E}^\dagger D_j \mathcal{B} \bar{u} + \mathcal{E}^\dagger D_j \bar{f}_p. \tag{2.12}$$

If k is at least the index of the DAE, then (2.12) uniquely determines the x' component. The

other components of $\hat{z}$ will usually not be the higher derivatives of x and will be ignored. Thus taking the first block row of (2.12) we get the least squares completion

$$x' = \hat{A}x + \hat{B}\bar{u} + \hat{G}\bar{f}_p. \tag{2.13}$$

The constraints

$$0 = Gx + \tilde{B}\bar{u} + \tilde{G}\bar{f}_p \tag{2.14}$$

are found by multiplying (2.11) by a maximal rank left annihilator U of $D_j\mathcal{E}$. We partition $\hat{G}$ and $\tilde{G}$ conformal with $\bar{f}_p$ so that $\hat{G} = \left[\hat{G}_0, \ldots, \hat{G}_{k-1}\right]$. If f_p is constant, then $\bar{f}_p = \left[f_p^T, 0, \ldots, 0\right]^T$ so that only $\hat{G}_0$ is needed. The $\tilde{G}_i$ notation is similar. The constant fault case happens often enough to warrant this consideration.

2.3 LTI Systems

LTI systems have some properties that we will exploit in the following sections. First, we will need the following definitions and theorems.

Definition 2.3.1. *The matrix pencil for the linear time invariant system of DAEs in* (2.10) *is* $sE + F$ *for a complex scalar* s*.*

Definition 2.3.2. *The finite generalized eigvenvalues of the system in* (2.10) *are all* s *such that* $\det(sE + F) = 0$*.*

Finite generalized eigenvalues play the same role for (2.10) that regular eigenvalues do for LTI ODEs. That is, if $u(t) = 0$ and λ is one finite generalized eigenvalue of (2.10), then one mode of the solution is $e^{\lambda t}$.

Definition 2.3.3. *If the determinant of* $sE + F$*, denoted* $\det(sE + F)$ *is not identically zero as a function of* s*, then the pencil is said to be regular.*

In general, the solvability of a general DAE can be difficult to determine; however, for (2.10) there is a nice characterization. The next theorem explores the relationship between the solvability of the LTI DAE and its matrix pencil.

Theorem 2.3.1. *The linear constant coefficient DAE is solvable if and only if* $sE + F$ *is a regular pencil.*

Recall, a matrix N is said to have nilpotency k if $N^k = 0$ and $N^{k-1} \neq 0$. The following technical theorem, from [32], provides a structure we will exploit for failure detection:

Theorem 2.3.2. *Suppose that $sE+F$ is a regular pencil. Then there exist nonsingular matrices P, Q such that*

$$PEQ = \begin{bmatrix} I & 0 \\ 0 & N \end{bmatrix}, \quad PFQ = \begin{bmatrix} C & 0 \\ 0 & I \end{bmatrix} \tag{2.15}$$

where N is a matrix of nilpotency k and I is an identity matrix. If $N = 0$, then define $k = 1$. In the special case that E is nonsingular, we take $PEQ = I$, $PFQ = C$, and define $k = 0$. If $\det(sE + F)$ *is identically constant, then* (2.15) *simplifies to $PEQ = N$, $PFQ = I$.*

The degree of nilpotency, k, is the same as the index of the DAE.

2.4 Useful Facts about Completions

Given a LTI DAE the least squares completion is a LTI ODE with LTI constraints. This section gives some useful results about the matrices computed by the completion procedure. In the following discussion we apply a similarity transformation to arrive at a simpler DAE. We will prove certain properties that completions of this simpler DAE possess and extend them to the completion of (2.10) via their relationship by a similarity transformation.

Suppose the matrix pencil $sE + F$ is regular. Then, by Theorem 2.3.2, there exists an orthogonal P and a nonsingular Q such that

$$PEQ = \begin{bmatrix} C_1 & C_2 \\ 0 & N \end{bmatrix}, \quad PFQ = \begin{bmatrix} D_1 & D_2 \\ 0 & D_3 \end{bmatrix} \tag{2.16}$$

where C_1 and D_3 are invertible and N is nilpotent of the same degree as the index of (2.10). Therefore, left multiplication by the orthogonal matrix P and the coordinate change given by $x = Q \begin{bmatrix} y_1 \\ z_2 \end{bmatrix}$ transforms the DAE into

$$\begin{aligned} C_1 y_1' &= -C_2 z_2' + D_1 y_1 + D_2 z_2 + f_1 \\ N z_2' &= D_3 z_2 + f_2. \end{aligned}$$

where $D_3^{-1}N$ is nilpotent. Then, using the transformation $y_2 = D_3^{-1} z_2$ and relabeling coefficients, we obtain the system

$$C_1 y_1' = -C_2 y_2' + D_1 y_1 + D_2 y_2 + f_1 \tag{2.17a}$$

$$N y_2' = y_2 + f_2. \tag{2.17b}$$

Note that (2.16) is not the usual similarity form of a regular pencil. We must impose the

additional restriction that P is orthogonal instead of just nonsingular to obtain a relationship between the completions of (2.10) and (2.17).

Since C_1 is nonsingular, (2.17a) is index zero. Then, the completion of (2.17) is (2.17a) and the completion of (2.17b). The following proposition [57], will be helpful in finding the completion of (2.17b).

Proposition 2.4.1. *Suppose the LTI DAE in* (2.10) *is solvable with index k and its derivative array is given in* (2.11). *Suppose that assumptions I.-IV. are satisfied for this system. Let* G_0 *be an* $n \times (k+1)n$ *matrix satisfying* $G_0 D_j \mathcal{E} = \begin{bmatrix} I & 0 & \dots & 0 \end{bmatrix}$ *and* $G_0 Z = 0$, *where* Z *is a matrix of maximal rank satisfying* $Z^T D_j \mathcal{E} = 0$. *Namely, the columns of* Z *form a basis for the null space* $N(\mathcal{E}^T D_j^T)$. *Then, the least squares completion of* (2.10) *defined by* (2.11) *is* $x' = G_0 D_j(-\mathcal{F}x + \mathcal{B}\bar{u} + \bar{f}_p)$.

In particular, this means given $D_j \mathcal{E} \begin{bmatrix} y_2' \\ \omega \end{bmatrix} = D_j(-\mathcal{F}y_2 + \bar{f}_2)$, the derivative array of (2.17b), we get the completion

$$y_2' = -G_0 D_j \mathcal{F} y_2 + G_0 D_j \bar{f}_2 = \hat{A}_2 y_2 + \hat{G}_2 \bar{f}_2. \tag{2.18}$$

If the fault is constant, we look at $\hat{G}_{2_0} = G_0 D_0$ where D_0 is the first block column of D_j. Note that in this case $F = -I$, so $-G_0 D_j \mathcal{F} = G_0 D_0$ and $\hat{A}_2 = \hat{G}_{2_0}$. Therefore $\hat{G}_{2_0}$ is invertible since $\hat{A}_2$ is invertible from [57]. We will need this fact later.

Proposition 2.4.1 leads us to the following

Lemma 2.4.1. *The stabilized least squares completion for* (2.17b) *is given by* (2.18) *and the matrices in* (2.18) *satisfy* $G_2 \hat{A}_2^{-1} \hat{G}_{2_0} - \tilde{G}_{2_0} = 0$.

Proof. We know that (2.18) is the least squares completion of (2.17b) and $\hat{A}_2$ is invertible. The lemma now follows because

$$G_2 \hat{A}_2^{-1} \hat{G}_{2_0} - \tilde{G}_{2_0} = U \begin{bmatrix} I \\ \lambda I \\ \lambda^2 I \\ \vdots \end{bmatrix} I - U \begin{bmatrix} I \\ \lambda I \\ \lambda^2 I \\ \vdots \end{bmatrix} = 0.$$

□

Equation (2.17a) is index zero so it is invariant under the least squares completion process. Thus, there are no constraints associated with (2.17a). Using the truncated versions of $\hat{G}$ and $\tilde{G}$, the completion for (2.17) is

$$y' = \hat{A}_1 y + \hat{G}_{1_0} f \tag{2.19a}$$

$$0 = G_1 y + \tilde{G}_{1_0} f \tag{2.19b}$$

where $G_1 = \begin{bmatrix} 0 & G_2 \end{bmatrix}$, $\tilde{G}_{1_0} = \begin{bmatrix} 0 & \tilde{G}_{2_0} \end{bmatrix}$ and

$$\hat{A}_1 = \begin{bmatrix} C_1^{-1} D_1 & * \\ 0 & \hat{A}_2 \end{bmatrix}, \ \hat{G}_{1_0} = \begin{bmatrix} * & * \\ 0 & \hat{G}_{2_0} \end{bmatrix} \tag{2.20}$$

letting $*$ denote nonzero blocks. The next lemma extends Lemma 2.4.1 to the entire completion of (2.17). In the proof of the next lemma, we assume 0 is not a finite eigenvalue of (2.10) since we will subsequently need to assume this for the purposes of fault detection.

Lemma 2.4.2. *The matrices in* (2.19) *satisfy* $G_1 \hat{A}_1^{-1} \hat{G}_{1_0} - \tilde{G}_{1_0} = 0$.

Proof. C_1 is invertible by assumption. $\hat{A}_2$ is invertible from [57]. Then $\hat{A}_1$ is invertible as defined in (2.20) since 0 is not a finite eigenvalue of (2.10). Consider $G_1 \hat{A}_1^{-1} \hat{G}_{1_0} - \tilde{G}_{1_0}$. This is equivalent to

$$\begin{bmatrix} 0 & G_2 \end{bmatrix} \begin{bmatrix} * & * \\ 0 & \hat{A}_2^{-1} \end{bmatrix} \begin{bmatrix} * & * \\ 0 & \hat{G}_{2_0} \end{bmatrix} - \begin{bmatrix} 0 & \tilde{G}_{2_0} \end{bmatrix} = \begin{bmatrix} 0 & G_2 \hat{A}_2 \hat{G}_{2_0} - \tilde{G}_{2_0} \end{bmatrix}.$$

An application of Lemma 2.4.1 completes the proof. □

Finally, we connect the transformed system in (2.17) and the original system (2.10) with the following

Lemma 2.4.3. *Consider the original system* (2.10) *and its stabilized least squares completion* (2.13)–(2.14). *Then* $G\hat{A}^{-1}\hat{G}_0 - \tilde{G}_0 = 0$.

Proof. In [57], it was shown that given the original system and its completion, the corresponding completion of (2.17) is

$$y' = Q^{-1} \hat{A} Q y + Q^{-1} \hat{G}_0 \bar{f} \tag{2.21a}$$

$$0 = GQy + \tilde{G}_0 \bar{f} \tag{2.21b}$$

if we redefine $Q = Q \begin{bmatrix} I & 0 \\ 0 & D_3^{-1} \end{bmatrix}$. Lemma 2.4.2 implies that $(GQ)(Q^{-1}\hat{A}^{-1}Q)(Q^{-1}\hat{G}_0) - \tilde{G}_0 = 0$. The current lemma follows because $(GQ)(Q^{-1}\hat{A}^{-1}Q)(Q^{-1}\hat{G}_0) - \tilde{G}_0 = G\hat{A}^{-1}\hat{G}_0 - \tilde{G}_0$. □

Recall $\hat{G}_{2_0}$ is invertible. We can now prove the following

Lemma 2.4.4. $\hat{G}_0$ *is invertible.*

Proof. Since (2.17a) is invariant under the stabilized least squares process we have $\hat{G}_{1_0} = \begin{bmatrix} C_1^{-1} & -C_1^{-1}C_2\hat{G}_{2_0} \\ 0 & \hat{G}_{2_0} \end{bmatrix}$ which is invertible. Equation (2.21) implies $\hat{G}_0 = Q\hat{G}_{1_0}$, a product of invertible matrices. □

We will use these results later in Chapter 3.

Chapter 3

Observer-Based Passive FDI

3.1 Observer Construction

As a first step in fault detection, this thesis considers LTI DAEs with outputs and possible process and sensor faults, f_p and f_s,

$$Ex' + Fx = Bu + f_p \tag{3.1a}$$

$$y = Hx + Du + f_s. \tag{3.1b}$$

We assume E, F are square matrices, E may be singular, u is a known control input, f_p is a process fault, f_s is a sensor fault, there is a scalar s for which $sE + F$ is invertible—that is, $\{E, F\}$ form a regular matrix pencil so that (3.1a) is solvable—x and u are functions of time t, x is $n \times 1$, y is $m \times 1$, and 0 is not a finite eigenvalue of the pencil $sE + F$. If we also assume assumptions I.-IV. are met, then we can compute the stabilized least squares completion with output,

$$x' = \hat{A}x + \hat{B}\bar{u} + \hat{G}\bar{f}_p \tag{3.2a}$$

$$0 = Gx + \tilde{B}\bar{u} + \tilde{G}\bar{f}_p \tag{3.2b}$$

$$y = Hx + Du + f_s. \tag{3.2c}$$

We construct a Luenberger observer that is unaware of the faults and ignores (3.2b) so that it takes the form

$$\hat{x}' = \hat{A}\hat{x} + L(y - \hat{y}) + \hat{B}\bar{u} \tag{3.3a}$$

$$\hat{y} = H\hat{x} + Du, \tag{3.3b}$$

where the eigenvalues of $\hat{A} - LH$ are designed to give us the desired convergence rate of the observer. If $\{\hat{A}, H\}$ is not completely observable, then we need the unobservable eigenvalues to have negative real part. If they are not finite eigenvalues of the matrix pencil $sE + F$, then they are from the stabilization parameter matrix Λ and are user specifiable. Observability of (3.1) does not imply the observability of (3.3) unless (3.2b) is viewed as an extra output. Such reduced order observers are considered in [7, 8] but are outside the scope of this work.

Let $e = x - \hat{x}$ be the observer error. Then,

$$e' = (\hat{A} - LH)e - Lf_s + \hat{G}\bar{f}_p \tag{3.4a}$$

$$y - \hat{y} = He + f_s. \tag{3.4b}$$

Using our outputs, observer estimate $\hat{x}$, and information on the solution manifold (3.2b), we have two residual vectors available to us instead of the usual one,

$$r_1 = y - \hat{y} \tag{3.5a}$$

$$r_2 = G\hat{x} + \tilde{B}\bar{u}. \tag{3.5b}$$

Both of these residuals should be zero if the observer has converged (that is, the estimation error is less than some tolerance) and there are no faults. We shall assume the observer error bound takes the form,

$$\|h(t)\| \leq \beta e^{\alpha(t-t_0)} + \epsilon = \theta(t), \tag{3.6}$$

for $\beta > 0, \epsilon > 0$, and $\alpha < 0$, where t_0 is the start time of the fault. Here α comes from the theoretical convergence rate of the observer and ϵ models numerical, measurement, and other small but nonzero errors. The constant β is a bound on the size of disturbances due to either the onset of the fault or by other disturbances to the system. θ is to simplify some expressions later.

3.2 Detection

As is often done in the literature, we assume that once fully developed the fault is constant and in a particular direction. As we will see later, with DAEs, the start up of the fault can play a larger role in detection than in the case of ODE systems. Certain faults can be more easily detected if they occur quickly as shown with the circuit example in Section 3.6. Thus, we assume faults take the form

$$f_i(t) = \kappa g(t) d_i, \;\; i = p, s.$$

$g(t)$ is a monotonically non-decreasing, smooth function that is zero for $t \leq t_0$ and one for $t \geq t_1$. On the interval $[t_0\ t_1]$, it takes the form $g(t) = c(t-t_0)^2 + d(t-t_0)^3$. Constants c and d are determined with the conditions $g(t_1) = 1$ and $g'(t_1) = 0$. If $t_0 = t_1$ then the fault is a piecewise constant, abrupt fault and there is no transient period. Otherwise, the fault appears in a smooth manner and is called a ramp fault. d_i is a constant vector and κ is a scaling constant allowing us to discuss thresholds and detection times in terms of the size of the fault.

Note the smoothness of the fault is only assumed for purposes of analysis and in order to simulate the fault. In practice, the fault does not appear in the observer equation and we use the real output y rather than a simulated output. Hence, we can detect most faults regardless of their regularity.

We assume that $\hat{A} - LH$ is invertible. The only time this might not be true in a case of interest is if we were designing dead beat observers for a discrete time system. This case will be left for future work.

Using (3.2b), if we consider $r_2 - 0$, we get

$$r_1 = He + f_s \tag{3.7a}$$
$$r_2 - 0 = r_2 - (Gx + \tilde{B}\bar{u} + \tilde{G}\bar{f}_p) \tag{3.7b}$$
$$r_2 = -Ge - \tilde{G}\bar{f}_p. \tag{3.7c}$$

After the initial ramp up interval of the fault and after the observer has converged, we have $e(t) = -(\hat{A} - LH)^{-1}(-Lf_s + \hat{G}\bar{f}_p)$. The residuals can be expressed

$$r_1 = -H(\hat{A} - LH)^{-1}(-Lf_s + \hat{G}\bar{f}_p) + f_s + Hh$$
$$r_2 = G(\hat{A} - LH)^{-1}(-Lf_s + \hat{G}\bar{f}_p) - \tilde{G}\bar{f}_p - Gh$$

or the more condensed

$$W \begin{bmatrix} \bar{f}_p \\ f_s \end{bmatrix} + \begin{bmatrix} H \\ -G \end{bmatrix} h(t) = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}. \tag{3.9}$$

Let $f = \begin{bmatrix} \bar{f}_p^T & f_s^T \end{bmatrix}^T$ and denote (3.9) by

$$Wf + Qh(t) = r, \tag{3.10}$$

where W and W_0 (to be used later) are

$$W = \begin{bmatrix} -H(\hat{A} - LH)^{-1}\hat{G} & H(\hat{A} - LH)^{-1}L + I \\ G(\hat{A} - LH)^{-1}\hat{G} - \tilde{G} & -G(\hat{A} - LH)^{-1}L \end{bmatrix} \tag{3.11}$$

$$W_0 = \begin{bmatrix} -H(\hat{A} - LH)^{-1}\hat{G}_0 & H(\hat{A} - LH)^{-1}L + I \\ G(\hat{A} - LH)^{-1}\hat{G}_0 - \tilde{G}_0 & -G(\hat{A} - LH)^{-1}L \end{bmatrix} \tag{3.12}$$

and $f_0 = \begin{bmatrix} d_p^T & d_s^T \end{bmatrix}^T$, where d_p and d_s are constant vectors determining the directions of the process and sensor faults, respectively.

Note that W and W_0 are partly determined by the original system, partly by the stabilization of the additional dynamics, and partly by the choice of L used in the design of the observer. If $W_0 \neq 0$, then generically most faults can be detected assuming the fault is large enough. However, we want more detailed information on the size of the fault κ and the time to detection.

There are different ways to evaluate residuals. One common way to do so is with element residual thresholds. These can be taken asymmetric but here we will say a fault is to be detected if $|r_i| > \tau_i$ for any i^{th} component of the residual vector r and the user chosen threshold vector τ which has positive entries. If a sufficiently large fault occurs and $t > t_1$, then there is at least one i such that

$$|\kappa[W_0 f_0]_i + [Qh(t)]_i| > \tau_i. \tag{3.13}$$

Proposition 3.2.1. *Suppose $h(t)$ is bounded according to* (3.6)*, $t > t_1$ and*

$$\kappa > \frac{\tau_i + \|Q\|\beta e^{-\alpha(t-t_0)} + \|Q\|\epsilon}{|[W_0 f_0]_i|} \tag{3.14}$$

for time t and some index i. Then (3.13) *holds and thus, the fault f can be detected by time t.*

Proof. Assumption (3.14) is equivalent to $|\kappa[W_0 f_0]_i| > \tau_i + \|Q\|\beta e^{-\alpha(t-t_0)} + \|Q\|\epsilon$. Therefore, either

$$\kappa[W_0 f_0]_i > \tau_i + \|Q\|\beta e^{-\alpha(t-t_0)} + \|Q\|\epsilon, \text{ or} \tag{3.15a}$$

$$\kappa[W_0 f_0]_i < -\tau_i - \|Q\|\beta e^{-\alpha(t-t_0)} - \|Q\|\epsilon. \tag{3.15b}$$

The bound in (3.6) and (3.15) implies either

$$\kappa[W_0 f_0]_i > \tau_i - [Qh(t)]_i \text{ or}$$
$$\kappa[W_0 f_0]_i < -\tau_i - [Qh(t)]_i .$$

This is equivalent to

$$\kappa[W_0 f_0]_i + [Qh(t)]_i > \tau_i \text{ or}$$
$$\kappa[W_0 f_0]_i + [Qh(t)]_i < -\tau_i.$$

Thus $|r_i| = |\kappa[W f_0]_i + [Qh(t)]_i| > \tau_i$ and Proposition 3.2.1 follows. □

The urgency of detecting the fault may vary from application to application; however, the ability to detect the fault as quickly as possible is always a benefit. Suppose that (3.14) holds. Then it is clear that if

$$e^{-\alpha(t-t_0)} < \frac{|W_0 f_0|_i \kappa - \tau_i - \|Q\|\epsilon}{\|Q\|\beta}$$

the fault with scaling parameter κ can be detected by time t. Alternatively, we have

Proposition 3.2.2. *If $|[W_0 f_0]_i|\kappa - \tau_i - \|Q\|\epsilon > 0$, and $t > t_1$, then detection can be done by time t if*

$$t - t_0 > -\frac{1}{\alpha} \log \left(\frac{|W_0 f_0|_i \kappa - \tau_i - \|Q\|\epsilon}{\|Q\|\beta} \right). \tag{3.16}$$

Now we turn our attention to the time varying aspect of the fault. Recall f in (3.10) is $\begin{bmatrix} \bar{f}_p \\ f_s \end{bmatrix}$ which includes the derivatives of f_p. Then,

$$f = \kappa \begin{bmatrix} g(t)d_p \\ g'(t)d_p \\ g''(t)d_p \\ g(t)d_s \end{bmatrix}. \tag{3.17}$$

We have already analyzed the case when $t > t_1$. When $t_0 \leq t \leq t_1$ a fault is detected if

$$\left| \kappa \left[W \begin{bmatrix} g(t)d_p \\ g'(t)d_p \\ g''(t)d_p \\ g(t)d_s \end{bmatrix} \right]_i + [Qh(t)]_i \right| > \tau_i.$$

Then, analogously to (3.14), $|r_i(t)| > \tau_i$ holds if

$$\kappa > \frac{\tau_i + \|Q\|\beta e^{-\alpha(t-t_0)} + \|Q\|\epsilon}{\left| \left[W \begin{bmatrix} g(t)d_p \\ g'(t)d_p \\ g''(t)d_p \\ g(t)d_s \end{bmatrix} \right]_i \right|}.$$

3.2.1 False Alarms

There is a trade-off when choosing the tolerance, τ, between permitting false alarms and detecting small faults. In some applications—especially in safety related processes like aircraft, automobiles, and chemical plants—false alarms are a necessary evil for safe operation of the

system. In other applications, it may be more cost effective to choose a higher tolerance so that false alarms are a rare occurrence or never happen at all.

False alarms occur if the residuals cross the threshold set by the tolerance, τ, and $f = 0$ in the equation

$$Wf + \begin{bmatrix} H \\ -G \end{bmatrix} h(t) = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}.$$

Then, if τ is to be chosen to prevent this, it is necessary that

$$\left| \begin{bmatrix} H \\ -G \end{bmatrix} h(t) \right| < \tau$$

where $<$ refers to component-wise comparison. Using (3.6), it is clear that if τ is chosen so that

$$\tau > \left\| \begin{bmatrix} H \\ -G \end{bmatrix} \right\| \beta e^{-\alpha(t-t_0)},$$

then no false alarms will occur.

3.2.2 Fault Location

Not all faults are treated equally because of the presence of the DAE. Let $P = [(sE + F)^{-1}F]^D[(sE + F)^{-1}F]$ where D denotes the Drazin pseudoinverse [19]. Then Pf_p enters the equations much like an ODE where $(I - P)f_p$ can contain derivatives. Thus if $(I - P)f_p = 0$ we can treat it as an immediately occurring fault where as if $(I - P)f_p \neq 0$ then there may be transient reactions to the fault of increasing magnitude the quicker the fault occurs. Note that $N(G) = R(P)$. This is illustrated with the circuit example in Section 3.6.

3.3 Fault Identification

The task of fault identification consists of determining the type, size, and location of the detected fault. In this section, the idea of fault identification is narrowed to specifying which fault has occurred given a library of possible faults. Consider process and sensor fault directions d_{p_i} and d_{s_i}. Then, the fault can be written $f_i(t) = \kappa\left(g(t)v_{1_i} + g'(t)v_{2_i} + g''(t)v_{3_i}\right)$, where $v_{1_i} = [d_{p_i}^T\ 0\ 0\ d_{s_i}^T]^T$, $v_{2_i} = [0\ d_{p_i}^T\ 0\ 0]^T$, and $v_{3_i} = [0\ 0\ d_{p_i}^T\ 0]^T$. Here, the subscript i denotes the i^{th} fault in a library of faults and f_i is used to denote all of the process and sensor fault components. Using this expansion, it is clear that $f_i(t)$ varies over a subspace given by span$\{v_{1_i}, v_{2_i}, v_{3_i}\}$. Given a library of faults, $\{f_1(t), \ldots, f_r(t)\}$, the next proposition constructs filtering matrices that identify faults by matrix multiplication.

Proposition 3.3.1. *Suppose* $\{f_1(t), \ldots, f_r(t)\}$ *are faults and there is at least one* l *such that* $Wv_{l_j} \notin \text{span}\{Wv_{1_i}, Wv_{2_i}, Wv_{3_i}\}$, *for each* $j \neq i$. *Define*

$$M_i = \begin{bmatrix} (Wv_{1_i})^T \\ (Wv_{2_i})^T \\ (Wv_{3_i})^T \end{bmatrix} \text{ and } \text{svd}(M_i) = U\Sigma V^T.$$

Let R_i^T *be the last columns of* V^T *corresponding to* $N(M_i)$. *Then, the following are true:*

1. $R_i W f_i(t) = 0$, $\forall t$
2. $R_i W f_j(t) \neq 0$, $\forall t$, $\forall j \neq i$ *such that* $f_j(t) \neq 0$.

Proof. $R_i W f_j(t) = \kappa R_i \left(g(t)Wv_{1_j} + g'(t)Wv_{2_j} + g''(t)Wv_{3_j}\right)$. By construction $M_i R_i^T = 0$, implying $R_i M_i^T = [R_i Wv_{1_i} \;\; R_i Wv_{2_i} \;\; R_i Wv_{3_i}] = 0$. Therefore, the first statement holds. The hypothesis of the proposition implies that for every f_j with $j \neq i$ there exists an l such that $R_i Wv_{l_j} \neq 0$. Hence, the second statement holds. □

Note that the hypothesis of the theorem is exactly what is needed to create a filter to identify f_i. If all r filters are to be created to identify all r faults, the proposition can be extended easily by requiring the hypothesis to hold for every i.

An analogous but alternative filter is given by

Proposition 3.3.2. *Suppose* $\{f_1(t), \ldots, f_r(t)\}$ *are faults and* $Wv_{l_i} \notin \text{span}\{Wv_{1_1}, Wv_{2_1}, Wv_{3_1}, \ldots, \widehat{Wv_{1_i}}, \widehat{Wv_{2_i}}, \widehat{Wv_{3_i}}, \ldots, Wv_{3_r}\}$, *for at least one* l, *where the use of* $\widehat{\;}$ *denotes missing elements. Define*

$$M_i^T = \left[Wv_{1_1},\ Wv_{2_1},\ Wv_{3_1},\ \ldots\ \widehat{Wv_{1_i}},\ \widehat{Wv_{2_i}},\ \widehat{Wv_{3_i}},\ \ldots\ Wv_{1_r},\ Wv_{2_r},\ Wv_{3_r}\right],$$

$\text{svd}(M_i) = U\Sigma V^T$, *and let* R_i^T *be the last columns of* V^T *corresponding to* $N(M_i)$. *Then, the following are true:*

1. $R_i W f_i(t) \neq 0$, $\forall t$ *such that* $f_i(t) \neq 0$
2. $R_i W f_j(t) = 0$, $\forall t$, $j \neq i$.

Proof. Consider $R_i W f_j(t) = \kappa R_i \left(g(t)Wv_{1_j} + g'(t)Wv_{2_j} + g''(t)Wv_{3_j}\right)$. By construction $M_i R_i^T = 0$, implying $\begin{bmatrix} R_i Wv_{1_j} & R_i Wv_{2_j} & R_i Wv_{3_j} \end{bmatrix} = 0$, for $j \neq i$. Therefore, the second statement holds. The hypothesis of the proposition implies that for every f_j with $j = i$ there exists an l such that $R_i Wv_{l_i} \neq 0$. Hence, the first statement holds. □

A more ambitious filter may sometimes be constructed. We rewrite (3.7a) and (3.7c)

$$\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} H \\ -G \end{bmatrix} e + \begin{bmatrix} f_s \\ -\tilde{G}\bar{f}_p \end{bmatrix}. \tag{3.18}$$

If $\begin{bmatrix} H \\ -G \end{bmatrix}$ does not have full row rank, then there exists a nontrivial R such that $R\begin{bmatrix} H \\ -G \end{bmatrix} = 0$. This leads us to

Proposition 3.3.3. *Let* $Q = \begin{bmatrix} H \\ -G \end{bmatrix}$. *Suppose* $\{f_1(t), \ldots, f_r(t)\}$ *are faults and*

$$Wv_{l_i} \notin \text{span}\{Wv_{1_1}, Wv_{2_1}, Wv_{3_1}, \ldots, \widehat{Wv_{1_i}}, \widehat{Wv_{2_i}}, \widehat{Wv_{3_i}}, \ldots, Wv_{3_r}, Q^T\},$$

for at least one l, *where the use of* $\widehat{\ }$ *denotes missing elements. Define*

$$M_i^T = \left[Wv_{1_1},\ Wv_{2_1},\ Wv_{3_1},\ \ldots\ \widehat{Wv_{1_i}},\ \widehat{Wv_{2_i}},\ \widehat{Wv_{3_i}}, \ldots\ Wv_{1_r},\ Wv_{2_r},\ Wv_{3_r},\ Q\right],$$

$\text{svd}(M_i) = U\Sigma V^T$, *and let* R_i^T *be the last columns of* V^T *corresponding to* $N(M_i)$. *Then, the following are true:*

1. $R_i W f_i(t) \neq 0$, $\forall t$ *such that* $f_i(t) \neq 0$
2. $R_i W f_j(t) = 0$, $\forall t$, $j \neq i$
3. $R_i Q = 0$.

Proof. Consider $R_i W f_j(t) = \kappa R_i \left(g(t)Wv_{1_j} + g'(t)Wv_{2_j}. + g''(t)Wv_{3_j}\right)$. By construction $M_i R_i^T = 0$. Hence, $R_i M_i^T = 0$, implying $\begin{bmatrix} R_i Wv_{1_j} & R_i Wv_{2_j} & R_i Wv_{3_j} & R_i Q \end{bmatrix} = 0$, for $j \neq i$. The second and third statements now follow easily. The hypothesis of the proposition implies there exists an l such that $R_i Wv_{l_i} \neq 0$, so statement one holds. □

Implementing Prop. 3.3.3 will result in

$$R_i r_j = \begin{cases} 0 & \text{if } i \neq j \\ R_i \begin{bmatrix} f_s \\ -\tilde{G}\bar{f}_p \end{bmatrix} & \text{if } i = j, \end{cases} \tag{3.19}$$

suggesting that faults can be identified immediately upon detection. As stated after Proposition 3.3.1, the hypotheses of Props. 3.3.1-3.3.3 are what are needed to create filters for fault f_i. The hypotheses can be extended so that filters can be created for all r faults by requiring that

the hypotheses hold for all i. The filters R_i can be used in real time for identification. One possibility of their use is given in the numerical examples in Section 3.6.

W_0 and its Affect on FDI

The hypotheses on the identification theorems imply that the null space $N(W)$ and range $R(W)$ of W or W_0 are very important in determining which faults can be identified. $N(W)$ denotes a matrix whose columns are a basis for the null space. In terms of detection, (3.10) indicates that if a fault $f \in N(W)$, then the fault cannot be detected. Therefore, an investigation into the null space of W or W_0 is relevant and helpful. This section studies W_0 exclusively so the results in this section are applicable to fully developed ($t > t_1$) faults only.

Proposition 3.3.4 fully characterizes the null space of W_0. The proposition indicates that the dimension of the null space is n. Recall, the fault in its constant regime is of dimension $n+m$, where n is the dimension of the state and m is the number of rows of H. Hence, we may identify up to m distinct faults that meet the hypotheses of Props. 3.3.1-3.3.3. That is to say, the maximum number of faults that can be identified cannot exceed the number of independent measurements.

Proposition 3.3.4. $N(W_0) = R\left(\begin{bmatrix} I_n \\ H\hat{A}^{-1}\hat{G}_0 \end{bmatrix}\right)$.

Proof. We will need the following:

$$LH(\hat{A} - LH)^{-1} + I = \hat{A}(\hat{A} - LH)^{-1}. \tag{3.20}$$

The matrix L has full column rank so multiplying the top block row of W_0 by L leaves the null space unaffected. Then, we use (3.20) to get

$$\tilde{W}_0 = \begin{bmatrix} -LH\left(\hat{A} - LH\right)^{-1}\hat{G}_0 & \hat{A}\left(\hat{A} - LH\right)^{-1}L \\ G\left(\hat{A} - LH\right)^{-1}\hat{G}_0 - \tilde{G}_0 & -G\left(\hat{A} - LH\right)^{-1}L \end{bmatrix}.$$

Since $\hat{A}^{-1}$ exists, we can carry out the block row operation, $G\hat{A}^{-1}R_1 + R_2 \to R_2$ yielding

$$\bar{W}_0 = \begin{bmatrix} -LH\left(\hat{A} - LH\right)^{-1}\hat{G}_0 & \hat{A}\left(\hat{A} - LH\right)^{-1}L \\ G\hat{A}^{-1}\hat{G}_0 - \tilde{G}_0 & 0 \end{bmatrix}.$$

Lemma 2.4.3 says that the (2,1) entry of $\bar{W}_0$ is 0. Consider $\bar{W}_0\phi = 0$ and partition $\phi = \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix}$ conformal with the block matrix W_0. The top block row implies $-LH(\hat{A}-LH)^{-1}\hat{G}_0\phi_1 + \hat{A}(\hat{A}-$

$LH)^{-1}L\phi_2 = 0$. Solving for ϕ_2 we see that

$$\phi_2 = L^\dagger(\hat{A} - LH)\hat{A}^{-1}LH(\hat{A} - LH)^{-1}\hat{G}_0\phi_1 \tag{3.21a}$$
$$= L^\dagger(\hat{A} - LH)\hat{A}^{-1}(-I + \hat{A}(\hat{A} - LH)^{-1})\hat{G}_0\phi_1 \tag{3.21b}$$
$$= L^\dagger LH\hat{A}^{-1}\hat{G}_0\phi_1 \tag{3.21c}$$
$$\phi_2 = H\hat{A}^{-1}\hat{G}_0\phi_1 \tag{3.21d}$$

with the restriction $(I - LL^\dagger)LH(\hat{A} - LH)\hat{G}_0\phi_1 = 0$. However, this simplifies to $0\phi_1 = 0$, so the only restriction on the null space is (3.21d). □

Therefore, some of the rows of W_0 are redundant. However, there are still advantages to using both residuals, even in this constant fault case. For detection, it is clear from (3.14) that a fault may be easier to see in certain components if we consider all rows of W_0. The same is true for fault identification.

Suppose we are only concerned with process faults. This means that $\phi_2 = 0$ and we can write a further truncated version of W,

$$\hat{W}_0 = \begin{bmatrix} -H(\hat{A} - LH)^{-1}\hat{G}_0 \\ G(\hat{A} - LH)^{-1}\hat{G}_0 - \tilde{G}_0 \end{bmatrix}.$$

Then (3.21d) implies that $N(\hat{W}_0) = N(H\hat{A}^{-1}\hat{G}_0)$. The dimension of this space is important for the task of fault identification. A smaller null space will give us greater freedom to identify faults. With this in mind, we have

Proposition 3.3.5. $\dim(N(H\hat{A}^{-1}\hat{G}_0)) = n - \text{rank}(H)$.

Proof. $\hat{A}^{-1}$ and $\hat{G}_0$ (see Lemma 2.4.4) have full column rank so $\text{rank}(H\hat{A}^{-1}\hat{G}_0) = \text{rank}(H)$. □

Linear Combinations of Faults

As in previous sections, we are interested in a library of faults $\{f_1(t), \ldots, f_r(t)\}$. However, in this section, $f_i(t) = g(t)v_{1_i} + g'(t)v_{2_i} + g''(t)v_{3_i}$. f_i is not scaled by κ because here we form a fault that is a linear combination of the f_i. The scalars involved in the linear combination are our κ_i values. Then we denote the fault $F = Y\kappa$, where $Y = [f_1 \ldots f_j]$ and $\kappa = [\kappa_1 \ldots \kappa_j]^T$. Recall, a fault is detected if $|[WY\kappa]_i + [Qh(t)]_i| > \tau_i$. Using the bound on $h(t)$, a fault is detected if $|[WY\kappa]_i| > \tau_i + \|Q\|(\beta e^{-\alpha(t-t_0)} + \epsilon)$.

For identification purposes we are interested in determining which faults make up the linear combination, Y, and their respective sizes κ. Creating the filters R_i according to propositions 3.3.2 or 3.3.3 are convenient ways to do this. Up to noise $Qh(t)$, $R_i r$ will be nonzero if and only if f_i is involved in the linear combination.

We can also estimate κ_i for each f_i. Due to the construction of R_i, $R_iWF = \kappa_i R_i W f_i$. This implies $|\kappa_i|\|R_iWf_i\| = \|R_ir - R_iQh(t)\|$ and noting that

$$\begin{aligned}\|R_ir - R_iQh(t)\| &\leq \|R_ir\| + \|R_iQh(t)\| \\ &\leq \|R_ir\| + \|R_iQ\|\theta(t) \\ \|R_ir - R_iQh(t)\| &\geq \|R_ir\| - \|R_iQh(t)\|r \\ &\geq \|R_ir\| - \|R_iQ\|\theta(t)\end{aligned}$$

we see that

$$\frac{\|R_ir\| - \|R_iQ\|\theta(t)}{\|R_iWf_i\|} \leq |\kappa_i| \leq \frac{\|R_ir\| + \|R_iQ\|\theta(t)}{\|R_iWf_i\|} \tag{3.23}$$

assuming that the noise $h(t)$ is small enough so that $\|R_ir\| - \|R_iQ\|(\beta e^{-\alpha(t-t_0)} + \epsilon) > 0$. The bounds on $|\kappa_i|$ are converging at the same rate of the observer to the true value of $\kappa_i \pm \frac{\|R_iQ\|}{\|R_iWf_i\|}\epsilon$. Note, this method of estimating κ is also applicable in the case where there is only one fault that is scaled by κ, not just linear combinations of two or more faults.

3.4 Disturbance Attenuation

Normal plant uncertainties are a major issue that must be addressed during FDI. Disturbances can cause false alarms, failed fault detection, and hinder fault identification. One popular approach for ODE and DAE FDI is sliding mode control [47]. Other approaches include the use of Laplace transformations to compute transfer function matrices [69] and unknown input observers [26]. Generally, these approaches require the disturbance to affect the differential equation in a manner that is linearly independent to the fault in some sense. To clarify, consider the following LTI DAE with disturbance $d(t)$ and faults f_p and f_s

$$Ex' = -Fx + Bu + D_f f_p + D_d d \tag{3.24a}$$

$$y = Hx + Du + f_s. \tag{3.24b}$$

One necessary assumption to decouple the disturbance using the above approaches is the rank criteria, $\text{rank}\,[D_f\ D_d] > \max\{\text{rank}(D_f),\ \text{rank}(D_d)\}$. This condition may be too restrictive in some cases. Therefore, in this section we aim to develop methods even if this condition is not met.

Instead, we assume that the disturbances occur in a frequency bandwidth that can be isolated from the frequencies of the faults. The faults under consideration in this paper have low frequency; therefore, we will assume that the disturbances occur at a higher frequency than the faults. High frequency disturbances occur routinely in many industrial and mechanical systems.

There exist numerous causes: imperfect functional performance of sensors used to measure actual load parameters; influence of pulsed power equipment; vibrations caused by imbalance, misalignment, resonance, defective bearings, reciprocating forces, etc; and aerodynamic noise caused by turbulence, acoustic modes, pressure pulsations, etc.

High frequency disturbances affecting DAE problems are more destructive than in the ODE case since derivatives of the disturbance will be involved in the solution of the DAE. Therefore, even very low amplitude, high frequency noise can be detrimental to FDI.

We consider (3.1a) with an unknown high frequency disturbance $d(t)$. That is $f_p + d$ in place of f_p. In terms of (3.24a) we have $D_f = D_d = I$ and the rank criteria does not hold. Recall the discrete Fourier transform (DFT) converts a finite list of equally spaced samples of a function into a list of coefficients ordered by their frequencies [9]. Let $\hat{y}_k = (\mathcal{F}_N\{y\})_k$ denote the DFT of y. Then $\hat{y}_k$ is a complex sequence with N terms where N is the number of samples of y. If y is our residual signal, we filter it by setting $\hat{y}_k = 0$ for those k that lie in the frequency bandwidth of our disturbance. $\hat{y}_k$ corresponds to the frequency of k/N cycles per sample. If the sampling frequency (number of samples taken per time unit) is F_s, then $\hat{y}_k$ corresponds to the frequency kF_s/N cycles per time unit. Hence, if t is seconds and the disturbances are known to occur between $30 - 40$Hz, we set $\hat{y}_k = 0$ for $30N/F_s \leq k \leq 40N/F_s$. Finally, to recover the filtered signal we apply the inverse DFT, $\mathcal{F}_N^{-1}$, to the adjusted coefficients $\hat{y}_k$.

Let $w = \mathrm{e}^{2\pi i/N}$. The computation of the DFT is equivalent to the matrix computation $\mathcal{F}_N\{y\} = \overline{F}_N y$ where $y = (y_0, \ldots, y_{N-1})^T$ and the j,k entry of $\overline{F}_N$ is $w^{(j-1)(k-1)}$ for j,k from 1 to N. Note that F_N is a symmetric matrix. Since matrix multiplication is a linear process and $\mathcal{F}_N$ maps N-periodic sequences to N-periodic sequences, $\mathcal{F}_N$ is a linear operator.

The procedure outlined in this section does not affect our previous results on detection and identification. Detection is a residual threshold comparison. If the fault does not contain the frequencies that are filtered, detection times will not significantly change.

For identification it suffices to show that the order in which we apply the DFT and the identification filters R_i does not matter. Suppose we have N samples of our residual vector $r(t)$ at equally spaced temporal nodes t_j. Let R be the matrix defined by $(R)_{jk} = r_k(t_j)$. Consider applying $\mathcal{F}_N$ to R for each column k. The filtering process, setting DFT coefficients equal to zero that correspond to specified frequencies, is equivalent to setting rows of F_N equal to zero. Let F_{N_0} be this modified matrix. Therefore, using the DFT to filter a specified bandwidth and then applying a fault identification filter R_i is equivalent to the matrix multiplication $R_i(\frac{1}{N}F_N\overline{F}_{N_0}R)^T = \frac{1}{N}R_iR^T\overline{F}_{N_0}F_N$. On the other hand, if we first apply R_i and then the DFT we get $\frac{1}{N}F_n\overline{F}_{N_0}\left(R_iR\right)^T = \frac{1}{N}F_N\overline{F}_{N_0}R^TR_i^T$, the transpose of the former.

3.5 Algorithms

In this section we extract the actual algorithms needed to carry out the procedures described in the previous section. The algorithms can be used concurrently, in real time, to perform robust FDI.

1. Use Prop. 3.2.1 to choose a problem specific threshold vector, τ, based on the faults under consideration.

2. Given (3.1), compute its completion, (3.2). Verify the chosen Λ creates a detectable pair $\{\hat{A}, H\}$.

3. Find L such that $\hat{A} - LH$ has desired stable eigenvalues.

4. Integrate (3.3a) using a differential equation solver, e.g. ode45 in Matlab.

5. Calculate $r(t) = [r_1^T(t)\ r_2^T(t)]^T$ according to (3.5) at each integration step.

6. At each step in the integration, if the residual $r > \tau$ in any component, then filter the residual using the following steps:
   (a) Set an interval on which the residual is to be filtered. That is, let δ_1, $\delta_2 > 0$, t be the current time, and $t_0 < t$ such that every point in the window $[t_0 - \delta_1\ t_0 + \delta_2]$ is less than t.
   (b) Interpolate the residual on this window at equally spaced nodes.
   (c) Find the DFT of this residual.
   (d) Set the coefficients that correspond to unwanted frequencies to zero.
   (e) Apply the inverse DFT to the filtered residual.

7. If the filtered residual still violates the threshold, then declare a fault has occurred.

Algorithm 3.1: Automated robust fault detection.

In the above algorithm, we apply the DFT on an interval that lies before t. The reason for allowing a buffer before t is because of numerical instability of the DFT at the boundary. Also, we do not begin the intervals at $t = 0$ for computational efficiency.

In the following algorithm, Proposition 3.3.3 is used below as an illustration; however, any of the identification propositions in the paper may be used.

1. Choose faults $\{f_1 \dots f_r\}$ to be identified and verify that the hypotheses of Prop. 3.3.3 are met.

2. Calculate a reasonable noise bound as in (3.6).

3. Calculate R_i offline, according to Prop. 3.3.3 for each fault, f_i, the user wishes to identify.

4. Perform steps 2-6 in Algorithm 3.1. The threshold comparison in step 6 is optional.

5. At each integration step after the fault has occurred, or, if not using a threshold, then at every integration step, compute $\|R_i r\|$. Fault f_i has occurred if the norm is greater than 0 (or greater than numerical and measurement error).

6. Calculate upper and lower bounds for κ_i using (3.23).

Algorithm 3.2: Fault identification and κ_i estimation.

3.6 Circuit Example

We shall consider the circuit shown in Figure 3.1 which is a variation of one from [63] and is modeled by

$$C_1 e_1' - i_{r_1} + i_{r_2} - i_v = f_{p_1} \tag{3.25a}$$
$$C_2 e_2' + i_l + i_{r_1} = f_{p_2} \tag{3.25b}$$
$$L i_l' - e_2 = f_{p_3} \tag{3.25c}$$
$$-e_1 + e_2 - R_1 i_{r_1} = f_{p_4} \tag{3.25d}$$
$$e_1 - R_2 i_{r_2} = f_{p_5} \tag{3.25e}$$
$$e_1 = -V + f_{p_6}, \tag{3.25f}$$

a linear, index $k = 2$ DAE. The state is $x = [e_1\, e_2\, i_l\, i_{r_1}\, i_{r_2}\, i_v]^T$ and the control is $u = V$. f_{p_j} is the j^{th} component of the process fault. For purposes of illustration we take $C_1 = 3, C_2 = 2, L = 2, R_1 = 4, R_2 = 5$. We assume $V = 4$, that is, there is constant voltage source. System (3.25) is LTI and of the form

$$Ex' + Fx = Bu + f_p, \tag{3.26}$$

where

$$E = \text{diag}(3, 2, 2, 0, 0, 0), \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}^T,$$

$$F = \begin{bmatrix} 0 & 0 & 0 & -1 & 1 & -1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -4 & 0 & 0 \\ 1 & 0 & 0 & 0 & -2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

In addition, we have the output equation

$$y = Hx + f_s, \;\; H = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Recall, f_s is a sensor fault. We let f_{s_i} denote components of the sensor fault.

3.6.1 Forming the Completion

Applying the differential operator $\mathcal{D} = d/dt + \Lambda$, where Λ is a diagonal matrix, $\Lambda = \text{diag}\{1.5, 1.2, 1.6, 1.8, 1.9, 2\}$, to (3.26) k times yields the derivative array equations

$$\mathcal{E}z + \mathcal{F}x = \mathcal{B}\bar{u} + D_f \bar{f}_p, \tag{3.27}$$

where

$$\bar{u} = \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}, D_f = \begin{bmatrix} I & 0 & 0 \\ \Lambda & I & 0 \\ \Lambda^2 & 2\Lambda & I \end{bmatrix}, \bar{f}_p = \begin{bmatrix} f_p \\ f_p' \\ f_p'' \end{bmatrix}.$$

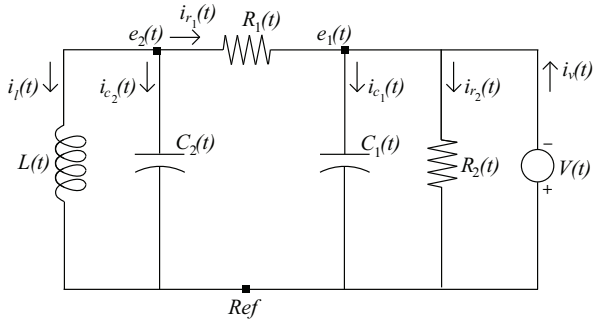


Figure 3.1: Circuit Example from [63].

As with (2.12) we have

$$z = \begin{bmatrix} x' \\ * \end{bmatrix} = -\mathcal{E}^\dagger \mathcal{F} x + \mathcal{E}^\dagger \mathcal{B} \bar{u} + \mathcal{E}^\dagger D_f \bar{f}_p.$$

Let $\hat{A}$, $\hat{B}$, and $\hat{G}$ be the first n rows of $-\mathcal{E}^\dagger\mathcal{F}$, $\mathcal{E}^\dagger\mathcal{B}$, and $\mathcal{E}^\dagger D_f$, respectively. Then $x' = \hat{A}x + \hat{B}\bar{u} + \hat{G}\bar{f}_p$ is the completion. Let Θ be a maximum rank, left annihilator of $\mathcal{E}$. Then, left multiplication of Θ on (3.27) yields the constraints $0 = -\Theta\mathcal{F}x + \Theta\mathcal{B}\bar{u} + \Theta D_f \bar{f}_p$. Define G, $\tilde{B}$, $\tilde{G}$ as $-\Theta\mathcal{F}$, $\Theta\mathcal{B}$, and ΘD_f, respectively. Therefore, the stabilized completion with constraints is

$$\begin{aligned} x' &= \hat{A}x + \hat{B}\bar{u} + \hat{G}\bar{f}_p \\ 0 &= Gx + \tilde{B}\bar{u} + \tilde{G}\bar{f}_p. \end{aligned}$$

The annihilator can be taken as $\Theta = [0\ I]U^T$, where U is found via the singular value decomposition, $\mathcal{E} = U \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} V^T$. The zero block in Θ is $((k+1)n - p \times p)$ and the identity has $(k+1)n - p$ rows and columns where k is the index of the DAE, n is the dimension of the state x, and $p = \text{rank}(\mathcal{E})$.

3.6.2 Fault Detection

The standard observer (3.3) is used. A feedback L that stabilizes $\hat{A} - LH$ is

$$L = \begin{bmatrix} 0.2854 & 0.0000 & 2.7963 & 0.0832 \\ 1.4102 & -0.5000 & -0.5164 & -0.0037 \\ 0.5000 & 2.0000 & -0.0000 & 0.0000 \\ 0.3191 & -0.1250 & -0.8612 & -0.0272 \\ -0.1320 & 0.0000 & 14.5869 & -0.5919 \\ -3.3224 & 0.1250 & -0.7633 & -1.3240 \end{bmatrix}$$

which places eigenvalues at $\{-1, -1.8, -1.6, -1.4, -1.2, -2\}$. Eigenvalues were chosen to be distinct and asymptotically stable. The initial conditions used in the simulations are $x(0)$, $\hat{x}(0)$ given by $[-V,\ 2,\ -2C_2 - \frac{V+2}{R_1},\ \frac{V+2}{R_1},\ \frac{-V}{R_2},\ -\frac{V+2}{R_1} - \frac{V}{R_2}]$, and $[-V,\ 1,\ -C_2 - \frac{V+1}{R_1},\ \frac{V+1}{R_1},\ \frac{-V}{R_2},\ -\frac{V+1}{R_1} - \frac{V}{R_2}]$, respectively.

When a sensor fault is present, it is often possible to detect it almost immediately since it is directly connected to the output. Thus while they are important, they are easier to detect. We will focus on process faults.

In practice, thresholds and other parameter values are taken with specific design issues in mind. Here our purpose is just to discuss our results and illustrate them. Accordingly, no physical significance should be attached to the specific values chosen.

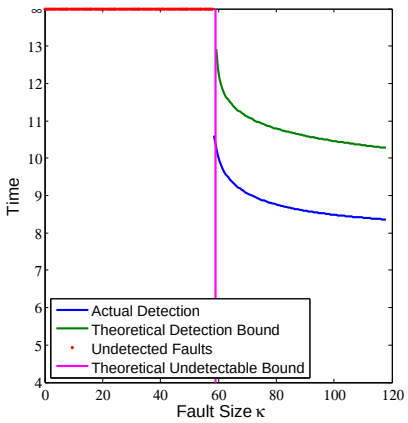


Figure 3.2: Comparison between actual and theoretical detection times, undetectable faults, and the theoretical undetectable fault bound.

As a first example of a fault with zeros in the sensor fault positions we take

$$f_0 = \kappa g(t) \left[\begin{array}{cccccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{array} \right]^T,$$

where the entries of f_0 include process and sensor fault components as in Section 3.3 for computational convenience. Let $t_0 = t_1 = 8$ so that the fault is piecewise constant. Figures 3.2 and 3.3 display the results of the simulation using a large range of κ values. The absolute detection time error in Figure 3.3 is $t_h - t_a$, where t_h is the theoretical detection time bound and t_a is the actual detection time from the simulation. The theoretical undetectable bound in this example accurately predicts detectable and undetectable values of κ and the theoretical bound on the time of detection is valid.

Figure 3.4 shows the residual for a given κ resulting in detection at the time corresponding to the dashed line.

In some scenarios, the fault may intensify in a smooth manner until it reaches its maximum/minimum, where it remains. The time varying nature of ramp faults require nonzero derivatives of f when DAEs are used. To understand the implications of this in terms of our example, we rewrite (3.25) into differential (3.28a)–(3.28b) and algebraic equations (3.28c)–(3.28f):

$$C_2 e_2' + i_l + i_{r_1} = f_{p_2} \tag{3.28a}$$

$$L i_l' - e_2 = f_{p_1} \tag{3.28b}$$

$$-C_1 V' - i_{r_1} + i_{r_2} - i_v = f_{p_1} - C_1 f_{p_6}' \tag{3.28c}$$

$$-e_1 + e_2 - R_1 i_{r_1} = f_{p_4} \tag{3.28d}$$

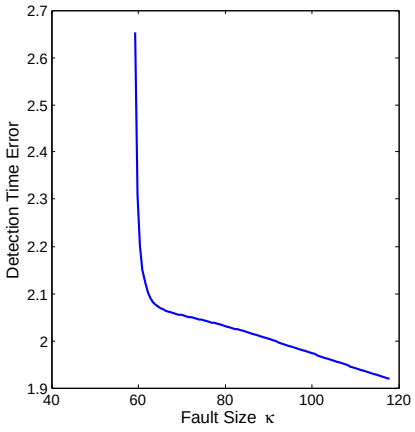


Figure 3.3: Absolute detection time error.

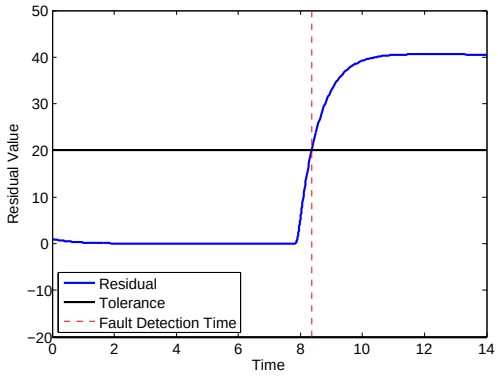


Figure 3.4: First component of the residual r_1 for $\kappa = 117$.

$$e_1 - R_2 i_{r_2} = f_{p_5} \tag{3.28e}$$

$$e_1 = -V + f_{p_6}. \tag{3.28f}$$

Equation (3.28c) is a result of differentiating (3.28f) and substituting the result into (3.25a). Although the completion includes $\bar{f}_p$, a vector that incorporates the derivatives of f_p, (3.28) shows that only f'_{p_6} affects the solution. Therefore ramp faults that contain a nonzero component for f'_{p_6} cause the system to respond differently in terms of residual behavior than those with f'_{p_6} components equal to zero. The difference in behavior is more pronounced as $|f'_{p_6}|$ increases. Hence, if the ramp up time is very short, some residuals observe a spike corresponding to the time when the fault is intensifying.

Three ramp faults are considered:

$$f_1 = \kappa g(t) \left[\begin{array}{cccccc|cccc} -0.56 & 1.52 & -0.69 & 0.61 & 0.37 & 0.63 & 1.06 & 0.35 & 0.17 & -0.75 \end{array} \right]^T$$

$$f_2 = \kappa g(t) \left[\begin{array}{cccccc|cccc} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \right]^T$$

$$f_3 = \kappa g(t) \left[\begin{array}{cccccc|cccc} 0.5774 & 0 & 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 & 0 \end{array} \right]^T.$$

The fault begins at $t_0 = 8$ and reaches its maximum, in the norm sense, at $t_1 = t_0 + t_r$, where it remains. t_r is the ramp up time. Figures involving f_1 were generated with $\kappa = 57$, f_2 with $\kappa = 72,801$, and f_3 with $\kappa = 5,026$. Figures 3.5 – 3.10 display results from the simulations.

Figure 3.5 is a good example of a spike that can occur due to derivative information from a fault. Compare this to Figure 3.6 which exhibits no spike. The spike occurs because f_2 contains a nonzero value for f_{p_6}, the only fault component whose derivative information affects the solution.

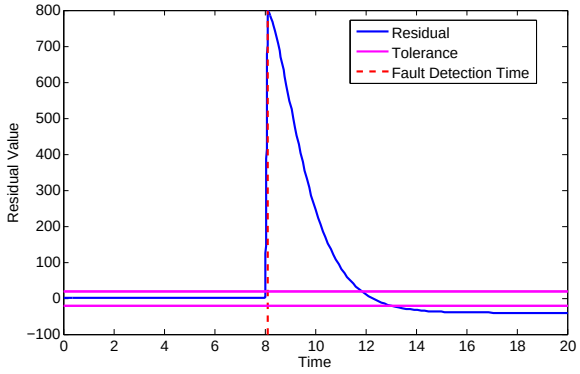


Figure 3.5: First component of r_1 for fault f_2.

The second component of r_1, r_{1_2}, is unaffected by the fault in both cases. This agrees with the theory because $r_{1_2} = i_l - \hat{i}_l + f_{s_2}$ with $f_{s_2} = 0$. Therefore, only faults that affect i_l result in a nonzero r_{1_2}. Equation (3.28) implies that only f_s and f_{p_3} directly affect i_l. Simulations for f_3 and f_4 show the indicated residual quickly going to zero (see Fig. 3.7).

If the ramp time is shortened the residuals do not necessarily converge to their piecewise counterparts in a uniform sense. However residuals generated by process faults are very close to piecewise residuals as long as f'_{p_6} does not have much influence in that component. As shown in Figures 3.8 – 3.10, even if the fault has a nonzero entry in f_{p_6}, the presence of a transient effect is dependent on the sensitivity of that particular residual to f_{p_6}. Over the long term, both piecewise and ramp faults generate equal residuals.

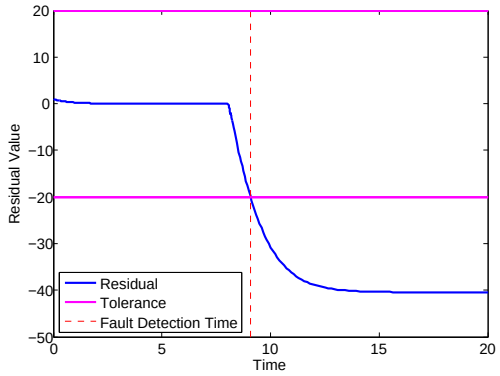


Figure 3.6: First component of r_1 for fault f_3.

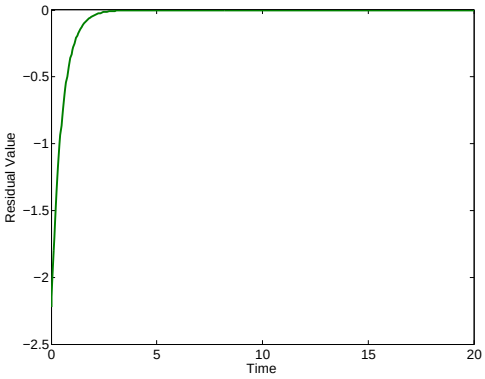


Figure 3.7: No response in second component of r_1 for both f_3 and f_4.

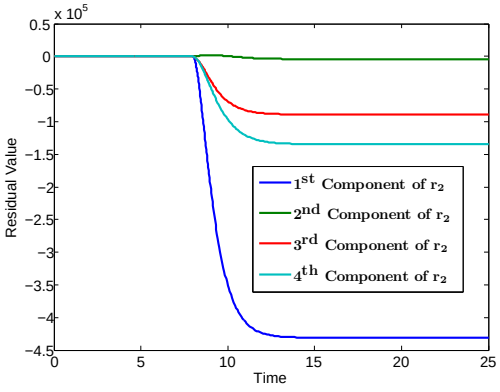


Figure 3.8: All components of r_2 for piecewise constant fault f_2.

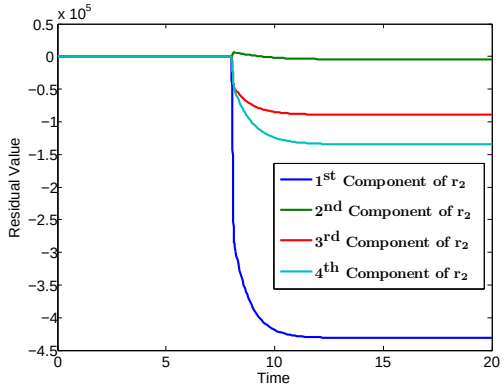


Figure 3.9: All components of r_2 for ramp fault f_2 with ramp time 1/8.

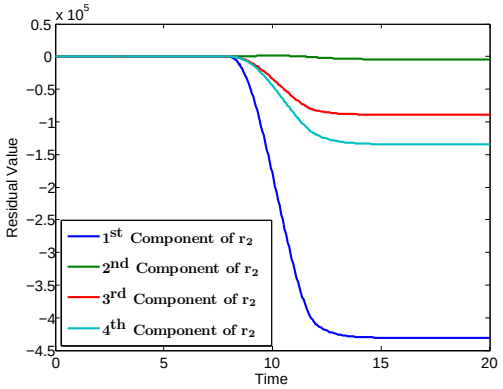


Figure 3.10: All components of r_2 for ramp fault f_2 with ramp time 4.

The addition of derivative information by implementing a ramp fault has little affect on the detection times for f_1 and f_3. An example plot analogous to Figure 3.2 is given for f_3 in Figure 3.11. However, for f_2 the fault is always immediately detected due to the large transient response. Figure 3.12 illustrates this effect for f_2.

Transients can also delay the detection time of a fault. This scenario is encountered when the transient portion of the residual does not break the threshold, but subsequently the residual crosses the threshold in a direction opposite to the transient. For example, consider Figure 3.13 which graphs the first component of r_1 where f_2 is the fault with $\kappa = 100,000$, and $t_r = 84$. By increasing the ramp up time, a situation is created such that the transient effect is not great enough to detect a fault. Moreover, since the long term behavior of r_{1_1} is negative and the transient reaction is positive, it takes longer for the residual to cross the threshold than it would have if there was no transient effect.

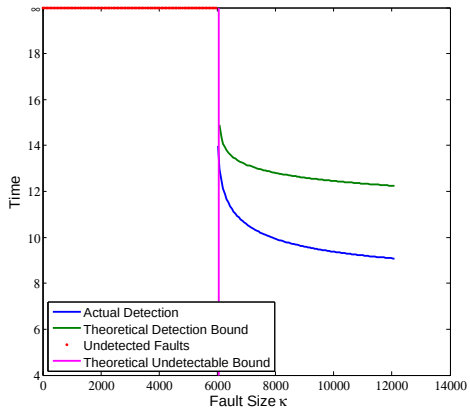


Figure 3.11: Comparison between actual and theoretical detection time, undetectable faults, and the theoretical undetectable fault bound for f_3.

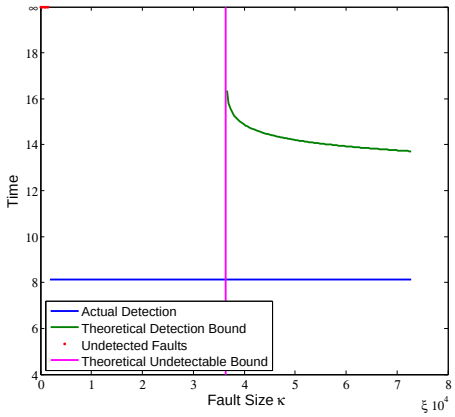


Figure 3.12: Comparison between actual and theoretical detection time, undetectable faults, and the theoretical undetectable fault bound for f_2.

3.6.3 Fault Identification

We consider a fault library $\{f_1, f_2, f_3\}$ where the $f_i^T = \kappa_i g(t) f_{0i}$ are given by

$$f_{01}^T = \left[\begin{array}{cccccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right], \kappa_1 = 5 \tag{3.29a}$$

$$f_{02}^T = \left[\begin{array}{cccccc|cccc} 2 & 1 & 0 & -3 & 0 & 0 & 1 & 2 & 3 & 4 \end{array}\right], \kappa_2 = 1 \tag{3.29b}$$

$$f_{03}^T = \left[\begin{array}{cccccc|cccc} 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right], \kappa_3 = 30. \tag{3.29c}$$

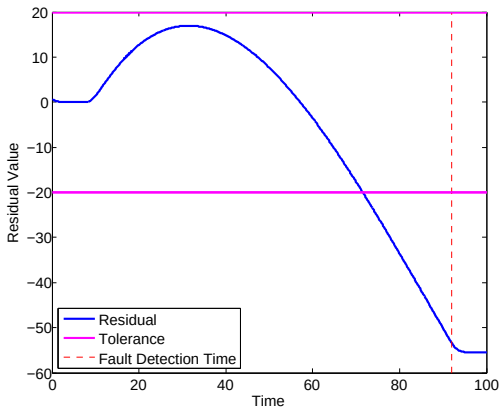


Figure 3.13: First component of r_1 with $\kappa = 10^5$ and γ_2.

Defining $M_i = \begin{bmatrix}(Wv_{1_i}) & (Wv_{2_i}) & (Wv_{3_i})\end{bmatrix}^T$ and letting $R_i = \text{null}(M_i)^T$, we see that the assumptions of Prop. 3.3.2 are met.

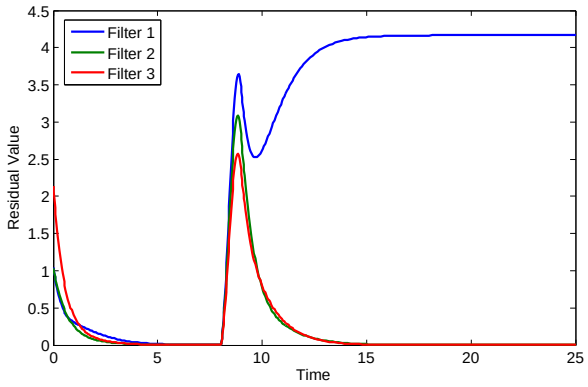


Figure 3.14: Behavior of filters constructed using Prop. 3.3.2 for f_1.

The filters are applied to the residuals $r_i = \begin{bmatrix} r_{1_i} \\ r_{2_i} \end{bmatrix}$, for each fault f_i. According to (3.10) and Prop. 3.3.2,

$$R_i r_j = \begin{cases} R_i \begin{bmatrix} H \\ -G \end{bmatrix} h(t) & \text{if } i \neq j \\ R_i W f_j + R_i \begin{bmatrix} H \\ -G \end{bmatrix} h(t) & \text{if } i = j. \end{cases}$$

Figures 3.14–3.16 show the filters properly identifying each fault. For example, notice that

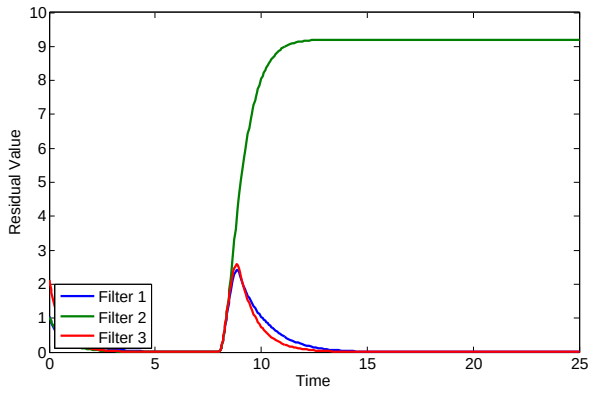


Figure 3.15: Behavior of filters constructed using Prop. 3.3.2 for f_2.

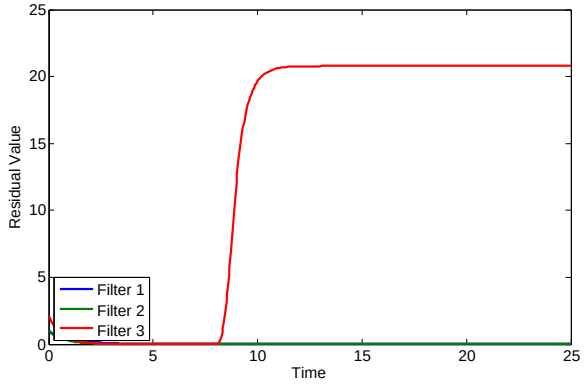


Figure 3.16: Behavior of filters constructed using Prop. 3.3.2 for f_3.

in Figure 3.14, filter 1, which depicts $\|R_1 r_1\|$, remains nonzero for all t while the other two filters go to zero as t increases. Notice the filters that go to zero observe jumps before behaving in their asymptotic manner. This is due to disturbances including the contamination from the exponentially decaying part of the solution to (3.4a).

Proposition 3.3.3 gives modified filters that not only filter fault signals, but also the noise, $h(t)$, as well. The hypothesis of Prop. 3.3.3 is not met for $i = 3$, so only R_1 and R_2 can be created. This illustrates a difficulty in using the procedure in Prop. 3.3.3. Including Q in M_i further restricts the number of faults that can be identified. This is the price of forcing R_i to filter the noise as well as the fault signal. Figure 3.17 shows the filter properly isolating fault f_1. The graph for f_2 is omitted for space reasons but is similar with about three times the magnitude. Notice that in contrast to the filters in the previous example, the filters here do not

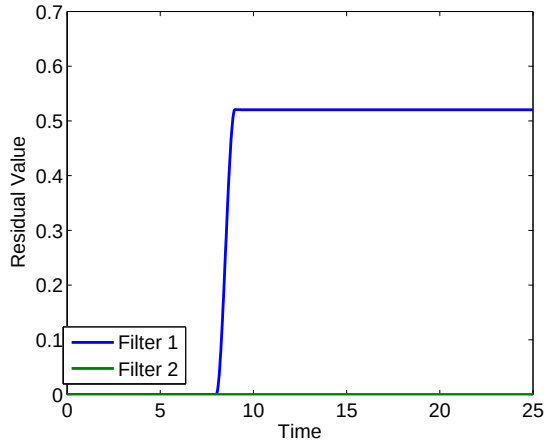


Figure 3.17: Behavior of filters constructed using Prop. 3.3.3 for f_1.

observe jumps before behaving in their asymptotic manner because the noise is filtered.

3.6.4 Linear Combination of Faults

We consider the circuit example again to simulate the effectiveness of the procedures from the section on linear combinations of faults. We use the faults in (3.29). We will consider one ramp fault, $F = \kappa_1 f_1 + \kappa_2 f_2$, a linear combination of two faults from the fault library. Let $\kappa_1 = 5$, $\kappa_2 = 1$, $t_0 = 8$ and $t_1 = 8.125$. Figure 3.18 shows that filters 1 and 2 are nonzero while filter 3 goes to zero as the noise $h(t)$ goes to zero. This correctly implies that f_1 and f_2 are involved in the linear combination while f_3 is not. Figure 3.19 displays accurate bounds on the estimate for κ_1 and κ_2.

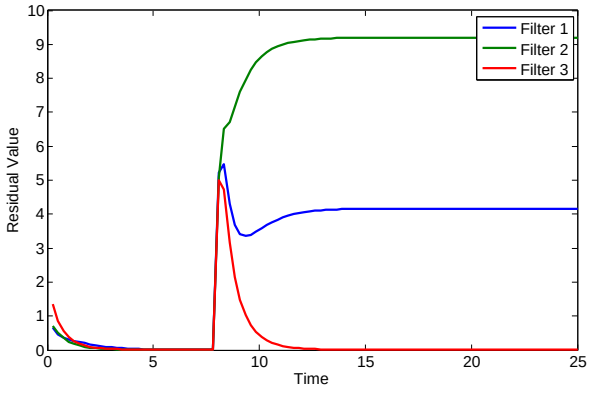


Figure 3.18: Behavior of filters applied to a linear combination $f(t) = \kappa_1 f_1(t) + \kappa_2 f_2(t)$.

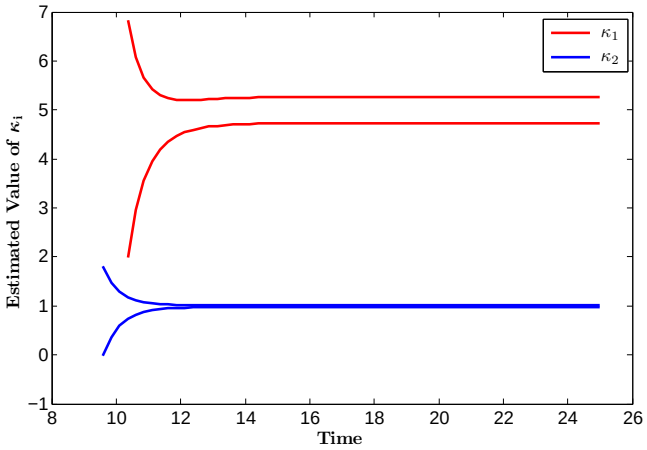


Figure 3.19: Upper and lower bounds for κ_i.

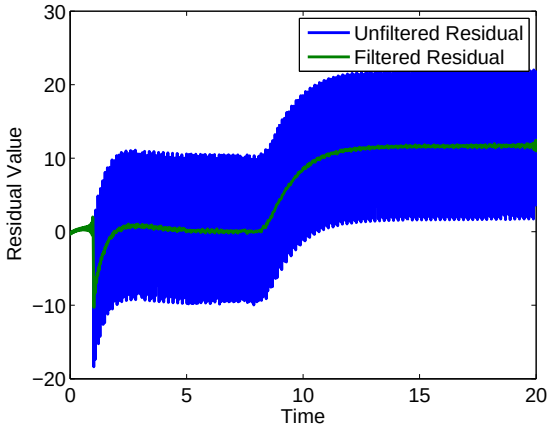


Figure 3.20: Residual before (blue) after (green) disturbance filtering.

3.6.5 Disturbance Filter

We simulate three disturbances, $d_1(t) = .01\sin(20(t-1)2\pi)$, $d_2(t) = .01\cos(25(t-1)2\pi)$, and $d_3(t) = .01\sin(30(t-1)2\pi)$ and filter all frequencies higher than 19 cycles per time unit (TU) using a sampling frequency of 100 samples/TU. The small disturbances have derivatives that are very large affecting the residuals, as shown in Fig. 3.20. We also include the fault $f_2(t)$ from

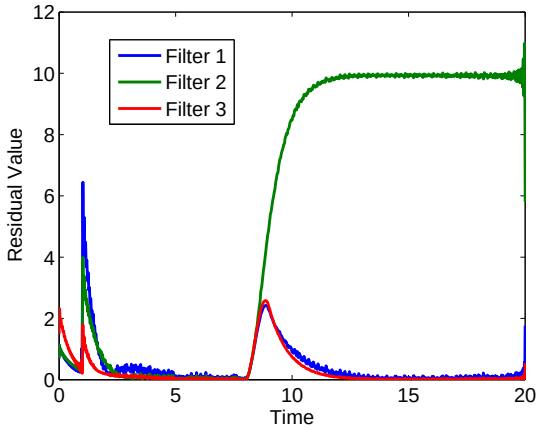


Figure 3.21: Fault identification filters applied to filtered residual.

(3.29) with $\kappa = 1$. We see that the residual is properly filtered and the fault is still visible. Let $f_i(t)$ be defined by (3.29). One R_i for each fault is created according to Prop. 3.3.2. The R_i are applied to the residuals after the noise is eliminated. Figure 3.21 shows f_2 properly identified.

In practice, the residual must be filtered in close to real time. At each time t, it is best to apply the DFT in a given interval $[t - \delta\ t + \delta]$ rather than the whole interval $[0\ t]$ for computational efficiency. To allow for inaccuracies near the endpoints of the interval, we declare detection if the filtered residual exceeds the threshold at some time in a subinterval of $[t-\delta\ t+\delta]$. The optimal choice of δ and the size of the subinterval are important questions for future work.

3.7 Conclusions for Chapter 3

In this chapter, we examined observer based fault detection for systems modeled by differential algebraic equations. We used a combination of the stabilized least squares completion method for DAEs and the Luenberger observer. By observing actual outputs from the real system and comparing them to outputs computed from the observer, we created residuals whose difference from zero indicated the presence of faults. Necessary and sufficient conditions were given for the creation of linear filters that identify faults by giving the residuals unidirectional properties. A frequency filtering technique was employed to make our techniques robust with respect to disturbances by attenuating such factors.

In Section 3.5, we introduced a pair of algorithms for robust fault detection and identification. They can be implemented concurrently or sequentially depending on user preference. Both constructing the fault library and determining the frequency bandwidth where disturbances will lie are problem dependent and may not be applicable in some instances.

Chapter 4

Auxiliary Signal Design

Chapter 3 presents the observer-based approach for FDI in DAEs. This approach fits into the passive category of FDI approaches because it does not act on the system to expose faults. As stated in the introduction, one downside to passive detection is that faults may be masked (by controllers) until they become severe or are not detectable at all until the faulty component becomes necessary for operation (e.g. brakes on a car). An alternative to passive approaches are active approaches: the problem is to design an auxiliary signal exclusively for fault detection and inject it into the system. While passive approaches are usually implemented in an ongoing manner, active tests are either done over short time intervals or over a series of short time intervals. The aim of the auxiliary signal is to facilitate detection, usually by forcing the input-output sets for the nominal and faulty models to be disjoint. An additional goal in auxiliary signal design is to find the smallest such signal that guarantees detection, thereby limiting the effect of the signal on the system. For example, to check the brakes on a car, we would rather tap gently on the brake pedal instead of slam it into the floor. Active detection has been studied extensively for ODE systems in [20, 50, 51, 60].

In this chapter, we consider the analogous auxiliary signal design problem for fault detection in systems that can be modeled by linear DAEs. Since DAEs are singular, the standard techniques for solving this problem in the ODE case are not directly applicable to the DAE case. Hence, this paper explores a previously uninvestigated area of active fault detection. It is important to note that we do not assume our models are index one DAEs as is often done in the literature. We allow for higher index DAE.

The main contribution of this chapter, given in Section 4.1, is a method to determine the optimal inputs for guaranteed active fault diagnosis by solving a bi-level optimization problem in the case where the constraint dynamics form a linear time-invariant DAE. The outer minimum determines the signal of minimum norm such that the inner problem is satisfied. The inner problem guarantees that the inputs being considered are proper (i.e., inputs guarantee the

output measurements are consistent with at most one model). A sufficient condition for a minimal, proper signal to exist is given in Section 4.1.4.

In Section 4.2, we develop an efficient test to correctly identify faulty models given an output. The tests are efficient enough to be done in real time with sensor data. Section 4.3 summarizes the previous sections and reduces them into succinct algorithms for convenient implementation. Section 4.4 provides numerical examples and also addresses numerical concerns. The extension to model uncertainty is discussed in Section 4.6. Section 4.7 explores a subtlety, namely a class of problems where no minimal proper signal will exist. We modify the approach of Section 4.1 and develop a related but alternate algorithm that constructs a nearly minimal proper signal for this class. In Section 4.8, we explore the idea of injecting the test signal and observing the system on possibly overlapping but distinct intervals and find that at times, there is a definite advantage of doing so. Finally, conclusions are given in Section 4.9.

4.1 Minimal Auxiliary Signal

To simplify the discussion, we assume there is only one possible faulty system. Extensions to cases with multiple faulty models can be handled by performing a sequence of tests with each test designed to isolate one faulty case of interest. Alternatively, in some cases one can design a single test signal to handle all of the faulty cases simultaneously.

It is also assumed that both the nominal and the faulty system can be modeled by linear time invariant differential-algebraic equations.

4.1.1 Problem Formulation

The task at hand is to construct a test signal that will let us identify which model is true, the nominal one or the faulty one. Assume the test signal is to be applied over the interval $[0,\ T]$, the observation window is the same as the test interval, and the two models are of the form

$$E_i x_i' + F_i x_i = B_i u + M_i \mu_i \tag{4.1a}$$

$$y_i = C_i x_i + D_i u + N_i \eta_i. \tag{4.1b}$$

$i = 0$ is the normal model and $i = 1$ is the faulty model. u is the test signal and the number of outputs is the same for both models. The matrices $E_i, F_i, B_i, M_i, C_i, D_i$, and N_i are all constant. E_i is singular in at least one of the two models so (4.1) is a singular system. η_i and μ_i are additive noise or model uncertainty terms. The initial conditions, $x_i(0)$, are also uncertain. We assume M_i and N_i are invertible, allowing noise into all equations, and we define the noise measure for

model i as

$$\Gamma_i^2(x_i(0), \eta_i, \mu_i) = \frac{1}{2} x_i(0)^T P_{0_i} x_i(0) + \frac{1}{2} \int_0^T \eta_i^T Q_i \eta_i + \mu_i^T R_i \mu_i \, dt,$$

where $P_{0_i} \geq 0$, $Q_i > 0$, $R_i > 0$. μ_i and η_i are piecewise smooth; thus, they represent exogenous disturbances, modeling error, and biases as opposed to stochastic effects. The weighting matrices Q_i and R_i can be made into identity matrices by letting $\tilde{\mu}_i = R_i^{-\frac{1}{2}} \mu_i$ and $\tilde{\eta}_i = Q_i^{-\frac{1}{2}} \eta_i$. This will simplify the remaining formulas in this section. The new coefficients of μ_i, η_i in (4.1) are now $M_i R_i^{-\frac{1}{2}}$ and $N_i Q_i^{-\frac{1}{2}}$, respectively, however, these new matrices are renamed M_i and N_i for notational convenience in the rest of this paper. Thus, assume the noise measure is already in the form

$$\Gamma_i^2(x_i(0), \eta_i, \mu_i) = \frac{1}{2} x_i(0)^T P_{0_i} x_i(0) + \frac{1}{2} \int_0^T \|\eta_i\|^2 + \|\mu_i\|^2 \, dt. \tag{4.2}$$

There are several ways to measure the total amount of uncertainty. Two previously used total noise measures are

$$\Gamma_\infty(u) = \max\{\Gamma_0, \Gamma_1\}$$
$$\Gamma_{L_2}(u) = \sqrt{\Gamma_0^2 + \Gamma_1^2}.$$

Γ_∞ was used in [20] and a number of related papers. Γ_∞ and Γ_{L^2} each has its advantages. Generally it is easier to find the test signal with Γ_{L^2} and easier to perform the tests with Γ_∞. We will use a combination of the two, but initially use Γ_{L^2} [2, 25]. Note that

$$\Gamma_\infty \leq \Gamma_{L^2} \leq \sqrt{2}\, \Gamma_\infty. \tag{4.3}$$

A proper test signal is one for which the same output cannot come from both models if the noise bound holds. Therefore, a proper test signal found using one uncertainty bound could be used with the other uncertainty bound modulo a 41.4% increase (or decrease) in magnitude of the test signal. For the remainder of this section let $\Gamma = \Gamma_{L^2}$. Assume that the uncertainty is bounded by $\Gamma < \gamma$, where $\gamma > 0$.

As noted, a test signal is known as proper if it is not possible to get the same output from both models given the uncertainty bound. Alternatively, u is proper if the same output holding for both models implies the noise required to created those outputs exceeds the bound.

Definition 4.1.1. *A signal, u, is known as proper if $y_1 - y_0 = 0$ implies*

$$\min_{\substack{x_i(0),\, \mu_i,\, \eta_i \\ i=0,\,1}} \Gamma > \gamma.$$

For a given u, let

$$\begin{aligned}
\phi(u) = \min_{\substack{x_i(0),\mu_i,\eta_i \\ i=0,1}} \Gamma, \text{ such that} \\
E_0 x_0' + F_0 x_0 = B_0 u + M_0 \mu_0 \\
E_1 x_1' + F_1 x_1 = B_1 u + M_1 \mu_1 \\
0 = y_0 - y_1.
\end{aligned}$$

Then u is proper if $\phi(u) \geq \gamma$ and minimal proper if u is the smallest u such that $\phi(u) \geq \gamma$. The control, u, enters the DAE in an affine way so if u is proper, then any larger multiple of u will also be proper. There are several ways to measure the size of u. The one used here is the L^2 norm,

$$\|u\|_{L^2}^2 = \int_0^T u^T u \, d\tau. \tag{4.4}$$

Another common measure in the literature is the amount that u disturbs the performance of the normal system during the test.

The problem is summarized below:

Problem 4.1.1.

$$Find\ J = \inf_u ||u||^2 \tag{4.5a}$$

$$s.t$$

$$\inf_{\substack{x_i(0),\mu_i,\eta_i \\ i=0,1}} \Gamma^2(u) \geq \gamma^2 \tag{4.5b}$$

$$E_0 x_0' + F_0 x_0 = B_0 u + M_0 \mu_0 \tag{4.5c}$$

$$E_1 x_1' + F_1 x_1 = B_1 u + M_1 \mu_1 \tag{4.5d}$$

$$0 = y_0 - y_1. \tag{4.5e}$$

The solution of Problem 4.1.1 is known as the minimal proper u with respect to the L^2 bound. Frequently, the minimal proper u is unique up to a factor of negative one. However, examples can be constructed where it is not unique.

4.1.2 Necessary Conditions and Problem Reformulation

Necessary conditions for the inner minimum can be obtained by performing linear time invariant coordinate changes on the system. First, compute the SVD of E_i

$$E_i = U_i \begin{bmatrix} \Sigma_i & 0 \\ 0 & 0 \end{bmatrix} V_i^T$$

and let

$$\tilde{\Sigma}_i = \begin{bmatrix} \Sigma_i^{-1} & 0 \\ 0 & I \end{bmatrix} U_i.$$

Next, let

$$\tilde{\Sigma}_i U_i^T F_i V_i = \begin{bmatrix} A_{i11} & A_{i12} \\ A_{i21} & A_{i22} \end{bmatrix} \tag{4.6}$$

where A_{i11} is a square matrix with size equal to the size of Σ_i. $\tilde{\Sigma}_i U_i^T M_i$ is invertible, so an orthogonal W_i exists such that

$$\tilde{\Sigma}_i U_i^T M_i W_i = \begin{bmatrix} M_{i11} & M_{i12} \\ 0 & M_{i22} \end{bmatrix}, \tag{4.7}$$

where M_{i11} and M_{i22} are invertible and M_{i11} is the same size as Σ_i. Hence, by pre-multiplying (4.5c) and (4.5d) by $\tilde{\Sigma}_i U_i^T$ and performing the change of coordinates

$$\mu_i = W_i \begin{bmatrix} \mu_{i1} \\ \mu_{i2} \end{bmatrix}, \quad x_i = V_i \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}, \tag{4.8}$$

system (4.5c)–(4.5e) becomes

$$x'_{01} = A_{011}x_{01} + A_{012}x_{02} + B_{01}u + M_{011}\mu_{01} + M_{012}\mu_{02} \tag{4.9a}$$

$$0 = A_{021}x_{01} + A_{022}x_{02} + B_{02}u + M_{022}\mu_{02} \tag{4.9b}$$

$$x'_{11} = A_{111}x_{11} + A_{112}x_{12} + B_{11}u + M_{111}\mu_{11} + M_{112}\mu_{12} \tag{4.9c}$$

$$0 = A_{121}x_{11} + A_{122}x_{12} + B_{12}u + M_{122}\mu_{12} \tag{4.9d}$$

$$\begin{aligned} 0 = {} & C_{01}x_{01} + C_{02}x_{02} + D_0 u + N_0\eta_0 \\ & - (C_{11}x_{11} + C_{12}x_{12} + D_1 u + N_1\eta_1) \end{aligned} \tag{4.9e}$$

with noise measure

$$\Gamma_i^2(x_{i1}(0), x_{i2}(0), \eta_i, \mu_i) = \frac{1}{2}\begin{bmatrix} x_{i1}(0) \\ x_{i2}(0) \end{bmatrix}^T V_i^T P_{0_i} V_i \begin{bmatrix} x_{i1}(0) \\ x_{i2}(0) \end{bmatrix} + \frac{1}{2}\int_0^T \|\eta_i\|^2 + \|\mu_i\|^2 \, dt,$$

where

$$\begin{bmatrix} C_{i1} & C_{i2} \end{bmatrix} = C_i V_i \tag{4.10a}$$

$$\begin{bmatrix} B_{i1} \\ B_{i2} \end{bmatrix} = \tilde{\Sigma} U_i^T B_i. \tag{4.10b}$$

The partitions are conformal with the coordinate change on x_i.

Unlike in most of the literature, here A_{i22} are not assumed to be nonsingular. That is, it is not assumed that the original DAE models are index one. Hence, A_{i22} may have a nontrivial null space. A_{i12} may also have a nontrivial null space. If $x_{i2} \in N(A_{i22}) \cap N(A_{i12})$, then x_{i2} does not affect the dynamics or the constraints and is a free control variable. Since this may cause problems for the optimizer, we will assume that $N(A_{i22}) \cap N(A_{i12}) = \{0\}$.

In the initial model (4.1), x and the μ, η variables played very different roles. However, in (4.9), x_{02}, x_{12}, μ_{ij}, and η_i all play a similar role in acting as what are called algebraic variables as opposed to differential variables. Algebraic variables include both controls and non-differentiated state variables.

Uncertainty on the initial condition of any of the algebraic variables is a situation that can lead to numerical problems during optimization. Generally speaking, optimization software does not allow implementation of such problems (e.g. GPOPS II). Our solution is to find

$$\tilde{P}_i = \begin{bmatrix} \hat{P}_i & 0 \\ 0 & 0 \end{bmatrix} \tag{4.11a}$$

$$\text{such that } \tilde{P}_i \leq V_i^T P_{0_i} V_i \tag{4.11b}$$

$$\hat{P}_i > 0 \tag{4.11c}$$

for each model. Then we get a more conservative noise measure

$$\Gamma_i^2(x_{i1}(0), x_{i2}(0), \eta_i, \mu_i) = \frac{1}{2} x_{i1}^T(0) \hat{P}_i x_{i1}(0) + \frac{1}{2}\int_0^T \|\eta_i\|^2 + \|\mu_i\|^2 \, dt.$$

Hence, the u found using this measure will still be proper for the original measure.

Let $\zeta_1 = \begin{bmatrix} \mu_{02}^T & \mu_{12}^T & \eta_0^T \end{bmatrix}^T$ and $\zeta_2 = \begin{bmatrix} \mu_{01}^T & \mu_{11}^T & \eta_1^T \end{bmatrix}^T$. Then (4.9) has the form

$$z_1' = \hat{A} z_1 + \begin{bmatrix} \hat{D} & \hat{N} \end{bmatrix} \begin{bmatrix} z_2 \\ \zeta_1 \end{bmatrix} + \hat{B} u + \hat{M} \zeta_2 \tag{4.12a}$$

$$0 = \tilde{A} z_1 + \begin{bmatrix} \tilde{D} & \tilde{N} \end{bmatrix} \begin{bmatrix} z_2 \\ \zeta_1 \end{bmatrix} + \tilde{B} u + \tilde{M} \zeta_2, \tag{4.12b}$$

where

$$\hat{A} = \begin{bmatrix} A_{011} & 0 \\ 0 & A_{111} \end{bmatrix}, \ \hat{D} = \begin{bmatrix} A_{012} & 0 \\ 0 & A_{112} \end{bmatrix}, \ \hat{N} = \begin{bmatrix} M_{012} & 0 & 0 \\ 0 & M_{112} & 0 \end{bmatrix} \tag{4.13a}$$

$$\hat{B} = \begin{bmatrix} B_{01} \\ B_{11} \end{bmatrix}, \ \hat{M} = \begin{bmatrix} M_{011} & 0 & 0 \\ 0 & M_{111} & 0 \end{bmatrix}, \ \tilde{A} = \begin{bmatrix} A_{021} & 0 \\ 0 & A_{121} \\ C_{01} & -C_{11} \end{bmatrix} \tag{4.13b}$$

$$\tilde{D} = \begin{bmatrix} A_{022} & 0 \\ 0 & A_{122} \\ C_{02} & -C_{12} \end{bmatrix}, \ \tilde{B} = \begin{bmatrix} B_{02} \\ B_{12} \\ D_0 - D_1 \end{bmatrix}, \ \tilde{M} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -N_1 \end{bmatrix} \tag{4.13c}$$

$$z_1 = \begin{bmatrix} x_{01} \\ x_{11} \end{bmatrix}, \ z_2 = \begin{bmatrix} x_{02} \\ x_{12} \end{bmatrix}, \ \tilde{N} = \begin{bmatrix} M_{022} & 0 & 0 \\ 0 & M_{122} & 0 \\ 0 & 0 & N_0 \end{bmatrix}. \tag{4.13d}$$

$\begin{bmatrix} \tilde{D} & \tilde{N} \end{bmatrix}$ has full row rank and $\tilde{N}$ is invertible. Assume $\tilde{D}$ has full column rank. Then, we can pick a subset of the columns of $\tilde{N}$, call them $\tilde{N}_1$, so that $\begin{bmatrix} \tilde{D} & \tilde{N}_1 \end{bmatrix}$ is invertible. Let $\tilde{N}_2$ denote the remaining columns. Let $\zeta_1 = \begin{bmatrix} \zeta_{11}^T & \zeta_{12}^T \end{bmatrix}^T$ be a conformal partition. Then solving for $\begin{bmatrix} z_2^T & \zeta_{11}^T \end{bmatrix}^T$ yields

$$- \begin{bmatrix} z_2 \\ \zeta_{11} \end{bmatrix} = \begin{bmatrix} \tilde{D} & \tilde{N}_1 \end{bmatrix}^{-1} \left(\tilde{A} z_1 + \tilde{N}_2 \zeta_{12} + \tilde{B} u + \tilde{M} \zeta_2 \right). \tag{4.14}$$

Let $\hat{R} = \begin{bmatrix} \hat{D} & \hat{N}_1 \end{bmatrix}$, where $\hat{N}_1$ are the columns of $\hat{N}$ that are conformal to the selection made

for $\tilde{N}_1$. Eliminating $\begin{bmatrix} z_2 \\ \zeta_{11} \end{bmatrix}$ from the system, (4.12) can be rewritten as

$$z_1' = Az_1 + Bu + N\zeta_{12} + M\zeta_2,$$

where $A = \hat{A} - \hat{R}\begin{bmatrix} \tilde{D} & \tilde{N}_1 \end{bmatrix}^{-1} \tilde{A}$ and B, N, M are defined in the analogous way.

The total uncertainty using the L^2 noise measure is

$$\Gamma^2 = \frac{1}{2}\left(z_1^T(0) \begin{bmatrix} \hat{P}_0 & 0 \\ 0 & \hat{P}_1 \end{bmatrix} z_1(0) + \int_0^T |\zeta_{11}\|^2 + \|\zeta_{12}\|^2 + \|\zeta_2\|^2 \, dt \right).$$

ζ_{11} has been solved for in terms of the other variables. Let $\tilde{A}_0$ denote the bottom half of the matrix $\begin{bmatrix} \tilde{D} & \tilde{N}_1 \end{bmatrix}^{-1} \tilde{A}$ where the partition is conformal with $\begin{bmatrix} z_2^T & \zeta_{11}^T \end{bmatrix}$ and do the same for the other matrices in (4.14). Then,

$$\begin{aligned} \Gamma^2 = {} & \frac{1}{2} z_1^T(0) P_0 z_1(0) + \frac{1}{2}\int_0^T z_1^T \tilde{A}_0^T \tilde{A}_0 z_1 + z_1^T \tilde{A}_0^T \tilde{G}_0 \zeta_{12} + z_1^T \tilde{A}_0^T \tilde{B}_0 u \\ & + z_1^T \tilde{A}_0^T \tilde{M}_0 \zeta_2 + \zeta_{12}^T \tilde{G}_0^T \tilde{A}_0 z_1 + \zeta_{12}^T (I + \tilde{G}_0^T \tilde{G}_0)\zeta_{12} + \zeta_{12}^T \tilde{G}_0^T \tilde{B}_0 u \\ & + \zeta_{12}^T \tilde{G}_0^T \tilde{M}_0 \zeta_2 + u^T \tilde{B}_0^T \tilde{A}_0 z_1 + u^T \tilde{B}_0^T \tilde{G}_0 \zeta_{12} + u^T \tilde{B}_0^T \tilde{B}_0 u \\ & + u^T \tilde{B}_0^T \tilde{M}_0 \zeta_2 + \zeta_2^T \tilde{M}_0^T \tilde{A}_0 z_1 + \zeta_2^T \tilde{M}_0^T \tilde{G}_0 \zeta_{12} + \zeta_2^T \tilde{M}_0^T \tilde{B}_0 u \\ & + \zeta_2^T (I + \tilde{M}_0^T \tilde{M}_0)\zeta_2 \, dt, \end{aligned} \tag{4.15}$$

where

$$P_0 = \begin{bmatrix} \hat{P}_0 & 0 \\ 0 & \hat{P}_1 \end{bmatrix}. \tag{4.16}$$

Minimizing Γ^2 yields the same solution as minimizing Γ so the right-hand side of (4.15) can be minimized directly. Let $I(z_1, \zeta_{12}, \zeta_2)$ denote the quadratic integrand in (4.15). Then the Hamiltonian for this inner problem can be defined by

$$H(z_1, \zeta_{12}, \zeta_2) = \frac{1}{2} I(z_1, \zeta_{12}, \zeta_2) + \lambda^T (Az_1 + Bu + G\zeta_{12} + M\zeta_2),$$

where λ is the Lagrange multiplier. The necessary conditions for a minimum of Γ are

$$z_1' = Az_1 + Bu + G\zeta_{12} + M\zeta_2 \tag{4.17a}$$

$$-\lambda' = A^T\lambda + \tilde{A}_0^T \tilde{A}_0 z_1 + \tilde{A}_0^T \tilde{G}_0 \zeta_{12} + \tilde{A}_0^T \tilde{B}_0 u + \tilde{A}_0^T \tilde{M}_0 \zeta_2 \tag{4.17b}$$

$$0 = G^T\lambda + \tilde{G}_0^T \tilde{A}_0 z_1 + (I + \tilde{G}_0^T \tilde{G}_0)\zeta_{12} + \tilde{G}_0^T \tilde{B}_0 u + \tilde{G}_0^T \tilde{M}_0 \zeta_2 \tag{4.17c}$$

$$0 = M^T\lambda + \tilde{M}_0^T\tilde{A}_0 z_1 + \tilde{M}_0^T\tilde{G}_0\zeta_{12} + \tilde{M}_0^T\tilde{B}_0 u + (I + \tilde{M}_0^T\tilde{M}_0)\zeta_2. \tag{4.17d}$$

The matrix $\begin{bmatrix} I + \tilde{G}_0^T\tilde{G}_0 & \tilde{G}_0^T\tilde{M}_0 \\ \tilde{M}_0^T\tilde{G}_0 & I + \tilde{M}_0^T\tilde{M}_0 \end{bmatrix}$ is of the form $I + L^TL$ which is positive definite independent of the entries of L. Hence, it is invertible and using (4.17c)–(4.17d) the noise variables can be solved for in terms of λ, z_1, and u,

$$-\begin{bmatrix}\zeta_{12}\\ \zeta_2\end{bmatrix} = \begin{bmatrix} I + \tilde{G}_0^T\tilde{G}_0 & \tilde{G}_0^T\tilde{M}_0 \\ \tilde{M}_0^T\tilde{G}_0 & I + \tilde{M}_0^T\tilde{M}_0 \end{bmatrix}^{-1} \left(\begin{bmatrix} G^T \\ M^T \end{bmatrix}\lambda + \begin{bmatrix} \tilde{G}_0^T\tilde{A}_0 \\ \tilde{M}_0^T\tilde{A}_0 \end{bmatrix} z_1 + \begin{bmatrix} \tilde{G}_0^T\tilde{B}_0 \\ \tilde{M}_0^T\tilde{B}_0 \end{bmatrix} u \right). \tag{4.18}$$

Substituting this equation into (4.17a), (4.17b) and relabeling the coefficient matrices, the necessary conditions become

$$\begin{aligned} z_1' &= A_{z_1} z_1 + A_u u + A_\lambda \lambda \\ -\lambda' &= B_{z_1} z_1 + B_u u + B_\lambda \lambda, \end{aligned}$$

where

$$\begin{aligned} A_{z_1} &= A - \begin{bmatrix} G \\ M \end{bmatrix} Q^{-1} \begin{bmatrix} \tilde{G}_0^T\tilde{A}_0 \\ \tilde{M}_0^T\tilde{A}_0 \end{bmatrix} \\ A_u &= B - \begin{bmatrix} G \\ M \end{bmatrix} Q^{-1} \begin{bmatrix} \tilde{G}_0^T\tilde{B}_0 \\ \tilde{M}_0^T\tilde{B}_0 \end{bmatrix} \\ A_\lambda &= - \begin{bmatrix} G \\ M \end{bmatrix} Q^{-1} \begin{bmatrix} G^T \\ M^T \end{bmatrix} \\ B_{z_1} &= \tilde{A}_0^T\tilde{A}_0 - \begin{bmatrix} \tilde{A}_0^T\tilde{G}_0 \\ \tilde{A}_0^T\tilde{M}_0 \end{bmatrix} Q^{-1} \begin{bmatrix} \tilde{G}_0^T\tilde{A}_0 \\ \tilde{M}_0^T\tilde{A}_0 \end{bmatrix} \\ B_u &= \tilde{A}_0^T\tilde{B}_0 - \begin{bmatrix} \tilde{A}_0^T\tilde{G}_0 \\ \tilde{A}_0^T\tilde{M}_0 \end{bmatrix} Q^{-1} \begin{bmatrix} \tilde{G}_0^T\tilde{B}_0 \\ \tilde{M}_0^T\tilde{B}_0 \end{bmatrix} \\ B_\lambda &= A^T - \begin{bmatrix} \tilde{A}_0^T\tilde{G}_0 \\ \tilde{A}_0^T\tilde{M}_0 \end{bmatrix} Q^{-1} \begin{bmatrix} G^T \\ M^T \end{bmatrix} \\ Q &= \begin{bmatrix} I + \tilde{G}_0^T\tilde{G}_0 & \tilde{G}_0^T\tilde{M}_0 \\ \tilde{M}_0^T\tilde{G}_0 & I + \tilde{M}_0^T\tilde{M}_0 \end{bmatrix}. \end{aligned}$$

The boundary conditions are

$$\begin{aligned} \left(z_1^T(0)P_0 + \lambda^T(0)\right) dz_1(0) &= 0 \\ -\lambda^T(T) dz_1(T) &= 0. \end{aligned}$$

Since the initial and final states are free, the boundary conditions reduce to

$$P_0 z_1(0) + \lambda(0) = 0$$
$$\lambda(T) = 0.$$

Using (4.18), the integrand can now be written in terms of z_1 and u. Let $I(z_1, u)$ denote this integrand.

The necessary conditions for the inner minimum are taken as constraints when solving the original problem which is now formulated as an optimization problem. The reformulation is

Problem 4.1.2.

$$\textit{Find } J = \inf_u \|u\|^2 \tag{4.19a}$$

such that

$$z_1' = A_{z_1} z_1 + A_u u + A_\lambda \lambda \tag{4.19b}$$
$$-\lambda' = B_{z_1} z_1 + B_\lambda \lambda + B_u u \tag{4.19c}$$
$$\psi' = \frac{1}{2} I(z_1, u) \tag{4.19d}$$

with the boundary conditions

$$P_0 z_1(0) + \lambda(0) = 0 \tag{4.20a}$$
$$\lambda(T) = 0 \tag{4.20b}$$
$$\psi(0) = \frac{1}{2} z_1^T(0) P_0 z_1(0) \tag{4.20c}$$
$$\psi(T) \geq \gamma^2. \tag{4.20d}$$

(4.19d) is a convenient way to compute the cost of the inner minimization and (4.20d) ensures that u is proper. The coordinate changes necessary to change Problem 4.1.1 into this problem do not affect the auxiliary signal. Any approximations made were conservative in nature. Therefore the auxiliary signal found by solving Problem 4.1.2 will be proper for Problem 4.1.1. Since the latter is easier to solve numerically, we solve it instead.

4.1.3 Assumptions

In the previous section we made two main assumptions. In this section we summarize these and analyze their implications.

If $x_{i2} \in N(A_{i22}) \cap N(A_{i12})$, then x_{i2} does not affect the dynamics or the constraints and is

a free control variable. Since this may cause problems for the optimizer, we assume that

$$\text{Assumption 1:} \quad N(A_{i22}) \cap N(A_{i12}) = \{0\}.$$

During the reformulation we solve for the algebraic control variables. This requires that

$$\text{Assumption 2:} \quad \tilde{D} \text{ has full column rank.}$$

If the models are index one, then A_{i22} is invertible for each i. Hence, if the models are both index one, our assumption on $\tilde{D}$ and our assumption that $N(A_{i12}) \cap N(A_{i22}) = \{0\}$ are met automatically.

If at least one of the models is higher index, then A_{i22} has a null space for at least one i. In this case, for $\tilde{D}$ to have full column rank we need rank $\begin{bmatrix} C_{02} & -C_{12} \end{bmatrix} \geq \text{nullity}(A_{022}) + \text{nullity}(A_{122})$. Therefore, the number of outputs needed is at least the sum of the dimension of the null spaces of A_{i22}. Furthermore, we need $\begin{bmatrix} C_{02} & -C_{12} \end{bmatrix}$ to be one-to-one on $N(A_{022}) \bigoplus N(A_{122})$. Equivalently, we need

$$N(C_{02}) \cap N(A_{022}) = \{0\} \tag{4.21a}$$

$$N(C_{12}) \cap N(A_{122}) = \{0\} \tag{4.21b}$$

$$C_{02}N(A_{012}) \cap C_{12}N(A_{122}) = \{0\}. \tag{4.21c}$$

Essentially (4.21) says that $N(A_{022})$ and $N(A_{122})$ are seen by the output $y_1 - y_0$.

4.1.4 Existence

We now turn to the existence of the minimal, proper test signal in Problem 4.1.1. Recall, after a change of coordinates, the optimal control problem is reformulated as

$$\begin{aligned}
&\min \|u\|_{L^2}^2, \text{ subject to} \\
\Gamma^2(u) = \inf_{z_1,\, \zeta_1,\, \zeta_2} &\frac{1}{2} z_1^T(0) P_0 z_1(0) + \frac{1}{2} \int_0^T \|\zeta_1\|^2 + \|\zeta_2\|^2 \, dt \geq \gamma^2 \\
z_1' &= \hat{A} z_1 + \begin{bmatrix} \hat{D} & \hat{N} \end{bmatrix} \begin{bmatrix} z_2 \\ \zeta_1 \end{bmatrix} + \hat{B} u + \hat{M} \zeta_2 \\
0 &= \tilde{A} z_1 + \begin{bmatrix} \tilde{D} & \tilde{N} \end{bmatrix} \begin{bmatrix} z_2 \\ \zeta_1 \end{bmatrix} + \tilde{B} u + \tilde{M} \zeta_2,
\end{aligned}$$

where the matrices are defined in (4.13). Let $\tilde{N}_1$ be a subset of columns of $\tilde{N}$ so that $\begin{bmatrix}\tilde{D} & \tilde{N}_1\end{bmatrix}$ is invertible. Let $\tilde{N}_2$ be the remaining columns. As before, we can solve for z_2 and part of ζ_1,

$$-\begin{bmatrix} z_2 \\ \zeta_{11}\end{bmatrix} = \begin{bmatrix}\tilde{D} & \tilde{N}_1\end{bmatrix}^{-1}\left(\tilde{A}z_1 + \tilde{N}_2\zeta_{12} + \tilde{B}u + \tilde{M}\zeta_2\right).$$

Let $\tilde{A}_u$ denote the top r rows of the matrix $\begin{bmatrix}\tilde{D} & \tilde{N}_1\end{bmatrix}^{-1}\tilde{A}$ where r is the dimension of z_2. Define $\tilde{N}_u$, $\tilde{B}_u$, $\tilde{M}_u$ in the analogous way. Let $\hat{N}_1$, $\hat{N}_2$ be the columns of $\hat{N}$ that are conformal to the selection made for $\tilde{N}_1$, $\tilde{N}_2$. This time instead of removing the constraint equation, we eliminate z_2 and rewrite the resulting dynamics and constraints,

$$\begin{aligned} z_1' &= Az_1 + Bu + M\nu \\ 0 &= Cz_1 + Du + N\nu, \end{aligned}$$

where

$$\begin{aligned} A &= \hat{A} - \hat{D}\tilde{A}_u \\ B &= \hat{B} - \hat{D}\tilde{B}_u \\ M &= \begin{bmatrix}\hat{N}_1 & \hat{N}_2 - \hat{D}\tilde{N}_u & \hat{M} - \hat{D}\tilde{M}_u\end{bmatrix} \\ C &= \tilde{A} - \tilde{D}\tilde{A}_u \\ D &= \tilde{B} - \tilde{D}\tilde{B}_u \\ N &= \begin{bmatrix}\tilde{N}_1 & \tilde{N}_2 - \tilde{D}\tilde{N}_u & \tilde{M} - \tilde{D}\tilde{M}_u\end{bmatrix} \\ \nu &= \begin{bmatrix}\zeta_{11}^T & \zeta_{12}^T & \zeta_2^T\end{bmatrix}^T. \end{aligned}$$

The inner optimization problem's cost can be expressed

$$\Gamma^2(u) = \inf_{z_1,\,\nu} \frac{1}{2} z_1^T(0) P_0 z_1(0) + \frac{1}{2}\int_0^T \|\nu\|^2\, dt.$$

So, the problem can be reformulated as

$$\text{Find } J = \inf_u ||u||^2 \tag{4.22a}$$

such that

$$\inf_{z_1,\,\nu} \Gamma^2(u) \geq \gamma^2 \tag{4.22b}$$

$$z_1' = Az_1 + Bu + M\nu \tag{4.22c}$$

$$0 = Cz_1 + Du + N\nu. \tag{4.22d}$$

Without loss of generality, we can let $\gamma = 1$. Any other value of γ can be achieved by scaling the constant matrices appropriately.

Consider a related optimization problem:

$$\bar{\delta} = \max_{u \neq 0} \frac{\Gamma^2(u)}{\|u\|^2} \tag{4.23}$$

subject to (4.22c) and (4.22d).

Lemma 4.1.1. *If* (4.23) *has a bounded solution that is attained by* $\tilde{u}$*, then* (4.22) *has a bounded solution that is also attained by* $\tilde{u}$*.*

Proof of Lemma 4.1.1. Assume (4.23) holds. First, note that

$$\Gamma^2(u) \leq \bar{\delta}\|u\|^2, \text{ for all } u \tag{4.24}$$

and $\Gamma(u)$ is quadratic in u. The minimum in (4.22) must occur when $\Gamma^2(u) = 1$ because if not, then by the quadratic nature of Γ we can find a smaller u that satisfies the constraints by scaling u appropriately. Let $\mathcal{U}$ be the set of all feasible u. Then, by (4.24)

$$\min_{u \in \mathcal{U}} \|u\| \geq \frac{1}{\bar{\delta}}. \tag{4.25}$$

On the other hand, suppose the maximum in (4.23) is attained by $\tilde{u}$. Then there exists a sequence $u_i \to \tilde{u}$ such that $\frac{\Gamma^2(u_i)}{\|u_i\|^2} \to \bar{\delta}$. Since we can scale u_i so that $\Gamma^2(u_i) = 1$ for all i we see that

$$\|u_i\|^2 \to \frac{1}{\bar{\delta}}.$$

But, these u_i are also feasible due to the scaling so

$$\min_{u \in \mathcal{U}} \|u\|^2 \leq \|u_i\|^2$$

for all i. The result is that

$$\min_{u \in \mathcal{U}} \|u\|^2 \leq \frac{1}{\bar{\delta}}.$$

Together with (4.25) this implies

$$\min_{u \in \mathcal{U}} \|u\|^2 = \frac{1}{\bar{\delta}}.$$

Therefore, a solution of (4.23) solves (4.22) (and vice versa). □

Now consider yet another related optimization problem

$$\max_u \left(\Gamma^2(u) - \delta\|u\|^2\right) \tag{4.26}$$

subject to (4.22c)-(4.22d) for a positive parameter δ. The solution to the optimization problem in (4.26) is either ∞ or 0, depending on δ. If the optimal cost is zero, then

$$\delta \geq \frac{\Gamma^2(u)}{\|u\|^2}, \text{ for all } u.$$

Hence, $\bar{\delta}$ is the infimum of all δ's for which the solution of this optimization problem is unique and the corresponding cost finite (zero). Theorem 3.3.2 in [20] tells us that (4.26) subject to (4.22c)-(4.22d) has a bounded, unique solution if

1. The matrices D, R_δ^{-1} satisfy $\delta I - D^T R_\delta^{-1} D > 0$;

2. The Riccati equation

$$P' = (A - S_\delta R_\delta^{-1} C)P + P(A - S_\delta R_\delta^{-1} C)^T Q_\delta - S_\delta R_\delta^{-1} S_\delta^T, \; P(0) = P_0^{-1}$$

has a solution, where

$$\begin{bmatrix} Q_\delta & S_\delta \\ S_\delta^T & R_\delta \end{bmatrix} = \begin{bmatrix} M & B \\ N & D \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -\delta I \end{bmatrix}^{-1} \begin{bmatrix} M & B \\ N & D \end{bmatrix}^T.$$

Therefore, if the above conditions are met (4.26) has a unique solution for the given δ. Taking the infimum over the set of δ such that these conditions are met gives $\bar{\delta}$. Therefore, as long as this set is nonempty, (4.23) and thus, by Lemma 4.1.1, (4.22) has a bounded solution. Since (4.22) and Problem 4.1.1 are equivalent problems, the latter also has a bounded solution. This argument is summarized in the following

Theorem 4.1.1. *If there exists a δ such that the above conditions 1 and 2 are met, then there exists a solution to Problem 4.1.1.*

The initial condition for the Riccati equation requires P_0 to be invertible, but we assumed $P_{0_i} \geq 0$ for each i, permitting $P_0 = 0$. While the sufficient condition breaks down in this case, there are still many problems with $P_{0_i} = 0$ for which there exists a minimal proper signal.

4.1.5 Checking Minimality

Suppose u is the numerical solution to Problem 4.1.2. It is good to be able to verify that it is proper and close to minimal proper. Suppose u is minimal proper and let $u = \alpha u$ for $\alpha > 0$.

Consider the problem of finding $J = \min \|y_0 - y_1\|^2$ subject to (4.1a) for $i = 0, 1$ and $\Gamma^2 < \gamma^2$. If $\alpha > 1$, then $J > 0$ and if $0 < \alpha < 1$, then $J = 0$ should hold. Therefore, one way to check that u is minimal proper is to solve this problem and verify these conditions.

We proceed as before when we reformulated (4.5) into useful necessary conditions. In particular, (4.12a) and (4.12b) hold except (4.12b) is missing the last block row because here it is not assumed that $y_0 = y_1$. Thus, we have

$$\begin{aligned} z_1' &= \hat{A} z_1 + \begin{bmatrix} \hat{D} & \hat{N} \end{bmatrix} \begin{bmatrix} z_2 \\ \zeta_1 \end{bmatrix} + \hat{B} u + \hat{M} \zeta_2 \\ 0 &= \bar{A} z_1 + \begin{bmatrix} \bar{D} & \bar{N} \end{bmatrix} \begin{bmatrix} z_2 \\ \zeta_1 \end{bmatrix} + \bar{B} u, \end{aligned}$$

where

$$\hat{A} = \begin{bmatrix} A_{011} & 0 \\ 0 & A_{111} \end{bmatrix}, \ \hat{D} = \begin{bmatrix} A_{012} & 0 \\ 0 & A_{112} \end{bmatrix}, \ \hat{N} = \begin{bmatrix} M_{012} & 0 & 0 \\ 0 & M_{112} & 0 \end{bmatrix}, \ \hat{B} = \begin{bmatrix} B_{01} \\ B_{11} \end{bmatrix}$$

$$\hat{M} = \begin{bmatrix} M_{011} & 0 & 0 \\ 0 & M_{111} & 0 \end{bmatrix}, \ \bar{A} = \begin{bmatrix} A_{021} & 0 \\ 0 & A_{121} \end{bmatrix}, \ \bar{D} = \begin{bmatrix} A_{022} & 0 \\ 0 & A_{122} \end{bmatrix}, \ \bar{B} = \begin{bmatrix} B_{02} \\ B_{12} \end{bmatrix}$$

$$z_1 = \begin{bmatrix} x_{01} \\ x_{11} \end{bmatrix}, \ z_2 = \begin{bmatrix} x_{02} \\ x_{12} \end{bmatrix}, \ \bar{N} = \begin{bmatrix} M_{022} & 0 & 0 \\ 0 & M_{122} & 0 \end{bmatrix}.$$

Recall, M_{i22} are invertible and $\zeta_1 = \begin{bmatrix} \mu_{02}^T & \mu_{12}^T & \eta_0^T \end{bmatrix}^T$. Therefore, $\bar{N}$ has an invertible block and μ_{02}, μ_{12} can be written explicitly in terms of the other quantities

$$- \begin{bmatrix} \mu_{02} \\ \mu_{12} \end{bmatrix} = \begin{bmatrix} M_{022} & 0 \\ 0 & M_{122} \end{bmatrix}^{-1} \left(\bar{A} z_1 + \bar{D} z_2 + \bar{B} u \right).$$

Letting $\hat{N}_0 = \begin{bmatrix} M_{012} & 0 \\ 0 & M_{112} \end{bmatrix}$, the dynamics can be rewritten

$$\begin{aligned} z_1' &= \left(\hat{A} - \hat{N}_0 \begin{bmatrix} M_{022} & 0 \\ 0 & M_{122} \end{bmatrix}^{-1} \bar{A} \right) z_1 + \left(\hat{D} - \hat{N}_0 \begin{bmatrix} M_{022} & 0 \\ 0 & M_{122} \end{bmatrix}^{-1} \bar{D} \right) z_2 \\ &\quad + \left(\hat{B} - \hat{N}_0 \begin{bmatrix} M_{022} & 0 \\ 0 & M_{122} \end{bmatrix}^{-1} \bar{B} \right) u + \hat{M} \zeta_2 \\ &= \breve{A} z_1 + \breve{D} z_2 + \breve{B} u + \breve{M} \zeta_2 \end{aligned}$$

by defining the matrices $\breve{A}$, $\breve{D}$, $\breve{B}$ and $\breve{M}$ appropriately. The noise measure is

$$\Gamma^2 = \frac{1}{2} z_1^T(0) P_0 z_1(0) + \frac{1}{2} \int_0^T \|\eta_0\|^2 + \|\zeta_2\|^2 + \left\| \begin{bmatrix} \mu_{02} \\ \mu_{12} \end{bmatrix} \right\|^2 dt.$$

Therefore, we must find

$$J = \min \|y_1 - y_0\|^2 \tag{4.27a}$$

such that

$$z_1' = \breve{A} z_1 + \breve{D} z_2 + \breve{B} u + \breve{M} \zeta_2 \tag{4.27b}$$

$$\psi' = \frac{1}{2} \left(\|\eta_0\|^2 + \|\zeta_2\|^2 + \left\| \begin{bmatrix} \mu_{02} \\ \mu_{12} \end{bmatrix} \right\|^2 \right) \tag{4.27c}$$

$$\psi(0) = \frac{1}{2} z_1(0)^T P_0 z_1(0), \quad \psi(T) < \gamma^2 \tag{4.27d}$$

where y_i is given in (4.1b) and u is fixed. If u is proper, then solving this problem should yield $J > 0$. If u is minimal proper, then for any smaller u we should confirm $J = 0$.

4.2 Model Identification

We now turn to the problem of deciding which model is correct based on measurements y. In [20], Γ_∞ is used to measure the noise, meaning that the output from one model is independent of the noises in the other model. The advantage to this approach is that geometrically one can think of two independent, potentially overlapping output sets that are made disjoint by a proper auxiliary signal. This work has avoided using Γ_∞ because there are algebraic and computational complications that arise. Fortunately, due to (4.3), the proper auxiliary signal found by solving Problem 4.1.2 using the Γ_{L^2} bound is proper for the Γ_∞ bound modulo a 41.4% ($u = \sqrt{2}u$) increase in magnitude of the auxiliary signal. It is likely that the scaled signal will not be minimal proper for the max bound; however, it is still proper so the output sets are still disjoint. Observations tell us the signal will be close to optimal. If a smaller signal is desired, note that Section 4 gives an algorithm to improve the optimality.

The realizability of an input-output pair $\{u, y\}$ by a given model can be tested using a single inequality test based on the noise bound. In particular, suppose we find

$$W_i = \min_{x_i(0),\, \eta_i,\, \mu_i} \left(\frac{1}{2} x_{i1}(0)^T \hat{P}_i x_{i1}(0) + \frac{1}{2} \int_0^T \|\eta_i\|^2 + \|\mu_{i1}\|^2 + \|\mu_{i2}\|^2 \, dt \right) \tag{4.28}$$

subject to the constraints

$$x'_{i1} = A_{i11}x_{i1} + A_{i12}x_{i2} + B_{i1}u + M_{i11}\mu_{i1} + M_{i12}\mu_{i2} \tag{4.29a}$$

$$0 = A_{i21}x_{i1} + A_{i22}x_{i2} + B_{i2}u + M_{i22}\mu_{i2} \tag{4.29b}$$

$$y = C_{i1}x_{i1} + C_{i2}x_{i2} + D_iu + N_i\eta_i, \tag{4.29c}$$

where the constraints were initially derived in and around (4.9). In the following definition, we formalize the idea of a realizable model, that is, a model that yields permissable output given an input.

Definition 4.2.1. *Model i is realizable for a given control u if $W_i < \gamma^2$, where W_i is defined in* (4.28) *for models $i = 0, 1$.*

In this section the measured output, y, is taken from on-line measurements of the system. From an efficiency and/or safety standpoint it is desirable to detect and identify faults in real-time, or as close to real-time as possible. This forces the realizability test to be computationally efficient. Accordingly, solving complex optimal control problems (e.g. (4.28) and (4.29)) is computationally prohibitive. In the following, we present an alternative that is suitable for real-time FDI.

In what follows we omit the model number i and remember that the following procedure is done for both models $i = 0, 1$, separately. Letting

$$\tilde{D}_0 = \begin{bmatrix} A_{i22} \\ C_{i2} \end{bmatrix}, \quad \tilde{N}_0 = \begin{bmatrix} 0 & M_{i22} & 0 \\ 0 & 0 & N_i \end{bmatrix}, \quad P^{-1} = \hat{P}_i$$

$$\tilde{B}_0 = \begin{bmatrix} B_{i2} \\ D_i \end{bmatrix}, \quad \tilde{A}_0 = \begin{bmatrix} A_{i21} \\ C_{i1} \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ y \end{bmatrix}$$

$$\hat{N}_0 = \begin{bmatrix} M_{i11} & M_{i12} & 0 \end{bmatrix}, \quad \hat{B}_0 = B_{i1}, \quad \zeta = \begin{bmatrix} \mu_{i1} \\ \mu i2 \\ \nu_i \end{bmatrix}$$

$$x = x_{i1}, \quad \bar{x} = x_{i2}, \quad A_{11} = A_{i11}, \quad \hat{D}_0 = A_{i12},$$

(4.29) becomes

$$x' = A_{11}x + \hat{B}_0u + \hat{D}_0\bar{x} + \hat{N}_0\zeta$$

$$b = \tilde{A}_0x + \tilde{B}_0u + \tilde{D}_0\bar{x} + \tilde{N}_0\zeta.$$

Recall in Section 4.1.3 we assume

$$\tilde{D} = \begin{bmatrix} A_{022} & 0 \\ 0 & A_{122} \\ C_{02} & C_{12} \end{bmatrix}$$

has full column rank. Therefore, $\tilde{D}_0$ also has full column rank. $\tilde{N}_0$ has full row rank and an invertible 2×2 block. Hence, we can select a subset of columns of $\tilde{N}_0$, call them $\tilde{N}_{01}$ such that $\begin{bmatrix} \tilde{D}_0 & \tilde{N}_{01} \end{bmatrix}$ is invertible. Let $\tilde{N}_{02}$ denote the remaining columns and $\zeta = \begin{bmatrix} \tilde{\nu} \\ \nu \end{bmatrix}$ be a conformal partition and ordering. Then, we can solve for $\bar{x}$ and some of the noise $\tilde{\nu}$

$$\begin{bmatrix} \bar{x} \\ \tilde{\nu} \end{bmatrix} = - \begin{bmatrix} \tilde{D}_0 & \tilde{N}_{01} \end{bmatrix}^{-1} \left[\tilde{A}_0 x + \tilde{B}_0 u - b + \tilde{N}_{02} \nu \right].$$

This eliminates the constraint equation and modifies the cost, so now we have

$$\begin{aligned} W = \min_{x(0),\nu} \frac{1}{2} & \left(x(0)^T P^{-1} x(0) + \int_0^T \nu^T \nu + \| Q_1 \nu + Q_2 x + c \|^2 \, dt \right) \\ x' = & \, Ax + N\nu + a \end{aligned}$$

where

$$\begin{aligned} \begin{bmatrix} * \\ Q_1 \end{bmatrix} &= - \begin{bmatrix} \tilde{D}_0 & \tilde{N}_{01} \end{bmatrix}^{-1} \tilde{N}_{02} \left(\text{partition conformal with } \begin{bmatrix} \bar{x} \\ \tilde{\nu} \end{bmatrix} \right) \\ \begin{bmatrix} * \\ Q_2 \end{bmatrix} &= - \begin{bmatrix} \tilde{D}_0 & \tilde{N}_{01} \end{bmatrix}^{-1} \tilde{A}_0 \\ \begin{bmatrix} * \\ c \end{bmatrix} &= - \begin{bmatrix} \tilde{D}_0 & \tilde{N}_{01} \end{bmatrix}^{-1} \left(\tilde{B}_0 u - b \right) \\ A &= A_{11} - \hat{R}_0 \begin{bmatrix} \tilde{D}_0 & \tilde{N}_{01} \end{bmatrix}^{-1} \tilde{A}_0 \\ N &= \hat{N}_{02} - \hat{R}_0 \begin{bmatrix} \tilde{D}_0 & \tilde{N}_{01} \end{bmatrix}^{-1} \tilde{N}_{02} \\ a &= (\hat{B}_0 - \hat{R}_0 \begin{bmatrix} \tilde{D}_0 & \tilde{N}_{01} \end{bmatrix}^{-1} \tilde{B}_0) u + \hat{R}_0 \begin{bmatrix} \tilde{D}_0 & \tilde{N}_{01} \end{bmatrix}^{-1} b. \end{aligned}$$

$\hat{N}_0 = \begin{bmatrix} \hat{N}_{01} & \hat{N}_{02} \end{bmatrix}$ is a conformal partition and ordering with $\zeta = \begin{bmatrix} \tilde{\nu} \\ \nu \end{bmatrix}$ and $\hat{R}_0 = \begin{bmatrix} \hat{D}_0 & \hat{N}_{01} \end{bmatrix}$. The next theorem takes advantage of this problem reformulation.

Theorem 4.2.1. *Consider the problem*

$$W = \min_{x(0),\nu} \frac{1}{2}\left(x(0)^T P^{-1} x(0) + \int_0^T \nu^T \nu \right.$$
$$\left. + (Q_1\nu + Q_2 x + c)^T (Q_1\nu + Q_2 x + c)\, dt \right) \tag{4.30a}$$
$$x' = Ax + N\nu + a, \tag{4.30b}$$

where $P^{-1} > 0$ *and* $x(0), x(T)$ *are free. Let* $Q = (I + Q_1^T Q_1)$*. This optimization problem has a unique solution with past cost given by*

$$W(t) = \frac{1}{2} x^T K(t)^{-1} x - v(t)^T x + \phi(t), \tag{4.31}$$

where

$$K' = AK + KA^T - NQ^{-1}Q_1^T Q_2 K + NQ^{-1}N^T +$$
$$K\left(Q_2^T Q_1 Q^{-1} Q_1^T Q_2 - Q_2^T Q_2\right) K - KQ_2^T Q_1 Q^{-1} N^T, K(0) = P \tag{4.32a}$$
$$v' = -K^{-1}NQ^{-1}Q_1^T c - K^{-1}NQ^{-1}N^T v + K^{-1}a - A^T v$$
$$+ Q_2^T Q_1 Q^{-1} Q_1^T c + Q_2^T Q_1 Q^{-1} N^T v - Q_2^T c,\ v(0) = 0 \tag{4.32b}$$
$$\phi' = -v^T NQ^{-1}Q_1^T c + v^T a - \frac{1}{2} v^T NQ^{-1}N^T v + \frac{1}{2} c^T c$$
$$- \frac{1}{2} c^T Q_1 Q^{-1} Q_1^T c,\ \phi(0) = 0. \tag{4.32c}$$

The optimal past cost is realized by the optimal (smallest) noise

$$\nu = -Q^{-1}\left(Q_1^T Q_2 x + Q_1^T c - N^T (K^{-1}x - v)\right). \tag{4.33}$$

Proof. The Hamiltonian is

$$H = \frac{1}{2}\nu^T Q\nu + \frac{1}{2} x^T Q_2^T Q_2 x + \nu^T Q_1^T Q_2 x$$
$$+ \nu^T Q_1^T c + x^T Q_2^T c + \frac{1}{2} c^T c + \lambda^T (Ax + N\nu + a).$$

The necessary conditions are

$$x' = Ax + N\nu + a$$
$$-\lambda' = A^T\lambda + Q_2^T Q_2 x + Q_2^T Q_1 \nu + Q_2^T c$$
$$0 = Q\nu + Q_1^T Q_2 x + Q_1^T c + N^T \lambda$$

and the boundary conditions are

$$\lambda(0) = -P^{-1}x(0)$$
$$\lambda(T) = 0.$$

This makes the optimal noise

$$\nu = -Q^{-1}\left(Q_1^T Q_2 x + Q_1^T c + N^T \lambda\right). \tag{4.34}$$

Assume $\lambda(t)$ and $x(t)$ satisfy this linear relation

$$\lambda(t) = -K^{-1}(t)x(t) + v(t),$$

where $K(0) = P$ and $v(0) = 0$ due to the boundary conditions. If we can find such $K(t)$ and $v(t)$ then the assumption is valid. The sweep method [11] yields (4.32a) and (4.32b). Therefore, the assumption is valid and note in light of this, (4.34) agrees with (4.33).

It remains to show (4.31), (4.32c), and that the solution is unique. We use the forward dynamic programming approach [5] by defining the past cost $L(t, x)$ with the additional constraint $x(t) = x$. The forward dynamic programming equation is

$$\begin{aligned}\frac{\partial L}{\partial t} = \min_{\nu} -\frac{\partial L}{\partial x}^T (Ax + N\nu + a) + \frac{1}{2}\nu^T Q\nu + \frac{1}{2}x^T Q_2^T Q_2 x \\ + \nu^T Q_1^T Q_2 x + \nu^T Q_1^T c + x^T Q_2^T c + \frac{1}{2}c^T c,\end{aligned} \tag{4.35}$$

where the minimization is subject to (4.30b). The minimizer is
$\nu = -Q^{-1}\left(Q_1^T Q_2 x + Q_1^T c - N^T \frac{\partial L}{\partial x}\right)$.

On the other hand, assume the past cost can be expressed

$$L(t, x) = \frac{1}{2}x^T K^{-1} x - v^T x + \phi(t),\ \phi(0) = 0, \tag{4.36}$$

where $\phi(t)$ is to be determined. Then, we have

$$\frac{\partial L}{\partial t} = -\frac{1}{2}x^T (K^{-1}K'K^{-1})x - v'^T x + \phi'$$
$$\frac{\partial L}{\partial x} = K^{-1}x - v.$$

This together with the right hand side of (4.35) determines $\phi(t)$ and this agrees with (4.32c). Since we found a valid ϕ, the assumption in (4.36) is valid and $L = W$, so (4.31) holds. Since W is the solution to the HJB equation and K, v, ϕ are unique because they are solutions to

differential equations, the solution we found is unique. □

Let us return to the problem in (4.28) and (4.29). Theorem 4.2.1 gives an explicit, easy to compute formula for the past cost, however, the unknown state, x, is involved in the expression. We have the dynamics of x, but not the initial condition.

Substituting the optimal noise into the dynamics of the state yields

$$x' = (A - NQ^{-1}Q_1^TQ_2 + NQ^{-1}N^TK^{-1})x + (a - NQ^{-1}Q_1^Tc - NQ^{-1}N^Tv).$$

The latter term is independent of x. Thus,

$$x(t) = \Phi(t,0)x(0) + \int_0^t \Phi(t,s)\left(a - NQ^{-1}Q_1^Tc - NQ^{-1}N^Tv\right)\,ds, \tag{4.37}$$

where $\Phi(t,0)$ satisfies

$$\frac{d\Phi(t,0)}{dt} = \left(A - NQ^{-1}Q_1^TQ_2 + NQ^{-1}N^TK^{-1}\right)\Phi(t,0) \tag{4.38a}$$

$$\Phi(\tau,\tau) = I,\ \forall\tau. \tag{4.38b}$$

Using (4.37) in the past cost yields

$$\begin{aligned} W(t) = &\frac{1}{2}\left[x^T(0)\Phi^T(t,0) + f^T(t)\right]K^{-1}(t)\left[\Phi(t,0)x(0) + f(t)\right] \\ &- v^T\left[\Phi(t,0)x(0) + f(t)\right] + \phi(t), \end{aligned} \tag{4.39}$$

where

$$f(t) = \int_0^t \Phi(t,s)\left(a - NQ^{-1}Q_1^Tc - NQ^{-1}N^Tv\right)\,ds.$$

$x(0)$ must minimize the cost, so differentiating (4.39) with respect to $x(0)$ and setting the result equal to zero yields an explicit representation for the minimum initial condition with respect to the past cost at time t,

$$x(0) = \Phi^{-1}(t,0)\left[K(t)v - f(t)\right]. \tag{4.40}$$

All quantities in this expression are known or can be found by numerical integration. $f(t)$ is found by numerically solving the differential equation

$$f'(t) = \left(A - NQ^{-1}Q_1^TQ_2 + NQ^{-1}N^TK^{-1}\right)f(t) + \left(a - NQ^{-1}Q_1^Tc - NQ^{-1}N^Tv\right) \tag{4.41}$$

with zero initial conditions.

Therefore, to utilize this approach we must integrate (4.32), (4.38), and (4.41) numerically.

At each time step in the numerical integrator, (4.39) gives us the optimal past cost, where $x(0)$ is given in (4.40). If at any time, t, the noise violates the bound then the model under consideration is not realizable. If u is a proper signal this condition holds for exactly one model. Theoretically, it is only necessary to implement a single realizability test for one of the two models because exactly one model will pass the test. However, model and numerical error require the use of both tests.

The realizability test in this section has the ability to be implemented in real-time. In other words, as measurements are made available $W(t)$ can be calculated. This can result in faster detection because a non-realizable model can be detected before the end of the test interval. If $W(t)$ exceeds the noise bound by some threshold, the test can be stopped and a fault is declared. In safety related applications, the system may be shut down before further risk is undertaken. When efficiency is the main goal, finding the fault earlier could result in mitigating financial loss.

4.3 Algorithms

Given a proper signal for the L^2 or the L_∞ bound, the test in (4.27) can be applied as an ad hoc method to find a smaller proper signal for that bound. Fix u and suppose we solve

Problem 4.3.1.

$$J = \min \|y_1 - y_0\|^2$$

with the constraints

$$z_1' = \breve{A} z_1 + \breve{D} z_2 + \breve{B} u + \breve{M} \zeta_2 \tag{4.42a}$$

$$\left.\begin{aligned} \psi' &= \frac{1}{2}\left(\|\eta_0\|^2 + \|\zeta_2\|^2 + \left\| \begin{bmatrix} \mu_{02} \\ \mu_{12} \end{bmatrix} \right\|^2 \right) \\ \psi(0) &= \frac{1}{2} z_1(0)^T P_0 z_1(0), \quad \psi(T) < \gamma^2 \end{aligned}\right\} \quad \textit{For } L^2 \textit{ bound} \tag{4.42b}$$

$$\left.\begin{aligned} \psi_0' &= \frac{1}{2}\left(\|\eta_0\|^2 + \zeta_2^T \, diag(I,0,0) \zeta_2 + \|\mu_{02}\|^2 \right) \\ \psi_0(0) &= \frac{1}{2} z_1(0)^T \, diag(\hat{P}_0, 0) z_1(0), \quad \psi_0(T) < \gamma^2 \\ \psi_1' &= \frac{1}{2}\left(\zeta_2^T \, diag(0,I,I) \zeta_2 + \|\mu_{12}\|^2 \right) \\ \psi_1(0) &= \frac{1}{2} z_1(0)^T \, diag(0, \hat{P}_1) z_1(0), \quad \psi_1(T) < \gamma^2 \end{aligned}\right\} \quad \textit{For } L_\infty \textit{ bound} \tag{4.42c}$$

Recall, from Section 4.1.5, if $J = 0$, then the signal is not proper with respect to the bound we implement, either (4.42b) or (4.42c). If $J > 0$ we can conclude that the signal is proper.

In Algorithm 4.1, we assume 0 is not a proper signal and u is the signal from solving Problem 4.1.2. Algorithm 4.1 shrinks u using a scaling parameter, α, until the relative error is within τ, for $\tau > 0$. The relative error is defined as

$$\frac{||u_{\text{test}} - u^*||}{||u^*||},$$

where u^* is the true minimal proper signal with respect to the L_∞ bound and u_{test} is a scaled test signal based on u (see Algorithm 4.1). u^* is unknown. A good alternative is to use the last known proper signal, u_{proper}, in place of u^*. Then we calculate

$$\tau_r = \frac{||u_{\text{test}} - u_{\text{proper}}||}{||u_{\text{proper}}||}$$

when the relative error is desired. Indeed, if u^* is close to u in terms of shape, using u_{proper} in place of u^* means that the relative error will be less than τ.

We now briefly digress to support the claim that the shape of u^* is close to the shape of u. Recall the L_∞ bound is $\Gamma^2_\infty(u) = \max\{\Gamma^2_0, \Gamma^2_1\}$. To write a more easily computable form we will need the fact that the maximum of two numbers can be written as

$$\max\{c_1, c_2\} = \max_{0 \le \beta \le 1} \beta c_1 + (1-\beta)c_2.$$

Using this fact, we can rewrite the L_∞ bound as

$$\Gamma^2_\infty = \inf_{\substack{x_0(0), x_1(0) \\ \mu_0, \mu_1, \eta_0, \eta_1}} \max_{\beta \in [0,\,1]} \beta\Gamma^2_0 + (1-\beta)\Gamma^2_1.$$

Since M_i, N_i are invertible for $i = 0, 1$ the "inf max" can be rewritten as a "max inf" (see [20]). Therefore,

$$\Gamma^2_\infty = \max_{\beta \in [0,\,1]} \inf_{\substack{x_0(0), x_1(0) \\ \mu_0, \mu_1, \eta_0, \eta_1}} \beta\Gamma^2_0 + (1-\beta)\Gamma^2_1. \tag{4.43}$$

The next lemma shows that when $\beta = 0.5$ the optimal proper signal for the L^2 bound is the same as the optimal proper signal for the L_∞ signal up to scaling. Hence, the shape of u^* is the same as the shape of u if $\beta = 0.5$. Simulations have shown β falls within the interval $[0.3,\ 0.7]$. Since β typically falls in an interval close to 0.5 we expect the shapes to be close.

Lemma 4.3.1. *Let u^* be the minimal proper signal with respect to the L^2 bound. Fix $\beta = 0.5$ in (4.43). Then the minimal proper signal with respect to the L_∞ bound and $\beta = 0.5$ is $\sqrt{2}u^*$.*

Proof. From (4.3) it is already clear that $\sqrt{2}u^*$ is proper for the L_∞ bound. It remains to show that it is the smallest proper signal. Suppose there is a smaller proper signal, $\tilde{u}$ and note that

with $\beta = 0.5$,

$$\Gamma_\infty^2(u) = \frac{1}{2} \inf_{\substack{x_0(0),\, x_1(0) \\ \mu_0,\, \mu_1,\, \eta_0,\, \eta_1}} \left(\Gamma_0^2 + \Gamma_1^2\right) = \frac{1}{2}\Gamma_{L^2}^2(u).$$

Since $\tilde{u}$ is proper for the L_∞ bound, $\frac{1}{2}\Gamma_{L^2}^2(\tilde{u}) \geq \gamma^2$. Due to the quadratic nature of $\Gamma_{L^2}^2$,

$$\Gamma_{L^2}^2\left(\frac{\tilde{u}}{\sqrt{2}}\right) = \frac{1}{2}\Gamma_{L^2}^2(\tilde{u}) \geq \gamma^2.$$

Therefore, $\frac{\tilde{u}}{\sqrt{2}}$ is proper for the L^2 bound. However,

$$\left\| \frac{\tilde{u}}{\sqrt{2}} \right\| < \|u^*\|.$$

But this is a contradiction because u^* is assumed to be minimal proper for the L^2 bound. $\square$

Let N be the number of iterations the user is willing to invest and select a tolerance τ. Algorithm 4.1 will terminate when either the relative error is less than τ or the maximum number of iterations has been reached. The algorithm returns u_{proper}, the last known proper signal found. The auxiliary signal is almost always found in advance of the operation of the system, so the extra computing time required to use this algorithm is not detrimental to FDI.

$n \leftarrow 0$; $n_0 \leftarrow 1$; $\alpha \leftarrow 1$
$u_{\text{proper}} \leftarrow u$
while $n < N$ **do**
  $\alpha \leftarrow \alpha - 10^{-n_0}$;
  $u_{\text{test}} \leftarrow \alpha u$;
  Find J by solving Problem 4.3.1;
  $n \leftarrow n + 1$;
  if $J = 0$ **then**
    if $\tau_r < \tau$ **then**
      return u_{proper};
    else
      $\alpha \leftarrow \alpha + 10^{-n_0}$;
      $n_0 \leftarrow n_0 + 1$;
    end
  else
    $u_{\text{proper}} \leftarrow u_{\text{test}}$;
  end
end

Algorithm 4.1: Ad hoc minimization of proper signal.

We have developed all the pieces necessary to perform real-time FDI. In the following, we summarize how these pieces fit together in an FDI algorithm. Our approach is as follows: 1) solve Problem 4.1.2 to find a minimal proper u for the bound $\Gamma_{L^2}^2 < \gamma$. Let $\tilde{u} = \sqrt{2}u$. Note that $\tilde{u}$ is a proper signal for $\Gamma_\infty^2 < \gamma$. Observation tells us it will also be close to minimal proper; 2) Use Algorithm 4.1 to shrink the signal so that it is approximately minimal proper with respect to the Γ_∞ bound; 3) Use the resulting signal to perform the model identification in Section 4.2. If the size of the auxiliary signal is of great consequence for a particular application, it is advisable to do step 2) before using the signal online in the model identification techniques. This complete FDI algorithm is summarized in Algorithm 4.2.

1. Solve Problem 4.1.2 to find a minimal proper u for the bound $\Gamma_{L^2}^2 < \gamma^2$. Let $\tilde{u} = \sqrt{2}u$ and note that $\tilde{u}$ is a proper signal for $\Gamma_\infty^2 < \gamma^2$.

2. Use Algorithm 4.1 to shrink the signal so that it is approximately minimal proper with respect to the Γ_∞ bound.

3. Use the resulting signal as input in the model identification in Section 4.2.

   (a) Integrate (4.32), (4.38), and (4.41) numerically.

   (b) Find $x(0)$ from (4.40) and use the result to calculate the past cost (4.39).

   (c) If $W(t) > \gamma^2$ for any t then the model under consideration is not realizable.

   (d) Repeat (a)-(c) for each model.

Algorithm 4.2: Complete algorithm for FDI.

4.4 Numerical Simulations

High-quality optimization software is required to solve Problem 4.1.2. There are a variety of good optimization software packages available. In order to handle the specific formulations presented in this section the software needs to accept optimal control problems with equality and inequality constraints both inside the time interval and at the endpoints. The particular package used in the examples in this paper is GPOPS-II developed at the University of Florida [35–37, 58]. The method employed by GPOPS-II is an *hp*-adaptive version of the Radau pseudospectral method. This is a direct transcription approach where the integration is done with an orthogonal collocation Gaussian quadrature implicit integration method where collocation is performed at the Legendre-Gauss-Radau points. SOS (Sparse Optimization Suite), as developed by Applied Mathematical Analysis LLC (http://www.appliedmathematicalanalysis.com) is another example of a direct transcription code commonly used for optimal control problems. In general, direct transcription refers to any approach where all differential equations, integrals, and constraints are discretized. The resulting large nonlinear programming (NLP) problem is then passed to an optimization package. The grid is automatically refined until the desired tolerance is met. The user supplies all equations and constraints. A listing of the MATLAB codes used in this section is available in Appendix A.1.

4.4.1 Proper u

Example 4.4.1. *Consider the following ODE example from Section 4.2.7 of [20]. An ODE model is chosen as our first example to show our software duplicates results in the literature*

[20]. The normal (0) and faulty (1) models are given by

$$
\begin{aligned}
x_0' &= \begin{bmatrix} 0 & 1 \\ -9 & 0 \end{bmatrix} x_0 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \mu_0 \\
y_0 &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_0 + \eta_0 \\
x_1' &= \begin{bmatrix} 0 & 1 \\ -4 & 0 \end{bmatrix} x_1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \mu_1 \\
y_1 &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_1 + \eta_1,
\end{aligned}
$$

with the noise bound

$$
\Gamma^2(\mu_0,\, \mu_1,\, \eta_0,\, \eta_1) = \frac{1}{2} \int_0^T \|\mu_0\|^2 + \|\mu_1\|^2 + \|\eta_0\|^2 + \|\eta_1\|^2 \, dt < 1
$$

and $T = 1$.

In terms of (4.1) and (4.2), $E_0 = E_1 = I$, $M_0 = M_1 = N_0 = N_1 = I$, $P_0 = P_1 = 0$, $C_0 = C_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}$, and

$$
\begin{aligned}
F_0 &= -\begin{bmatrix} 0 & 1 \\ -9 & 0 \end{bmatrix}, \; F_1 = -\begin{bmatrix} 0 & 1 \\ -4 & 0 \end{bmatrix} \\
B_0 &= B_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.
\end{aligned}
$$

The reduction technique in Section 4.1.2 is implemented with the same software algorithm as in the DAE case. The minimal proper u is shown in Fig. 4.1.

One way to check if u in Fig. 4.1 is minimal proper is to solve (4.27). If u is minimal proper and $\alpha > 1$, then we should observe $J > 0$. If $0 < \alpha < 1$, then we should observe $J = 0$. Several values of α were tested and the minimal values of J are tabulated in Table 4.1. As expected, J is nonzero only when $\alpha > 1$ and J is increasing with α.

Table 4.1: Minimal Proper u check for Example 1.

α	0.6	0.9	0.99	1	1.01	1.1	1.8
J	0	0	0	*	0.0003	0.039	2.5564

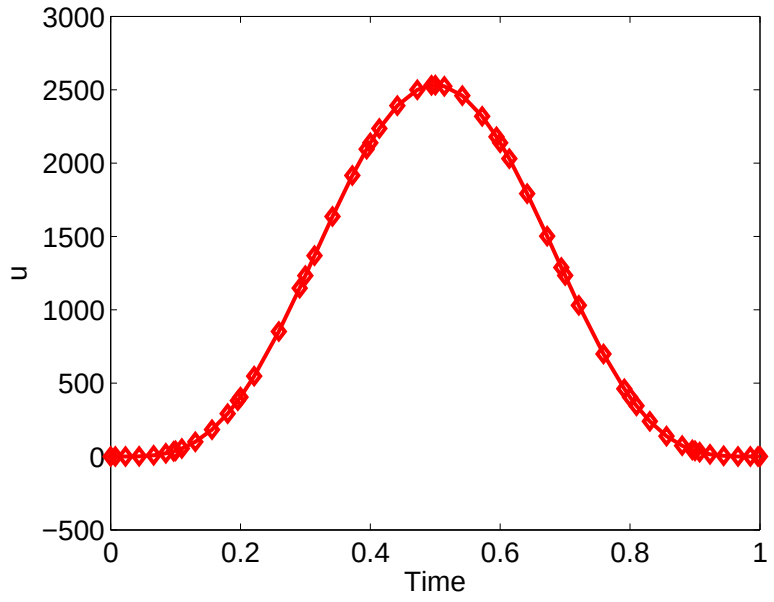


Figure 4.1: Minimal proper u for the L^2 bound–Example 1.

Example 4.4.2. *This example tests the techniques of this paper on DAE models, one of which is a high index DAE. This is an example not covered by previous results in the literature. Let the normal and faulty models take the form of* (4.1) *where the fault under consideration occurs in* E_i *and* F_i

$$E_0 = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 0 & 2 & 3 & 0 & 5 & 0 \\ 0 & 0 & 3 & 4 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad E_1 = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 0 & -2 & 3 & 0 & 5 & 0 \\ 0 & 0 & 3 & 4 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$F_0 = \begin{bmatrix} 0 & 0 & 0 & -1 & 1 & -1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -4 & 0 & 0 \\ 1 & 0 & 0 & 0 & -2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad F_1 = \begin{bmatrix} 0.5 & 1 & 1.5 & -1 & 1 & -1 \\ 0 & -1 & 2.5 & 1 & 2.5 & 0 \\ 0 & -1 & 1.5 & 2 & 0 & 3 \\ -1 & 1 & 0 & -4 & 0 & 0 \\ 1 & 0 & 0 & 0 & -2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The rest of the parameters are

$$B_0 = B_1 = \begin{bmatrix} 1 & 1 & -1 & 0 & 0 & 0 \end{bmatrix}^T, \quad D_0 = D_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$C_0 = C_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$P_{0_0} = \begin{bmatrix} 0.1116 & 0.1522 & 0.1946 & -0.0449 & -0.1777 & -0.0674 \\ 0.1522 & 0.3166 & 0.4049 & -0.0934 & 0.0304 & -0.1401 \\ 0.1946 & 0.4049 & 0.6652 & 0.0773 & 0.0389 & 0.1159 \\ -0.0449 & -0.0934 & 0.0773 & 0.2899 & -0.0090 & 0.4348 \\ -0.1777 & 0.0304 & 0.0389 & -0.0090 & 0.9645 & -0.0135 \\ -0.0674 & -0.1401 & 0.1159 & 0.4348 & -0.0135 & 0.6522 \end{bmatrix}$$

$$P_{0_1} = \begin{bmatrix} 0.0813 & 0.1788 & 0.1870 & -0.0432 & -0.0407 & -0.0647 \\ 0.1788 & 0.5025 & 0.2720 & -0.0628 & -0.3622 & -0.0942 \\ 0.1870 & 0.2720 & 0.7563 & 0.0562 & 0.2550 & 0.0844 \\ -0.0432 & -0.0628 & 0.0562 & 0.2947 & -0.0589 & 0.4421 \\ -0.0407 & -0.3622 & 0.2550 & -0.0589 & 0.7021 & -0.0883 \\ -0.0647 & -0.0942 & 0.0844 & 0.4421 & -0.0883 & 0.66311 \end{bmatrix}$$

and $M_0 = M_1 = I_{6\times 6}$. $N_0 = N_1 = I_{3\times 3}$. *Hence, model 0 is index 3 and model 1 is index 1. The noise for model i is given by* (4.2) *and we assume the total noise in the* L^2 *measure is bounded by* $\gamma = 1$. *With the choices for* P_{0_0} *and* P_{0_1}, $\hat{P}_0 = \hat{P}_1 = I$ *are selected to satisfy* (4.11). *Then,* $P_0 = I$ *in* (4.16).

The software package chosen for implementation is GPOPS-II. Figure 4.2 shows the minimal proper u for this example.

From Section 4.1.5, we can check if this signal is minimal proper. The results are tabulated in Table 4.2 for various values of α. Recall if $\alpha < 1 (> 1)$ then we expect $J = 0 (> 0)$ if u is minimal proper. The results in the table provide good evidence that u in Fig. 4.2 is indeed minimal proper.

The identification test in Section 4.2 requires the use of the Γ_∞ noise bound. Recall $\tilde{u} = \sqrt{2}u$, where u is given in Figure 4.2, is proper for $\Gamma_\infty < \gamma$. Table 4.3 verifies this and shows it is very close to minimal proper. There is no utility in applying Algorithm 4.1 to this example.

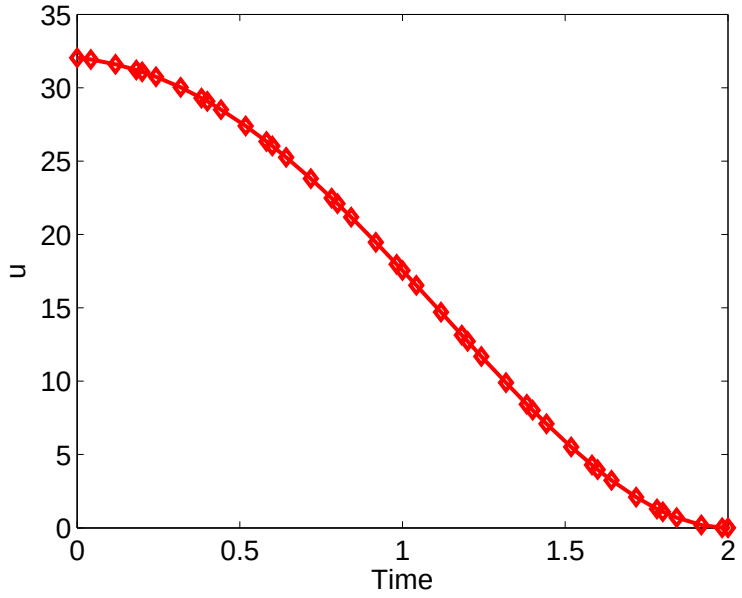


Figure 4.2: Minimal proper u for the L^2 bound–Example 2.

Table 4.2: Minimal Proper u check for Example 2 using Γ_{L^2}.

α	0.6	0.9	0.99	1	1.01	1.1	1.8
J	0	0	0	*	.0006	0.0553	3.6287

Table 4.3: Minimal Proper u check for Example 2 using Γ_{∞}.

α	0.6	0.9	0.99	1	1.01	1.1	1.8
J	0	0	0	*	0.0013	0.1132	7.2861

4.4.2 Model Identification

In practice, y comes from real-time sensor data. In lieu of an actual test system, we simulate model 1 in Example 4.4.2 and generate the outputs y. This requires the numerical solution of the index one DAE

$$E_1 x_1' + F_1 x_1 = B_1 \sqrt{2} u + M_1 \mu_1.$$

We calculate the completion (see [14, 57])

$$x_1' = \hat{A} x_1 + \hat{B} \sqrt{2} \bar{u} + \hat{M} \bar{\mu}, \tag{4.44}$$

where $\hat{A}$, $\hat{B}$, $\hat{M}$ are numerically generated and for a vector q, $\bar{q}$ is a vector containing q and the k derivatives of q with respect to t. k is the index of the DAE.

The last three zero rows of E_1 impose conditions on the solution of the DAE. Let q_{i_j} denote the j^{th} component of a vector q for model i. A consistent solution satisfies

$$x_{1_2} - x_{1_4} = \mu_{1_4} + \mu_{1_6} \tag{4.45a}$$

$$x_{1_2} = \frac{\mu_{1_6} - \mu_{1_5}}{2} \tag{4.45b}$$

$$x_{1_1} = \mu_{1_6}. \tag{4.45c}$$

μ_1, η_1, $x_1(0)$ are random noises, but they must satisfy (4.45) and $\Gamma_1^2(x_1(0), \eta_1, \mu_1) < \gamma^2$. We employ a sine expansion for μ_1 and η_1. Let

$$\mu_{1_i} = \sum_{n=1}^{N} a_{n_i} \sin\left(\frac{2\pi n t}{T}\right)$$
$$\eta_{1_i} = \sum_{n=1}^{N} b_{n_i} \sin\left(\frac{2\pi n t}{T}\right).$$

Then, $\mu_{1_i}(0)$, $\eta_{1_i}(0) = 0$ so a consistent initial condition must have $x_{1_2} = x_{1_4}$ and $x_{1_1} = x_{1_5} = 0$. The free initial conditions, x_{1_3}, x_{1_6}, x_{1_2}, and the coefficients, a_{n_i}, b_{n_i}, are randomly selected over a uniform distribution on $[-1,\ 1]$ and then scaled randomly so that $\Gamma_1^2(x_1(0), \eta_1, \mu_1) < \gamma^2$. The sine expansions make it easy to calculate the noise measures. For example,

$$\|\mu_{1_i}\|^2 = \frac{T}{2} \sum_{n=1}^{N} |a_{n_i}|^2.$$

Model 1 is index 1 so we also need the first derivative of μ_1 to construct $\bar{\mu}$.

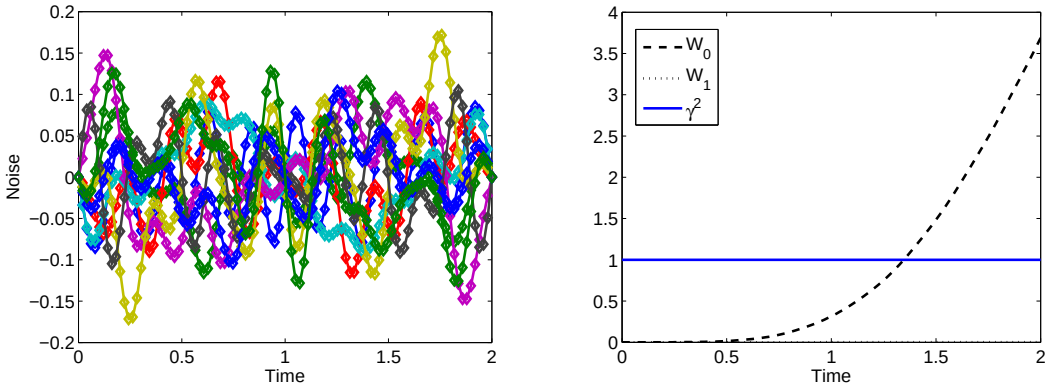
$$\frac{d\mu_{1_i}}{dt} = a_{n_i} \frac{2\pi n}{T} \cos\left(\frac{2\pi n t}{T}\right)$$
$$\frac{d\eta_{1_i}}{dt} = b_{n_i} \frac{2\pi n}{T} \cos\left(\frac{2\pi n t}{T}\right).$$

We integrate (4.44) and use the result to generate the output

$$y = C_1 x_1 + D_1 \sqrt{2} u + N_1 \eta_1.$$

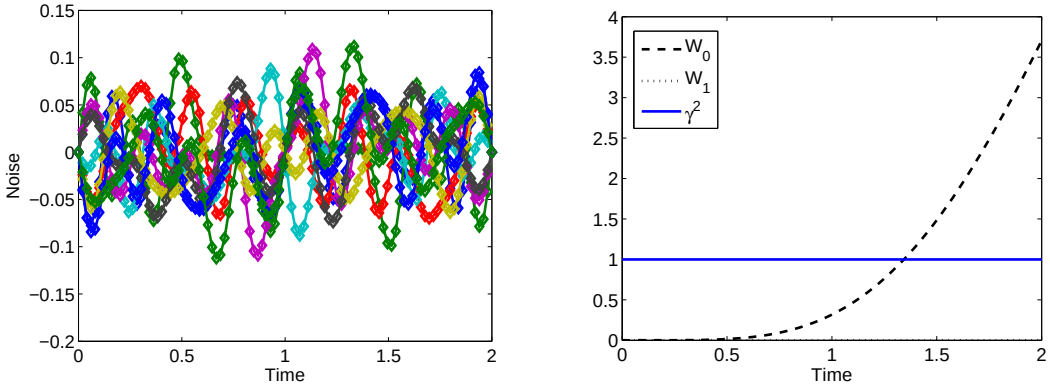
With this simulated output we can follow Section 4.2 for models 0 and 1 and find $W_i(t)$. If $W_i(t) > \gamma^2$ at any time, then model i is not realizable.

Figures 4.3 and 4.4 show two independent runs of the simulation. Part (A) in the figures



(a) (b)

Figure 4.3: (Run 1) (a) Random noises, μ_1, η_1 (b) $W_i(t)$ and γ^2.



(a) (b)

Figure 4.4: (Run 2) (a) Random noises, μ_1, η_1 (b) $W_i(t)$ and γ^2.

plots the various components of the random noises, μ_1 and η_1. Part (B) depicts $W_i(t)$ for each model and the bound on the noise γ. y is consistent with model one because it was simulated using model one. Both runs support this because $W_1(t)$ does not exceed γ. Since u is proper, this should mean that $W_0(t)$ exceeds the bound by some time $0 \leq t \leq T$. Indeed, in both runs we can correctly conclude that model 0 is not a realizable model because the estimated noise exceeds the bound. In fact, although the test interval is [0 2], model 0 can be rejected before $t = 1.4$ illustrating the capability of early detection. Recall, the initial conditions are not totally random as $x_{1_1} = x_{1_5} = 0$ and $x_{1_2} = x_{1_4}$ for consistency of the initial condition to hold. The randomly selected initial conditions are given below.

	Run 1	Run 2
$x(0)$	$\begin{bmatrix} 0 \\ -0.01 \\ 0.02 \\ -0.01 \\ 0 \\ -0.045 \end{bmatrix}$	$\begin{bmatrix} 0 \\ -0.0005 \\ 0.00025 \\ -0.005 \\ 0 \\ 0.00125 \end{bmatrix}$

4.5 Computational Study

We consider here a nonlinear model for a robot arm with elastic joints. Robot systems use harmonic drives, belts, or long shafts as transmission elements between the motors and the links (arms) and typically display oscillations both in fast motion and after sudden stop. Experimental tests and simulations have shown that the elasticity introduced at the joints by these transmission elements is the major reason for their vibrational behavior, so we must include elasticity in the dynamic model. As a result, the internal position of the motors does not determine the position of the driven arms. The dynamics of this consequential displacement can be modeled by inserting a linear torsional spring at each elastic joint, between the actuator and the link. The modeling process is described in full in [28].

The model for the robot arm shown in Figure 4.5 is given by (4.46). m_0, m_1, and m_p are masses, with m_p denoting the load or object being held, l_1 and l_2 are the lengths of the arms, K is the coefficient of elasticity of joint 2, NT is the transmission ratio at the second joint, JR_i are rotor inertias, q_i are angular coordinates describing the robot's configuration, and τ_i are the rotational torques caused by the drive motors. $x = [q_1, q_2, q_3, q_1', q_2', q_3']$, $u_1 = \tau_1$, and $u_2 = \tau_2$.

$$x_1' = x_4 \tag{4.46a}$$

$$x_2' = x_5 \tag{4.46b}$$

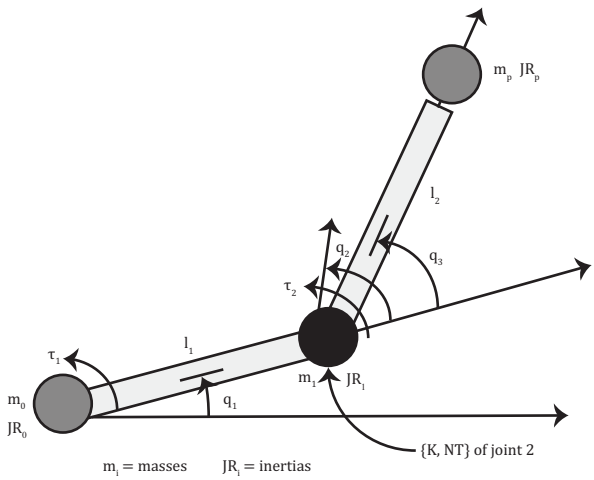


Figure 4.5: A two-link robot arm with the second joint elastic, from [27].

$$
\begin{aligned}
x_3' &= x_6 && \text{(4.46c)}\\
x_4' &= f_4(x_2,x_3,x_4,x_6) + g_{41}(x_3)u_1 - g_{41}(x_3)u_2 && \text{(4.46d)}\\
x_5' &= f_5(x_2,x_3,x_4,x_6) - g_{41}(x_3)u_1 + g_{52}(x_3)u_2 && \text{(4.46e)}\\
x_6' &= f_6(x_2,x_3,x_4,x_6) + g_{61}(x_3)u_1 - g_{61}(x_3)u_2 && \text{(4.46f)}\\
0 &= \cos x_1 + \cos(x_1+x_3) - 1 && \text{(4.46g)}
\end{aligned}
$$

Equation (4.46g) is a path constraint describing vertical motion on the endpoint of the robotic arm. The nonlinear functions are

$$
\begin{aligned}
g_{41}(x_3) &= \frac{A_2}{A_3(A_4 - A_3\cos^2 x_3)}\\
g_{52}(x_3) &= g_{41}(x_3) + \frac{1}{JR_1}\\
g_{61}(x_3) &= -g_{41}(x_3) - \frac{\cos x_3}{A_4 - A_3\cos^2 x_3}\\
f_4(x_2,x_3,x_4,x_6) &= \frac{A_2\sin x_3(x_4+x_6)^2 + A_3 x_4^2 \sin x_3\cos x_3}{A_4 - A_3\cos^2 x_3} + \frac{K\left(x_3 - \frac{x_2}{NT}\right)\left(\frac{A_2}{A_3}\left(\frac{NT-1}{NT}\right) + \cos x_3\right)}{A_4 - A_3\cos^2 x_3}\\
f_5(x_2,x_3,x_4,x_6) &= -f_4(x_2,x_3,x_4,x_6) + \frac{K}{NT}\left(x_3 - \frac{x_2}{NT}\right)\left(\frac{1}{JR_1} - 2g_{41}(x_3)\right)\\
f_6(x_2,x_3,x_4,x_6) &= -f_4(x_2,x_3,x_4,x_6) - \frac{K\left(x_3 - \frac{x_2}{NT}\right)\left(\frac{A_5}{A_3} - \left(\frac{3NT+1}{NT}\right)\cos x_3\right)}{A_4 - A_3\cos^2 x_3}
\end{aligned}
$$

$$- \frac{A_5 x_4^2 \sin x_3 + A_3 \sin x_3 \cos x_3 (x_4 + x_6)^2}{A_4 - A_3 \cos^2 x_3}$$

and the constants are

$$\begin{aligned} A_2 &= JR_p + m_p l_2^2 \\ A_3 &= m_p l_1 l_2 \\ A_4 &= (m_1 + m_p) l_1 l_2 \\ A_5 &= (m_1 + m_p) l_1^2. \end{aligned}$$

This model is nonlinear and cannot be directly used in our algorithm. Fault detection in nonlinear ODE models was studied in [68]. In the ODE case, the author shows auxiliary signals can sometimes be found for the nonlinear problem by linearizing the nonlinear models and finding a minimal proper signal for the linearized models. The work in [68] shows if u is strictly proper for the linearized system and β is a small enough, positive scalar, then βu is proper for the nonlinear system with the noise bounds scaled by β^2. Furthermore, if the nonlinearities satisfy certain technical assumptions, then u is proper for the nonlinear problem, except for possibly tighter noise bounds.

Although specific to ODE models, [68] demonstrates potential for an extension to nonlinear DAE models, although there are some subtleties. In general, one would expect a linearized DAE to provide local information on the nonlinear DAE it originates from. However, linearized DAEs can feature quite different phenomena. A linearized DAE may show lower or higher index than the original DAE, suggest incorrect stability, observability, etc., and fail to remain regular [49]. On the other hand, if one linearizes around a constant solution and if some technical assumptions hold, then the vector field determined by the linear time invariant linearization approximately agrees with the vector field of actual solutions in a neighborhood around the equilibrium point [15]. The technical assumptions consist of some constant rank assumptions during a "reduction process" involving repeated differentiations and the assumed invertibility of $F_x(0, \bar{x})$ where $F(x', x) = 0$ is the nonlinear DAE and $\bar{x}$ is the equilibrium point. These assumptions frequently hold in mechanics applications such as the robot arm under consideration. In the following, we linearize the nonlinear DAE models and find a minimal proper signal for the linearized models. The signal is then injected into the nonlinear models and the minimal noise required to satisfy equivalent outputs and the nonlinear DAEs is calculated.

4.5.1 Linearization

Recall, one assumption in our reduction procedure is that $\tilde{D}$ has full column rank. After a linearization, it can be shown that the model in (4.46) cannot meet this condition for reasonable

faults. We differentiate (4.46g) twice to find another constraint that involves the algebraic variable u_1. We also can eliminate (4.46f) with the added constraint in hand, so that the algebraic variables are now x_6, u_1. The reformulated, but equivalent, model given in (4.47) will now yield a linearization where $\tilde{D}$ is full column rank. This nonlinear DAE model has index 2. We drop the (x_i) dependencies of the functions $f_4(x_2, x_3, x_4, x_6)$, $f_5(x_2, x_3, x_4, x_6)$, and $g_{41}(x_3)$ to improve readability.

$$x_1' = x_4 \tag{4.47a}$$

$$x_2' = x_5 \tag{4.47b}$$

$$x_3' = x_6 \tag{4.47c}$$

$$x_4' = f_4 + g_{41}u_1 - g_{41}u_2 \tag{4.47d}$$

$$x_5' = f_5 - g_{41}u_1 + g_{52}u_2 \tag{4.47e}$$

$$0 = \cos x_1 + \cos(x_1 + x_3) - 1 \tag{4.47f}$$

$$\begin{aligned} 0 = &- \sin x_1(f_4 + g_{41}u_1 - g_{41}u_2) - x_4^2 \cos x_1 + \sin(x_1 + x_3) \\ &\left(\frac{K(x_3 - \frac{x_2}{NT})(\frac{A_5}{A_3} - \frac{3NT+1}{NT}\cos x_3)}{A_4 - A_3\cos^2 x_3} + \frac{A_5 x_4^2 \sin x_3 + A_3 \sin x_3 \cos x_3 (x_4 + x_6)^2}{A_4 - A_3\cos^2 x_3} + \right. \\ &\left. \frac{\cos x_3}{A_4 - A_3\cos^2 x_3}u_1 - \frac{\cos x_3}{A_4 - A_3\cos^2 x_3}u_2 \right) - (x_4 + x_6)^2 \cos(x_1 + x_3). \end{aligned} \tag{4.47g}$$

We observe that (4.47) can be rewritten

$$\begin{aligned} \bar{x}' &= F(\bar{x}, w, u) \\ 0 &= G(\bar{x}, w, u) \end{aligned}$$

where $\bar{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}^T$ are the dynamic variables, $w = \begin{bmatrix} x_6 & u_1 \end{bmatrix}^T$ are the algebraic variables, and $u = u_2$ is the auxiliary signal. The formulae for F and G are clear from (4.47).

The set points used in the linearized models correspond to the arm holding a mass motionless. Calculation of a set point will determine part, but not all, of u. Let u_1 be the free component. Thus, it is an algebraic variable in terms of the DAE, while u_2 is the test signal. The set points for the nominal and faulty models are found via a nonlinear solve in Matlab. The nominal model set point is found using a relatively large value for m_p to simulate an object being held. We will analyze two faults and refer to these faulty models as model 1 and model 2. Since our approach requires only two models at a time—one nominal and the other faulty—we construct two auxiliary signals, one for each faulty model paired with the nominal model. The first fault, model 1, simulates unintentionally dropping the object or realizing that the robotic arm is holding the wrong object by decreasing m_p in the faulty model. The second fault, model

2, simulates a breakdown of the second joint by lowering the elasticity parameter, K. Model 0 is the nominal model. The set points are given in Table 4.4 and values of all parameters are given in Table 4.5.

Table 4.4: Numerically computed set points for all models.

	Model 0 ($m_p = 10$, $K = 4$)	Model 1 ($m_p = 1$)	Model 2 ($K = 1$)
x_1	.57	.47	.64
x_2	1.70	1.97	1.46
x_3	.85	0.99	0.73
x_4	0	0	0
x_5	0	0	0
x_6	0	0	0
u_1	0	0	0
u_2	0	0	0

Table 4.5: Nominal parameter values for the robot arm.

Parameter	Description	Nominal Value
K	Coefficient of elasticity at joint 2	4
NT	Transmission ratio of joint 2	2
JR_0	Rotor inertia of joint 1	1
JR_1	Rotor inertia of joint 2	1
JR_p	Inertia at endpoint	1
l_1	Length of link 1	1
l_2	Length of link 2	1
m_1	Mass of joint 2	1
m_p	Mass at endpoint	10

The linearization for model i is

$$E_i z_i' + F_i z_i = B_i u + M_i \mu_i$$
$$y_i = C_i z_i + N_i \eta_i$$

where $z_i = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & u_1 \end{bmatrix}^T - \tilde{x}_i$ is the state vector consisting of dynamic and algebraic variables. $\tilde{x}_i$ is a numerically computed set point from Table 4.4 and $u = u_2$ is the

control. μ_i and η_i are additive noises. The linearized models for $i = 0, 1$, given below, may be calculated in Maple or Matlab.

$$E_0 = E_1 = \begin{bmatrix} I_{5\times 5} & 0 \\ 0 & 0 \end{bmatrix}$$

$$-F_0 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -0.3654 & 0.7309 & 0 & 0 & 0 & 0.1660 \\ 0 & -0.3026 & 0.6053 & 0 & 0 & 0 & -0.1660 \\ -1.5239 & 0 & -0.9878 & 0 & 0 & 0 & 0 \\ 0 & 0.5578 & -1.1156 & 0 & 0 & 0 & 0.0096 \end{bmatrix}$$

$$-F_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1.8304 & 3.6609 & 0 & 0 & 0 & 1.1796 \\ 0 & 3.1896 & -6.3791 & 0 & 0 & 0 & -1.1796 \\ -1.4504 & 0 & -0.9939 & 0 & 0 & 0 & 0 \\ 0 & 0.7549 & -1.5098 & 0 & 0 & 0 & -0.2150 \end{bmatrix}$$

$$M_0 = N_0 = I, \quad B_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -0.1660 \\ 1.1660 \\ 0 \\ -0.0096 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1.1796 \\ 2.1796 \\ 0 \\ 0.2150 \end{bmatrix}$$

$$C_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad C_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

The optimal auxiliary signals found using the algorithm from Section 4.1 with a noise bound of $\gamma = 0.25$ are given in Figure 4.6. Their similarity is a symptom of the effects the faults have on the system. Changes in mass or elasticity will both result in an unexpected vertical displacement of the endpoint. Applying a burst of torque, which is essentially what the system is forced to do given a short interval like [0 1], will expose the problem in either case. However, it is interesting to observe the minimal proper signals over a longer interval, [0 10], shown in Figure 4.7. Here, we see a definitive discrepancy in the signals because on a longer interval the software can

consider smaller signals that exploit more subtle differences such as resonant frequencies.

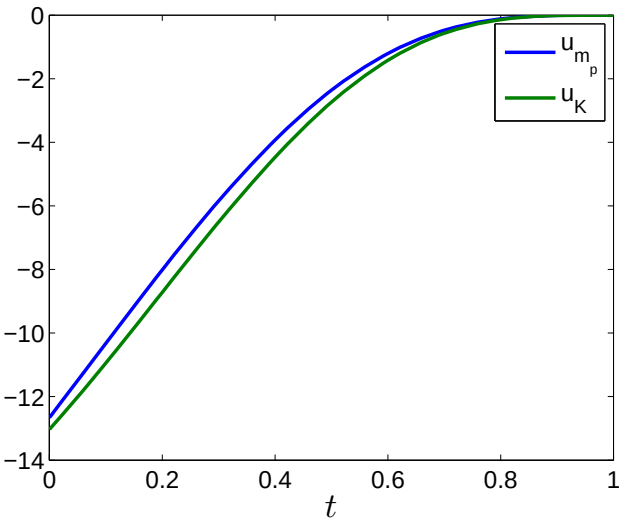


Figure 4.6: Comparison of numerically computed minimal proper signals u_{m_p}, the signal for models 0 and 1, and u_K, the signal for models 0 and 2, on [0 1].

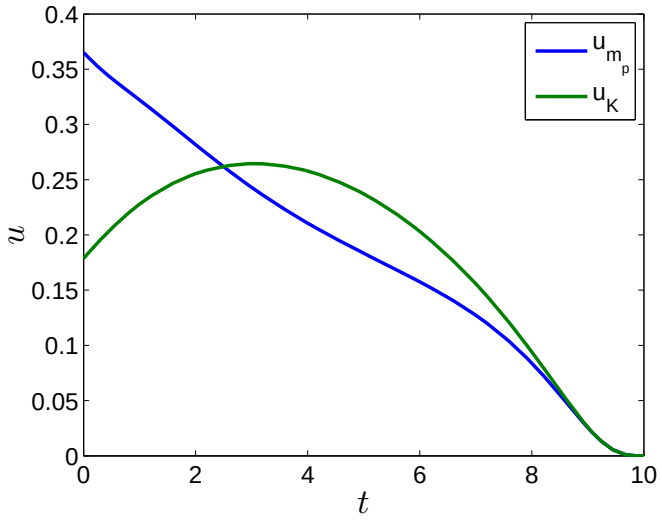


Figure 4.7: Comparison of numerically computed, minimal proper signals u_{m_p} and u_K on [0 10].

4.5.2 Evaluation of the Test Signal

Given a proper test signal u for the linearized problem, we want to examine its efficacy on the nonlinear problem. In [68], experimentation showed that a signal found for the linear problem with a noise bound of one is usually proper for the nonlinear problem, except the noise bound may have to be scaled from one. To test the signal, the author fixed u to be the minimal, proper signal for the linear problem and found the smallest noise necessary to satisfy the nonlinear constraints and equal outputs. The scaled noise bounds for the nonlinear test problems ranged from 0.708 to 1.117. Note, if the noise bound is found to be greater than one, then u found for the linear problem is still proper for the nonlinear problem with noise bound one, just not minimal proper.

The analogous problem here is

$$\min_{\eta_0,\eta_1,\mu_0,\mu_1,w_0,w_1} \delta \tag{4.48a}$$

$$\bar{x}_0' = F_0(\bar{x}_0, w_0, u) + \bar{\mu}_0 \tag{4.48b}$$

$$\bar{x}_1' = F_1(\bar{x}_1, w_1, u) + \bar{\mu}_1 \tag{4.48c}$$

$$\omega' = \eta_0^T\eta_0 + \bar{\mu}_0^T\bar{\mu}_0 + \tilde{\mu}_0^T\tilde{\mu}_0 + \eta_1^T\eta_1 + \bar{\mu}_1^T\bar{\mu}_1 + \tilde{\mu}_1^T\tilde{\mu}_1, \;\; \omega(0) = 0 \tag{4.48d}$$

$$0 = G_0(\bar{x}_0, w_0, u) + \tilde{\mu}_0 \tag{4.48e}$$

$$0 = G_1(\bar{x}_1, w_1, u) + \tilde{\mu}_1 \tag{4.48f}$$

$$0 = C_0[\bar{x}_0 \;\; w_0] + \eta_0 - (C_1[\bar{x}_1 \;\; w_1] + \eta_1) \tag{4.48g}$$

$$\delta \geq \frac{1}{2}(\bar{x}_0^T(0) - \tilde{x}_0)\hat{P}_0(\bar{x}_0(0) - \tilde{x}_0) + \frac{1}{2}(\bar{x}_1^T(0) - \tilde{x}_1)\hat{P}_1(\bar{x}_1(0) - \tilde{x}_1) + \omega(T), \tag{4.48h}$$

where $\tilde{x}_i$ is the set point for model i and $\mu_i = [\bar{\mu}_i \;\; \tilde{\mu}_i]$. The subscript i denotes which model, $i = 0$ (nominal) or $i = 1$ (faulty), is being considered. We have also simplified the equations by ignoring the coefficient matrices on the noises since they are identity matrices in our example. u is the signal from Figure 4.6. We can solve for some of the noise using equations (4.48e)–(4.48g) and rewrite the problem

$$\min_{\eta_1,\bar{\mu}_0,\bar{\mu}_1,w_0,w_1} \delta \tag{4.49a}$$

$$\bar{x}_0' = F_0(\bar{x}_0, w_0, u) + \bar{\mu}_0 \tag{4.49b}$$

$$\bar{x}_1' = F_1(\bar{x}_1, w_1, u) + \bar{\mu}_1 \tag{4.49c}$$

$$\begin{aligned}\omega' = {} & (C_1[\bar{x}_1 \;\; w_1] + \eta_1 - C_0[\bar{x}_0 \;\; w_0])^T (C_1[\bar{x}_1 \;\; w_1] + \eta_1 - C_0[\bar{x}_0 \;\; w_0]) + \bar{\mu}_0^T\bar{\mu}_0 \\ & + G_0^TG_0 + \eta_1^T\eta_1 + \bar{\mu}_1^T\bar{\mu}_1 + G_1^TG_1, \;\; \omega(0) = 0\end{aligned} \tag{4.49d}$$

$$\delta \geq \frac{1}{2}(\bar{x}_0^T(0) - \tilde{x}_0^T)\hat{P}_0(\bar{x}_0(0) - \tilde{x}_0) + \frac{1}{2}(\bar{x}_1^T(0) - \tilde{x}_1^T)\hat{P}_1(\bar{x}_1(0) - \tilde{x}_1) + \omega(T). \tag{4.49e}$$

Let K_1 denote the calculated value of K for the pair of models 0 and 1 and K_2 denote the value of K for models 0 and 2. Our simulations estimated $K_1 = .0672$ and $K_2 = .0360$. Recall, $\gamma = 0.25$ and note $\sqrt{K_1} = 0.259$, indicating u_{m_p} is proper on the nonlinear problem for essentially the same noise bound as for the linearized problem. $\sqrt{K_2} = 0.19$, thus the linear test signal u_K found with a noise bound of $\gamma = 0.25$, would work well on the nonlinear problem, but the signal would provide perfect detection only with the smaller additive noise bound of 0.19.

This example shows the utility of our algorithm on practical problems with models and faults that come from real-world applications, even ones with nonlinear phenomena. Finding a proper u was successful by utilizing linearized models.

4.5.3 Model Identification with Nonlinear Models

To identify which model is active, the natural problem to solve is analogous to (4.28) and (4.29), except with nonlinear constraints:

$$W_i = \min_{\eta_i, \bar{\mu}_i, \tilde{\mu}_i} \left(\frac{1}{2} \left(\bar{x}_i^T(0) - \tilde{x}_i^T \right) \hat{P}_i \left(\bar{x}_i(0) - \tilde{x}_i \right) + \frac{1}{2} \int_0^T \|\eta_i\|^2 + \|\mu_i\|^2 \, dt \right) \tag{4.50a}$$

$$\bar{x}_i' = F_i(\bar{x}_i, w_i, u) + \bar{\mu}_i \tag{4.50b}$$

$$0 = G_i(\bar{x}_i, w_i, u) + \tilde{\mu}_i \tag{4.50c}$$

$$y = C_i \left[\bar{x}_i \;\; w_i \right] + \eta_i, \tag{4.50d}$$

where y typically comes from sensor readings. Recall, if $W_i > \gamma^2$, then model i is not realizable. Solving this problem for $i = 0, 1$ will successfully identify the active model, but will be computationally expensive and cannot be done in real time.

An interesting question is whether we can use the linearized models and the linear model identification approach from Section 4.2 on problems with nonlinear phenomena. The original models were assumed to be linear in that development. This example raises the natural question of whether we can use that theory if the original models are nonlinear.

There are many places where the answer will likely be no, at least for general nonlinear DAE models and without modifying the approach. As with any linearized model, its accuracy and utility will suffer as the states and controls move away from the operating point. Since auxiliary signals, by their very definition, force systems off of their operating points to expose faults, the linearized models will be less accurate as the test goes on. This means the noise estimation technique in Theorem 4.2.1 will estimate the model error of the linearized model as well as the noise required to generate the observed output. The result could be imperfect detection.

Moreover, as mentioned earlier, linearized DAEs do not always accurately depict the local behavior of their parent nonlinear DAEs. They may feature different index, stability, or observability characterizations or fail to remain regular. These factors could lead to estimations that

are too corrupted by model error to be useful.

However, there are more optimistic scenarios. For example, if the technical assumptions of [15] hold, then the vector field determined by the linear time invariant linearization approximately agrees with the vector field of actual solutions in a neighborhood around the equilibrium points. Applications with small noise bounds and proper signals may also benefit from this approach. Small noises and signals will keep the states from straying too far from the operating point, cutting the noise generated from modeling error. Clearly, an avenue for future work is to formalize instances where this approach is useful when the original models are nonlinear.

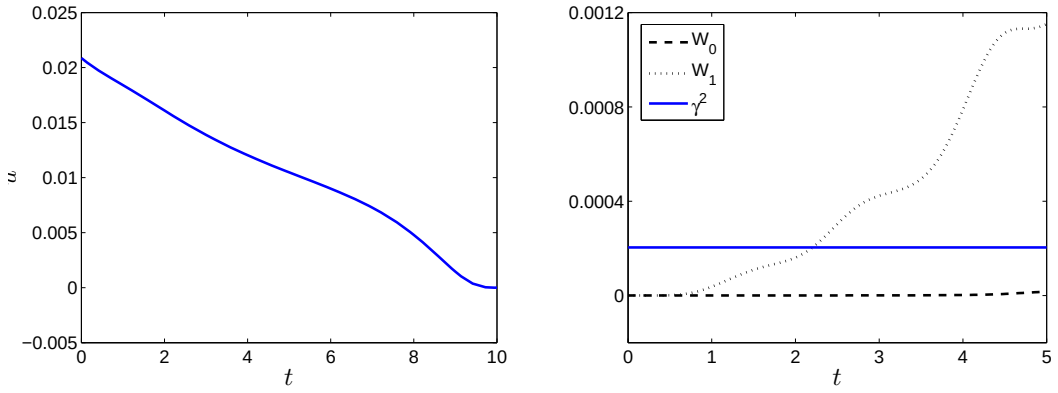
In practice, y comes from sensor data. In the absence of a physical experiment with measurements, we simulate a nonlinear model to obtain y. Due to the absence of damping and friction in the models, the proper signals computed in the previous section are not good candidates for simulation. In fact, most significant controls will drive the arm to move through kinematic singularities, for example, when $x_1 = x_3 = 0$. This is merely a simulation problem and not a problem of the approach because in practice there is no need to simulate the nonlinear model.

Smaller noises and smaller signals will mitigate this problem. Practically, they may also be more reasonable. The noise bound in the previous section, $\gamma = 1/4$, is a significant amount of noise in terms of the angles, which are in radians. The following figures result from the computation of a new proper signal for models 0 and 1 (see Table 4.4) with $\gamma = \frac{1}{70}$. Then, following Section 4.4.2, we generate noise that meets the bound and simulate the nominal model to find y. As y is computed we calculate the past noise cost required to generate y for each model.

The results are presented in Figure 4.8. The noise estimation correctly identifies model 0 as being the active model. Here, the system stays close to its operating point and the linearizations are good approximations of their nonlinear parents. The result is good detection. Even by experimenting with higher noise bounds and larger signals, we were unable to find an instance of failed detection, so long as the system was able to be simulated.

4.6 Model Uncertainty

Thus far, only additive uncertainty has been considered. In contrast to additive uncertainty, model uncertainty could include nonlinearities and other effects polluting the models. A number of extra technical difficulties are associated with this case. First, the output set can grow with increasing u since more input can create more uncertainty. Thus a larger multiple of a proper signal may not still be proper. Second, there are conditions that must be tested within the optimization interval. This section will discuss the extension of our approach to model uncertainty.



(a) (b)

Figure 4.8: (a) Proper auxiliary signal separating models 0 and 1 with $\gamma = \frac{1}{70}$. (b) $W_i(t)$ and γ^2.

We assume the two models ($i = 0, 1$) are of the form:

$$E_i x_i' = (-F_i + K_i \Delta_i G_i) x_i + (B_i + K_i \Delta_i H_i) u + M_i \mu_i \tag{4.51a}$$

$$y_i = (C_i + L_i \Delta_i G_i) x_i + (D_i + L_i \Delta_i H_i) u + N_i \eta_i. \tag{4.51b}$$

Here μ_i, η_i are the additive noises discussed in the previous sections and the matrices $L_i, K_i, G_i, H_i, \Delta_i$ represent multiplicative model uncertainties. Matrices L_i, K_i, G_i, H_i give the structure and location of the uncertainty. The uncertain matrix Δ_i is bounded by $\sigma(\Delta_i(t)) \leq 1$ where $\sigma(X)$ denotes the largest singular value of a matrix X. Scaling of the uncertain matrices permits bounds other than one. The initial and additive uncertainties are bounded by

$$\Gamma_i^2(x_i(0), \nu_i) = \frac{1}{2} x_i(0)^T P_{0_i} x_i(0) + \frac{1}{2} \int_0^s \|\mu_i\|^2 + \|\eta_i\|^2 \, dt < 1, \ \ \forall s \in [0, T]. \tag{4.52}$$

Again, scaling allows the consideration of bounds other than one. The extra condition $\forall s \in [0, T]$ introduced here is redundant, but will be used later. We follow the formulation in [1] which is given for the ODE case. We can consider the modified system:

$$E_i x_i' = -F_i x_i + B_i u + K_i \phi_i + M_i \mu_i \tag{4.53a}$$

$$y_i = C_i x_i + D_i u + L_i \phi_i + N_i \eta_i \tag{4.53b}$$

$$\xi_i = G_i x_i + H_i u, \tag{4.53c}$$

where $\phi_i = \Delta_i \xi_i$. From the model uncertainty bound $\sigma(\Delta_i) \leq 1$, we have that $\|\phi_i\| - \|\xi_i\| \leq 0$

over the interval $[0,\, T]$. This implies the more conservative condition

$$\int_0^s \|\phi_i\|^2 - \|\xi_i\|^2\, dt < 0,\ \forall s \in [0,\, T]. \tag{4.54}$$

Adding (4.52) and (4.54) the overall noise bound for each model is:

$$\begin{aligned}\Gamma_i^2(x_i(0), \nu_i, \phi_i, \xi_i) = \frac{1}{2} x_i(0)^T P_{0_i} x_i(0) + \frac{1}{2}\int_0^s \left(\|\mu_i\|^2 + \|\eta_i\|^2\right. \\ \left. + \|\phi_i\|^2 - \|\xi_i\|^2\right)\, dt < 1, \quad \forall s \in [0,\, T].\end{aligned} \tag{4.55}$$

This bound allows for more noise and introduces more conservatism in our solution. This is not conservative in the sense of missed faults. Instead, the auxiliary signal obtained this way, while proper, may not be optimal in terms of the original uncertainty. Unlike the additive uncertainty case, the noise measure is not positive definite in the model uncertainty case. Consequently, we must enforce (4.55) $\forall s \in [0,\, T]$ instead of just $s = T$.

The reduction techniques in Section 4.1.2 can now be employed. Recall, we perform the coordinate changes (4.8). Then (4.53a) can be written

$$x'_{i1} = A_{i11}x_{i1} + A_{i12}x_{i2} + B_{i1}u + K_{i1}\phi_i + M_{i11}\mu_{i1} + M_{012}\mu_{i2} \tag{4.56a}$$

$$0 = A_{i21}x_{i1} + A_{i22}x_{i2} + B_{i2}u + K_{i2}\phi_i + M_{i22}\mu_{i2} \tag{4.56b}$$

where A_{ijk} and M_{ijk} are given in (4.6) and (4.7). B_{ij} and C_{ij} are given in (4.10). The definitions of K_{ij} are analogous to B_{ij}. $y_0 - y_1 = 0$ can be expressed:

$$\begin{aligned}0 = {} & C_{01}x_{01} + C_{02}x_{02} + D_0 u + L_0\phi_0 + N_0\eta_0 \\ & - (C_{11}x_{11} + C_{12}x_{12} + D_1 u + L_1\phi_1 + N_1\eta_1).\end{aligned}$$

Now, letting $z_1 = \begin{bmatrix} x_{01} \\ x_{11} \end{bmatrix}$, $z_2 = \begin{bmatrix} x_{02} \\ x_{12} \end{bmatrix}$, $\phi = \begin{bmatrix} \phi_0 \\ \phi_1 \end{bmatrix}$, $\zeta_1 = \begin{bmatrix} \mu_{02} \\ \mu_{12} \\ \eta_0 \end{bmatrix}$, $\zeta_2 = \begin{bmatrix} \mu_{01} \\ \mu_{11} \\ \eta_1 \end{bmatrix}$, we have

$$z'_1 = \hat{A}z_1 + \begin{bmatrix} \hat{D} & \hat{N} \end{bmatrix} \begin{bmatrix} z_2 \\ \zeta_1 \end{bmatrix} + \hat{B}u + \hat{K}\phi + \hat{M}\zeta_2 \tag{4.57a}$$

$$0 = \tilde{A}z_1 + \begin{bmatrix} \tilde{D} & \tilde{N} \end{bmatrix} \begin{bmatrix} z_2 \\ \zeta_1 \end{bmatrix} + \tilde{B}u + \tilde{K}\phi + \tilde{M}\zeta_2 \tag{4.57b}$$

where most of the matrices are defined in (4.13). The additional matrices are

$$\hat{K} = \begin{bmatrix} K_{01} & 0 \\ 0 & K_{11} \end{bmatrix}, \; \tilde{K} = \begin{bmatrix} K_{02} & 0 \\ 0 & K_{12} \\ L_0 & -L_1 \end{bmatrix}.$$

As before, we can eliminate (4.57b) by solving for z_2 and part of ζ_1:

$$-\begin{bmatrix} z_2 \\ \zeta_{11} \end{bmatrix} = \begin{bmatrix} \tilde{D} & \tilde{N}_1 \end{bmatrix}^{-1} \left(\tilde{A} z_1 + \tilde{N}_2 \zeta_{12} + \tilde{B} u + \tilde{K} \phi + \tilde{M} \zeta_2 \right). \tag{4.58}$$

Letting $\nu = \begin{bmatrix} \phi^T & \zeta_{12}^T & \zeta_2^T \end{bmatrix}^T$, $\xi = \begin{bmatrix} \xi_0^T & \xi_1^T \end{bmatrix}^T$, and $\Theta = \begin{bmatrix} \tilde{D} & \tilde{N}_1 \end{bmatrix}^{-1}$ and substituting back into the dynamics and (4.53c) we get

$$z_1' = A z_1 + B u + M \nu \tag{4.59a}$$

$$\xi = G z_1 + H u + N \nu, \tag{4.59b}$$

where

$$A = \hat{A} - \hat{R}\Theta\tilde{A}, B = \hat{B} - \hat{R}\Theta\tilde{B}, K = \hat{K} - \hat{R}\Theta\tilde{K}$$
$$N = \hat{N} - \hat{R}\Theta\tilde{N}, M = \hat{M} - \hat{R}\Theta\tilde{M}, G = \begin{bmatrix} \tilde{G}_{01} & 0 \\ 0 & \tilde{G}_{11} \end{bmatrix} - \bar{R}\tilde{A}_u$$

$$H = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} - \bar{R}\tilde{B}_u, \bar{K} = -\bar{R}\tilde{K}_u, \bar{N} = -\bar{R}\tilde{N}_{2_u}$$
$$\bar{M} = -\bar{R}\tilde{M}_u, [\tilde{G}_{i1} \; \tilde{G}_{i2}] = V_i G_i, \bar{R} = \begin{bmatrix} \tilde{G}_{02} & 0 \\ 0 & \tilde{G}_{12} \end{bmatrix}$$
$$M = \begin{bmatrix} K & \bar{M}_1 & \bar{M}_1 \end{bmatrix}, N = \begin{bmatrix} \bar{K} & \bar{N}_1 & \bar{N}_2 \end{bmatrix}.$$

For convenience, in this section, we are redefining matrices such as A, B, M, N etc. that appeared in earlier sections. For a given matrix $\tilde{Q}$, $\tilde{Q}_u$ denotes the upper half of the matrix $\Theta\tilde{Q}$ that corresponds to the z_2 components.

When model uncertainty is involved, it is more straightforward to use the L_∞ noise measure. Hence, in this section, we define the total noise measure to be

$$\Gamma = \max_{i=0,1} \Gamma_i. \tag{4.60}$$

For computational purposes we rewrite this measure:

$$\begin{aligned}\Gamma &= \max_{\beta\in[0,\,1]} \beta\Gamma_0 + (1-\beta)\Gamma_1 \\ &= \max_{\beta\in[0,\,1]} \frac{1}{2}\left(x_0^T(0)\beta P_{0_0}x_0(0) + x_1^T(0)\beta P_{0_1}x_1(0)\right) + \frac{1}{2}\int_0^s \beta\left(\|\mu_0\|^2 + \right. \\ &\left. \|\eta_0\|^2 + \|\phi_0\|^2 - \|\xi_0\|^2\right) + (1-\beta)\left(\|\mu_1\|^2 + \|\eta_1\|^2 + \|\phi_1\|^2 - \|\xi_1\|^2\right)\, dt.\end{aligned}$$

P_{0_i} is bounded in (4.11). The initial uncertainty can be now be written concisely in terms of z_1 as

$$\frac{1}{2}z_1^T(0)P_\beta z_1(0),\ P_\beta = \begin{bmatrix}\beta\hat{P}_0 & 0\\ 0 & (1-\beta)\hat{P}_1\end{bmatrix}.$$

In terms of our new coordinates, the integrated portion of the overall noise is:

$$\frac{1}{2}\int_0^s \zeta_{11}^T J_{\beta_1}\zeta_{11} - \xi^T J_{\beta_2}\xi + \nu^T J_{\beta_3}\nu\ dt,$$

where $J_{\beta_2} = \text{diag}(\beta I, (1-\beta)I)$, $J_{\beta_3} = \text{diag}(J_{\beta_2}, \bar{J}_{\beta_1}, J_{\beta_4})$, and $J_{\beta_4} = \text{diag}(\beta I, (1-\beta I), (1-\beta)I)$. Here, $\text{diag}(J_{\beta_1},\ \bar{J}_{\beta_1})$ is a partition of $J_\beta = \text{diag}(\beta I, (1-\beta)I, \beta I)$ conformal with $[\zeta_{11}^T\ \zeta_{12}^T]^T$. The size of the identities in J_β, J_{β_2}, and J_{β_4} are conformal with the definitions of ζ_1, ξ, and ζ_2.

Recall, for a given matrix $\tilde{Q}$, that $\tilde{Q}_0$ denotes the lower half of the matrix $\Theta\tilde{Q}$ that corresponds to the ζ_{11} components. Then, in light of (4.58) and (4.59b) the noise measure can be expressed

$$\begin{aligned}\Gamma = \max_{\beta\in[0,\,1]} &\frac{1}{2}z_1^T(0)P_\beta z_1(0) + \frac{1}{2}\int_0^s z_1^T Q_1 z_1 + 2z_1^T Q_2 u + 2z_1^T Q_3\nu \\ &+ u^T Q_4 u + 2u^T Q_5\nu + \nu^T Q_6\nu\, dt\end{aligned} \tag{4.61}$$

where

$$\begin{aligned} Q_1 &= A^T J_{\beta_1}A - G^T J_{\beta_2}G, & Q_2 &= A^T J_{\beta_1}B - G^T J_{\beta_2}H \\ Q_3 &= A^T J_{\beta_1}M - G^T J_{\beta_2}N, & Q_4 &= B^T J_{\beta_1}B - H^T J_{\beta_2}H \\ Q_5 &= B^T J_{\beta_1}M - H^T J_{\beta_2}N, & Q_6 &= M^T J_{\beta_1}M - N^T J_{\beta_2}N + J_{\beta_3}. \end{aligned}$$

For a given u to be proper, we require the smallest noise possible to exceed the bound. The inner problem is to minimize (4.61) over all possible noise values, subject to (4.59a). Then the Hamiltonian for this inner problem can be defined by

$$H(z_1,\nu) = \frac{1}{2}\left(z_1^T Q_1 z_1 + 2z_1^T Q_2 u + 2z_1^T Q_3\nu + u^T Q_4 u + 2u^T Q_5\nu + \nu^T Q_6\nu\right)$$

$$+ \lambda^T(Az_1 + Bu + M\nu)$$

where λ is the Lagrange multiplier. The necessary conditions are

$$z_1' = Az_1 + Bu + M\nu \tag{4.62a}$$
$$-\lambda' = A^T\lambda + Q_1 z_1 + Q_2 u + Q_3\nu \tag{4.62b}$$
$$0 = M^T\lambda + Q_3^T z_1 + Q_5^T u + Q_6\nu \tag{4.62c}$$

with boundary conditions

$$P_\beta z_1(0) + \lambda(0) = 0 \tag{4.63a}$$
$$\lambda(s) = 0. \tag{4.63b}$$

Hence, the problem reformulation to find the minimal proper signal is

Problem 4.6.1.

$$\textit{Find } J = \inf_u \|u\|^2 \tag{4.64a}$$

such that

$$z_1' = Az_1 + Bu + M\nu \tag{4.64b}$$
$$-\lambda' = A^T\lambda + Q_1 z_1 + Q_2 u + Q_3\nu \tag{4.64c}$$
$$\psi' = \frac{1}{2}\left(z_1^T Q_1 z_1 + 2z_1^T Q_2 u + 2z_1^T Q_3\nu + u^T Q_4 u + 2u^T Q_5\nu + \nu^T Q_6\nu\right) \tag{4.64d}$$
$$0 = M^T\lambda + Q_3^T z_1 + Q_5^T u + Q_6\nu \tag{4.64e}$$

with the boundary conditions

$$P_\beta z_1(0) + \lambda(0) = 0 \tag{4.65a}$$
$$\psi(0) = \frac{1}{2} z_1^T(0) P_\beta z_1(0) \tag{4.65b}$$
$$\psi(s) \geq 1,\ \lambda(s) = 0 \tag{4.65c}$$
$$0 < \beta < 1,\ 0 \leq s \leq T. \tag{4.65d}$$

For additive noise we were able to set $s = T$; however, this is not the case when model uncertainty is included because the noise measure is not positive definite anymore. In this formulation, the end of the interval is essentially s because by time s, we know the cost has exceeded the bound.

4.7 Problems in High Index DAE

In the previous sections we developed an approach for active failure detection for linear DAE models using direct optimization. Supporting analysis and algorithms were provided. However, that development was based on the assumption that a certain optimization problem had a solution and gave a sufficient, but difficult to verify, mathematical criteria (see Theorem 4.1.1) for a solution to exist. In this section, we give a simple example to show that an optimal solution need not always exist if Theorem 4.1.1 is not satisfied. We then show how this can occur in a physical problem. That is, we replace the mathematical assumption with a more useful model characterization. Finally, if the original algorithm has difficulty, we show how to get a useful test signal by modifying the norm used in the outer minimization of Problem 4.1.2. Section 4.7.1 gives an example demonstrating how the previous algorithm can fail. In Section 4.7.2, we explain the source of the algorithmic failure. Section 4.7.3 explains how to modify the original algorithm to get a useful and nearly minimal test signal.

4.7.1 Example Where Previous Algorithm Fails to Converge

Recall, in the previous sections, we were able to reformulate Problem 4.1.1, and using industrial-grade optimal control software (e.g. GPOPS-II), compute proper test signals for a number of problems with high index DAE models. Now consider the following

Example 4.7.1. *Let $M_i = I$, $N_i = I$, and*

$$E_0 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, E_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, F_0 = F_1 = I, D_0 = D_1 = 0,$$

$$C_0 = C_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}, B_0 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, B_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Model 0 is an index two DAE and model 1 is an ODE. Assumptions 1 and 2 in Section 4.1.3 hold and we can reformulate the optimization problem using the procedure in Section 4.1.2. However, the algorithm is unable to converge to a solution for the minimal proper test signal. A typical test signal as the algorithm iterates is given in Figure 4.9. Further iterations provide similar graphs.

Closer examination of the analytic solution shows that the derivative of the test signal appears in the output. We can express y_0 and y_1,

$$\begin{aligned} y_0(t) &= -2u' - \mu_2' + 2u + \mu_1 + \mu_3 \\ y_1(t) &= e^{-t} z_1(0) + \mathcal{L}_1(e^t z_2(0) + \mathcal{L}_2(2u) + \nu_2)) \end{aligned} \tag{4.66a}$$

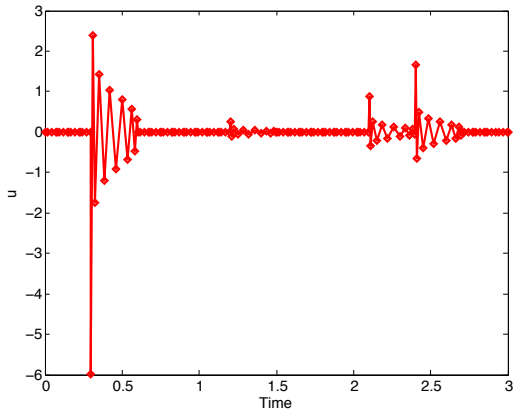


Figure 4.9: Computed test signal for Example 4.7.1 on iteration 20 using $\|u\|_2^2$.

$$+ \mathcal{L}_1(\nu_1 - \nu_2) + \nu_3, \tag{4.66b}$$

where

$$\mathcal{L}_1(f) = \int_0^t e^{-t+s} f(s)\, ds$$
$$\mathcal{L}_2(f) = \int_0^t e^{t-s} f(s)\, ds.$$

Test signals with very small L^2 norm can have very large derivatives. We illustrate how the presence of derivatives of the test signal in the output can lead to the lack of a minimum proper test signal in the idealized Example 4.7.2.

Example 4.7.2. *Suppose that the outputs for models 0 and 1 are*

$$y_0 = u' + \mu_0$$
$$y_1 = u + \mu_1,$$

where μ_i are L^2 functions and the noise bound is $\|\mu_0\|_2^2 + \|\mu_1\|_2^2 < 1$.

Let $b_n = \|\sin(n^2 t)\|_2^2$, $a_n = \frac{1}{n}$, and $u_n = \frac{a_n}{b_n}\sin(n^2 t)$. Then looking at $y_0 = y_1$ we see that as n increases we have u_n is proper for each n since it is not possible to have $y_0 = y_1$ while the noise bound holds, but $\lim_{n\to\infty} \|u_n\|_2 = 0$. Thus no minimum proper u exists.

This immediately presents several questions. One is how to determine if a derivative of u can appear in the output. In particular, we want a computational algorithm that can quickly test this. We focus on developing a computation that is easy to implement for small to medium sized problems. The second task is to determine whether the presence of derivatives of u in the

output causes the minimal proper test signal to not exist as a function or whether it is just one factor. The third question is how to compute a still useful proper test signal when it happens that the previous algorithm does not converge. We shall answer all three questions.

4.7.2 When is u' in the Output?

We start with the first problem by developing a computationally efficient test characterizing how derivatives of u enter the solution. We shall use derivative array theory [14, 46, 56]. Below, the index three case is developed. The more general case is then obvious. Putting aside concerns of smoothness and excluding noises from the calculation, suppose system (4.1) is differentiated three times. The result is

$$\overline{E}\overline{x}' + \overline{F}x = \overline{B}\overline{u}, \tag{4.67}$$

where

$$\overline{E} = \begin{bmatrix} E & 0 & 0 & 0 \\ F & E & 0 & 0 \\ 0 & F & E & 0 \\ 0 & 0 & F & E \end{bmatrix}, \overline{F} = \begin{bmatrix} F \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\overline{B} = \begin{bmatrix} B & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & B & 0 \\ 0 & 0 & 0 & B \end{bmatrix}, \overline{x} = \begin{bmatrix} x \\ x' \\ x'' \\ x''' \end{bmatrix}, \overline{u} = \begin{bmatrix} u \\ u' \\ u'' \\ u''' \end{bmatrix}.$$

Note that the if the index is three, then $\overline{E}$ is a 4×4 block matrix. In general $\overline{E}$ is taken $m \times m$ where m is at least one more than the index. The other matrices are sized accordingly. Let U be a maximal row rank left annihilator of $\overline{E}$. Then multiplying (4.67) by U gives

$$U\overline{F}x = U\overline{B}\overline{u} \tag{4.68}$$

and equation (4.68) describes all the constraints that hold on the solutions of the DAE. Thus

$$x = (U\overline{F})^{\dagger}U\overline{B}\overline{u} + \theta$$

where θ are solutions of the homogenous DAE. Thus we have proved the following

Proposition 4.7.1. *Derivatives of the input u do not appear in the output equation precisely when $C(U\overline{F})^{\dagger}U\overline{B}$ is zero except for the first r columns where r is the dimension of u.*

A left annihilator is easily computed using the singular value decomposition. Suppose that the

singular value decomposition of $\overline{E}$ is

$$\overline{E} = W \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} V^T.$$

Then a maximal rank left annihilator is $\begin{bmatrix} 0 & I \end{bmatrix} W^T$.

4.7.2.1 Previous High Index Example

Example 4.7.3. *Consider the high index problem in Example 4.4.2 in Section 4.4. The coefficient matrices are*

$$E_0 = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 0 & 2 & 3 & 0 & 5 & 0 \\ 0 & 0 & 3 & 4 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$E_1 = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 0 & -2 & 3 & 0 & 5 & 0 \\ 0 & 0 & 3 & 4 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$F_0 = \begin{bmatrix} 0 & 0 & 0 & -1 & 1 & -1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -4 & 0 & 0 \\ 1 & 0 & 0 & 0 & -2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$F_1 = \begin{bmatrix} 0.5 & 1 & 1.5 & -1 & 1 & -1 \\ 0 & -1 & 2.5 & 1 & 2.5 & 0 \\ 0 & -1 & 1.5 & 2 & 0 & 3 \\ -1 & 1 & 0 & -4 & 0 & 0 \\ 1 & 0 & 0 & 0 & -2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The rest of the parameters are

$$B_0 = B_1 = \begin{bmatrix} 1 & 1 & -1 & 0 & 0 & 0 \end{bmatrix}^T,$$

$$D_0 = D_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix},$$

$$C_0 = C_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Using the MATLAB "eig" command we see that model 1 has 3 nonzero eigenvalues and E_1 has rank 3 so that model 1 is index 1. On the other hand model 0 has only 1 nonzero eigenvalue and E_0 has rank 3 so that the problem is index 2 or index 3. Carrying out the calculation of Proposition 4.7.1 on model 0 we see that $C_0(U_0\overline{F_0})^\dagger U_0\overline{B_0} = 0$. Note that we only needed that all but the first column of $C_0(U_0\overline{F_0})^\dagger U_0\overline{B_0}$ was zero. Thus u' does not appear in the output of model 0. In the same way, we could also show u' does not appear in the output of model 1.

If there are several inputs, then it is up to the user which ones will be used as test signals. So, it is of interest to be able to characterize for which B there will not be any derivatives of the control appearing in the output. This can be done by looking at $C_0(U_0\overline{F_0})^\dagger U_0$. For this example where u' and u'' could possibly occur we look at the 7th through 18th columns of $C_0(U_0\overline{F_0})^\dagger U_0$ which are $C_0(U_0\overline{F_0})^\dagger U_0 = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ with Q_1 given by

$$\begin{bmatrix} 0.000 & -0.000 & 0 & -0.000 & 0.000 & -0.000 \\ -1.294 & 1.294 & 0 & 0.431 & 0.088 & 0.853 \\ -0.941 & 0.941 & 0 & 0.314 & -0.618 & 1.029 \end{bmatrix}$$

and

$$Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & -0.000 & 0.000 \\ 0 & 0 & 0 & 0 & 3.235 & -1.941 \\ 0 & 0 & 0 & 0 & 2.353 & -1.412 \end{bmatrix}.$$

Then u' and u'' will not appear in the output of model 0 precisely when

$$\begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} B_0 & 0 \\ 0 & B_0 \end{bmatrix} = 0.$$

4.7.2.2 New Example

Returning to Example 4.7.1 and letting $k = 2$ since it is an index two problem, we get that

$$(U_0\overline{F_0})^\dagger U_0\overline{B_0} = \begin{bmatrix} 2 & -2 & 0 \\ 2 & 0 & 0 \end{bmatrix}.$$

The first column does not matter. Thus we get that $(U_0\overline{F_0})^\dagger U_0\overline{B_0}$ will indicate that u' appears in the output if the first entry of $C_0 = [c_1, c_2]$ is nonzero since

$$C_0(U_0\overline{F_0})^\dagger U_0\overline{B_0} = \begin{bmatrix} 2c_1 + 2c_2 & -2c_1 & 0 \end{bmatrix}.$$

Hence, u' appears in the output signaling a potential failure of the auxiliary signal finding algorithm. And, as shown previously, the algorithm does fail for this example.

It might seem that not having derivatives of the input appearing in the outputs is very restrictive for higher index DAEs. We shall show later that this condition is not always necessary. However, this property is common among a number of physical systems. For example, constrained mechanics problems with position constraints take the form [10]

$$x_1' = f_1(x_1, x_2, x_3, u, t) \tag{4.69}$$
$$x_2' = f_2(x_1, x_2, t) \tag{4.70}$$
$$0 = f_3(x_2, t) \tag{4.71}$$
$$y = g(x_1, x_2, x_3, u, t) \tag{4.72}$$

which is index three if $\frac{\partial f_1}{\partial x_3}\frac{\partial f_2}{\partial x_1}\frac{\partial f_3}{x_2}$ is nonsingular. u' is absent since u does not appear in any of the algebraic constraints $f_3(x_2, t) = 0$ nor in f_2. Thus the approach of the previous sections can be immediately used on a wide variety of problems.

4.7.3 Modification of Original Algorithm

It would still be very useful to get a good test signal for problems where u', or a higher derivative of u, appears in the output and the original algorithm does not find a solution. We shall illustrate with an example where u' appears in the output and the original algorithm does not find a proper test signal, but a modification finds a nearly minimal test signal. A similar approach can be used if say u' and u'' appear in the output and the original algorithm does not converge.

The difficulty is that while u stays L^2 bounded, its derivative can be arbitrarily large and a minimum proper test signal does not necessarily exist in the usual sense. Accordingly, we modify the measure of the test signal by also including the size of its derivative. That is we

take the size of the test signal u as

$$\|u\|_\alpha^2 = \int_0^T \|u\|^2 + \alpha\|u'\|^2 dt$$

so that u and u' have to be L^2 bounded.

We add $u' = w$ to the optimization constraints, let $\alpha > 0$, and take the cost to be

$$\int_0^T \|u\|^2 + \alpha\|w\|^2 dt \tag{4.73}$$

Not only does this modification help us to find a good test signal but it also also sheds light on what is going wrong when $\alpha = 0$ for those problems where the previous algorithm did not converge. Figures 4.10–4.13 give the optimal test signal for several values of α for Example 4.7.1. If it exists, we denote the proper test signal which is minimal in the sense of (4.73) by u_α.

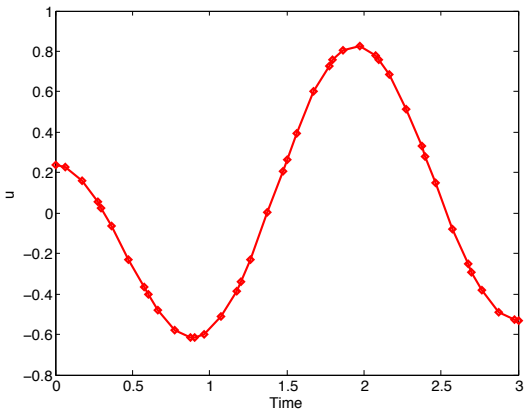


Figure 4.10: Minimal test signal for Example 4.7.1 using (4.73) with $\alpha = 0.1$.

Two things are immediately apparent. First, a useful minimal test signal is easily found using the new norm on u. The second is that as α is reduced the test signal is increasingly oscillating. The test signals are not growing larger in size but they also do not have a strong limit in L^2 as $\alpha \to 0^+$. They do appear to have a weak limit of zero in L^2. For the reader not familiar with weak limits the following example illustrates. Let $u_n(t) = \sin nt$ on $[-\pi,\ \pi]$ where n is a positive integer. For $n > 1$ all of the u_n have the same norm in $L^2[-\pi, \pi]$. However, for

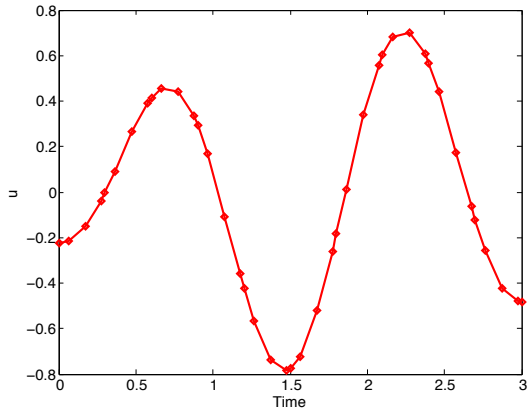


Figure 4.11: Minimal test signal for Example 4.7.1 using (4.73) with $\alpha = 0.01$.

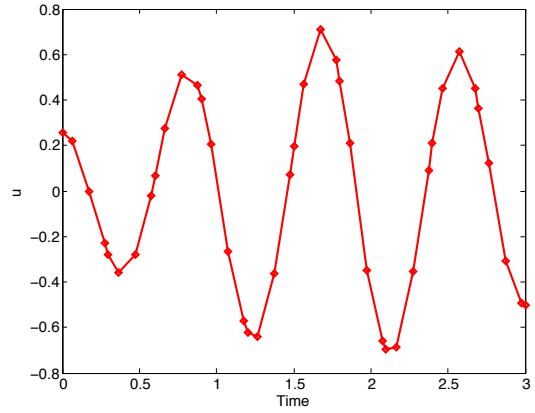


Figure 4.12: Minimal test signal for Example 4.7.1 using (4.73) with $\alpha = 0.001$.

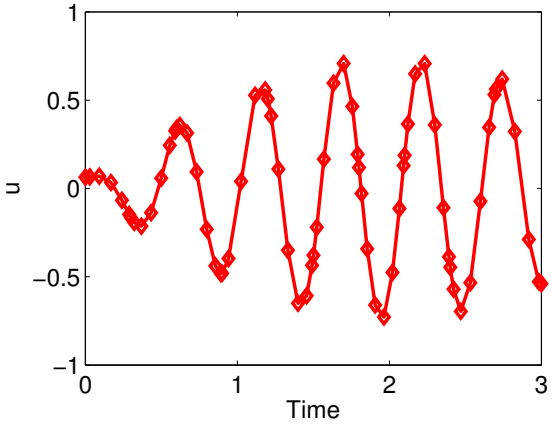


Figure 4.13: Minimal test signal for Example 4.7.1 using (4.73) with $\alpha = 0.0001$.

any function $f(t) \in L^2[-\pi, \pi]$ we have that

$$\lim_{n\to\infty} \int_{-\pi}^{\pi} \sin(nt) f(t) dt = 0.$$

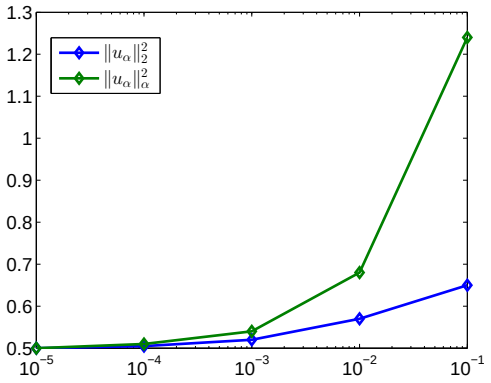


Figure 4.14: α vs. $\|u_\alpha\|_\alpha^2$ and $\|u_\alpha\|_2^2$ for Example 4.7.1.

It is interesting to examine what happens as α goes to 0. Figure 4.14 shows the plots of $\|u_\alpha\|_\alpha^2$ and the L^2 norm squared of the minimal test signal found for a given α. Note that as α goes to zero $\|u_\alpha\|_2^2$ converges to a number around 0.5. However, $\|u_\alpha\|_\alpha^2$ is also converging to the same number. Thus for small α the new way of measuring the test signal is producing a test signal that is almost the same norm as when using the old method.

That $\|u_\alpha\|_\alpha^2$ is close to $\|u_\alpha\|_2^2$ for a signal u_α does not mean that $\|u'_\alpha\|_2$ is small. As shown in Figure 4.15 we see that the $\|u'_\alpha\|_2$ blows up as $\alpha \to 0^+$. It is just that $\alpha\|u'_\alpha\|_2^2 \to 0^+$. This, in fact, tells us a lot about the minimality of test signals as the next proposition shows.

Proposition 4.7.2. *Suppose that we are given the problem of finding an auxiliary test signal for a problem with at least one higher index DAE model as described earlier. Suppose that a minimal proper test signal exists with the test signal measured by* (4.73)*. Denote this test signal by u_α. Suppose that $\lim_{\alpha\to 0} \|u'_\alpha\|_2 = \infty$. Then either there is no minimal proper test signal using the* (4.4) *measure or if there is a minimal test signal u in the sense of* (4.4)*, then u' does not exist as a L^2 function. In particular u cannot be smooth.*

Proof. Let u_α be the minimum proper test signal in the sense of (4.73). Suppose that there is a minimum proper test signal u in the sense of (4.4) which has an L^2 bounded derivative. Then for any $\alpha > 0$ we have

$$\|u\|_2^2 \leq \|u_\alpha\|_2^2 \tag{4.74}$$

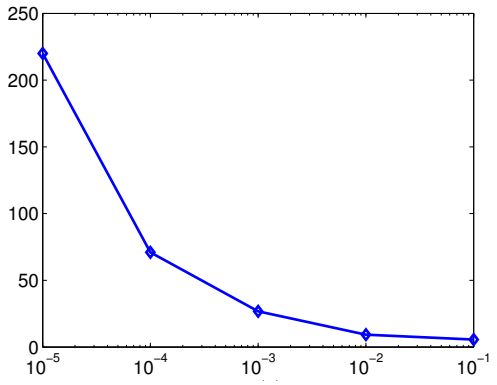


Figure 4.15: α vs. $\|u'_\alpha\|_2^2$ for Example 4.7.1.

due to the minimality of u in the sense of (4.4). However, the minimality of u_α in the sense of (4.73) implies that

$$\|u_\alpha\|_2^2 + \|u'_\alpha\|_2^2 \leq \|u\|_2^2 + \|u'\|_2^2 \tag{4.75}$$

Combining (4.74) and (4.75) we have that

$$\|u'\|_2^2 \geq \|u'_\alpha\|_2^2 \tag{4.76}$$

for all $\alpha > 0$. The proposition now follows since $\lim_{\alpha \to 0} \|u'_\alpha\|_2^2 = \infty$. □

4.7.3.1 Return to Example 4.7.3

We return to Example 4.7.3 but now we select B so that u' and/or u'' appear in the output. In particular, with $B_0 = B_1 = \begin{bmatrix} 1 & 1 & -1 & 1 & 0 & 0 \end{bmatrix}^T$ for both models u' appears in the output of model 0. For $B_0 = B_1 = \begin{bmatrix} 1 & 1 & -1 & 1 & 1 & 1 \end{bmatrix}^T$ we have u' and u'' both appear in the output. However, for this problem the original algorithm quickly found a minimum proper u. The test signal for one case is shown in Figure 4.16. Therefore, the mere presence of a derivative of u in the output does not gaurantee the failure of the original algorithm.

It is interesting in this case to see what happens if we try the modified cost (4.73) on this new example. The result is graphed in Figure 4.17. The result in Figure 4.17 is consistent with Proposition 4.7.2. Here we have $\|u'_\alpha\|_2$ stays bounded as $\alpha \to 0$ and a minimal proper test signal exists.

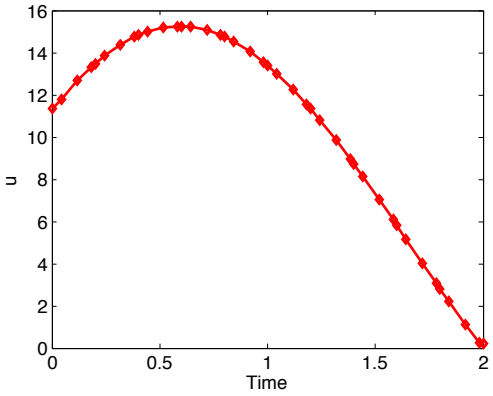


Figure 4.16: Minimum proper test signal for modified Example 4.7.3 with u' appearing in the output.

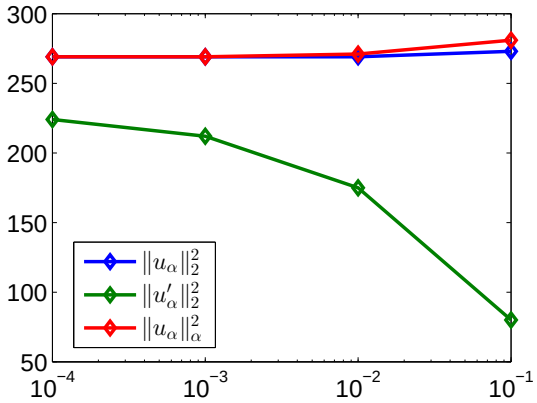


Figure 4.17: $\|u\|_\alpha^2$, $\|u_\alpha\|_2^2$, $\|u'_\alpha\|_2^2$ for modified Example 4.7.3 plotted against α.

4.8 Asynchronous Signal Design

While passive approaches are usually implemented in an ongoing manner, active tests are either done over shorter time intervals or over a series of short time intervals. However, in previous studies the interval on which the test signal is applied is the same as the window over which the system is observed. This need not always be the case. In fact, there may be practical limitations on when or how long the test signal can be applied, but a longer observation window may be appropriate. Also, if there is a delay in the test signal acting on the system there can be a difference between the interval in which the test signal is applied and when the system is observed.

In this section, we begin the consideration of asynchronous signal design, or the calculation of auxiliary signals that may have different test application and observation intervals. We begin with a modification of the framework of [20]. We will see that some parts of the theory are the same but other parts are different. In particular, the optimal test signals can be quite different. Unlike other works which, for example, use Riccati equations, we want our approach to be easily adapted to other problems, such as those with bounds on test signals, so we use direct optimization formulations and numerical solvers. This enables us to reduce the impact of the test signal on system performance during, or at the end, of the test when needed. Section 4.8.1 describes the mathematical formulation and how to characterize a minimal proper test signal for two measures of total model uncertainty. Section 4.8.2 gives computational examples and discusses some numerical issues.

4.8.1 Asynchronous Multi-model Formulation

In this section, we consider a multi-model formulation where both models are ODEs. As before we have one nominal ($i = 0$) and one faulty ($i = 1$) system. Each model has the form

$$x_i' = A_i x_i + B_i u + M_i \nu_i \tag{4.77a}$$

$$y_i = C_i x_i + N_i \mu_i. \tag{4.77b}$$

All matrices are constant in the original models. This is changed in our particular implementation to accommodate the control u being zero on a subinterval.

We assume the test is performed over an interval $0 \leq t \leq T$ and the output $y(t)$ is available for all of $[0\ T]$. However, the test signal u is only applied from r to s where $0 \leq r < s \leq T$. Since the $r = 0$, $s = T$ case is already done, we will focus on the case where $0 < r$ or $s < T$, or both. We assume that u is zero outside of $[r\ s]$. The case where the input is $u + v$ where v is another fixed known input is handled the same way as the $v = 0$ case so we omit v in what follows.

The model uncertainty is still given by (4.2), but to simplify notation in this section we let $P_i = P_{0_i}$. Recall, there are two ways of measuring the total uncertainty,

$$\Gamma_\infty = \max\{\Gamma_0, \Gamma_1\} \tag{4.78a}$$

$$\Gamma_2^2 = \Gamma_0^2 + \Gamma_1^2. \tag{4.78b}$$

Each has advantages. (4.78a) was considered in [20] and several related papers. The case (4.78b) is discussed in [25]. With whichever measure we use, we assume that the total amount of uncertainty is bounded by γ. By rescaling the P_i, M_i, N_i we may assume $\gamma = 1$ in what follows.

We continue to seek the smallest proper test signal and measure it with (4.4). Use of a different measure of u is trivial to implement since it amounts to altering one line of code where the outer cost is given. Several examples are given in Section 4.7 and in [20].

To make the discussion clearer we let $\text{Prob}_{\alpha,\beta}^{r,s}$ be the problem where the observation is over the interval $[\alpha\ \beta]$, the test signal is applied over $[r\ s]$ and $\alpha \le r < s \le \beta$. The classical problem $\text{Prob}_{\alpha,\beta}^{\alpha,\beta}$ is just written $\text{Prob}_{\alpha,\beta}$. Once we have found a proper u for $\text{Prob}_{0,T}^{r,s}$, it will also be proper as a test signal on $[0\ T]$. So the same tests to determine which model is correct given output y can be used as discussed in [20]. However, we do not expect the minimal test signal of this paper for $\text{Prob}_{0,T}^{r,s}$ to be minimal proper for $\text{Prob}_{0,T}$.

4.8.1.1 Existence of a Proper Test Signal

The first question is whether a proper test signal exists. Using the notation of (4.2) and (4.77), let

$$A = \begin{bmatrix} A_0 & 0 \\ 0 & A_1 \end{bmatrix},\ D = \begin{bmatrix} M_0 & 0 & 0 & 0 \\ 0 & 0 & M_1 & 0 \end{bmatrix},$$

$$P_\beta = \begin{bmatrix} \beta P_0 & 0 \\ 0 & (1-\beta)P_1 \end{bmatrix}, J_\beta = \begin{bmatrix} \beta I & 0 \\ 0 & (1-\beta)I \end{bmatrix},$$

$$C = \begin{bmatrix} C_0 & -C_1 \end{bmatrix},\ N = \begin{bmatrix} N_0 & -N_1 \end{bmatrix},\ B = \begin{bmatrix} B_0 \\ B_1 \end{bmatrix}.$$

Then, from [20] using Γ_∞, a minimal proper signal for $\text{Prob}_{0,T}$ exists provided that there is a κ such that the Riccati equation

$$P' = (A - S_{\kappa,\beta}R_{\kappa,\beta}^{-1}C)P + P(A - S_{\kappa,\beta}R_{\kappa,\beta}^{-1}C)^T$$
$$- PC^T R_{\kappa,\beta}^{-1} CP + Q_{\kappa,\beta} - S_{\kappa,\beta}R_{\kappa,\beta}^{-1}S_{\kappa,\beta}^T, \tag{4.79}$$

$$P(0) = P_\beta \tag{4.80}$$

has a solution, where

$$\begin{bmatrix} Q_{\kappa,\beta} & S_{\kappa,\beta} \\ S_{\kappa,\beta}^T & R_{\kappa,\beta} \end{bmatrix} = \begin{bmatrix} D & B \\ N & 0 \end{bmatrix} \begin{bmatrix} J_\beta & 0 \\ 0 & -\kappa I \end{bmatrix} \begin{bmatrix} D & B \\ N & 0 \end{bmatrix}^T. \tag{4.81}$$

This formulation guarantees proper test signals for either the Γ_∞ or the Γ_2 noise measures. We will discuss the two cases separately.

4.8.1.2 Γ_2 Total Noise Measure

We use (4.78b) and proceed by getting a computable characterization of a proper signal and then using numerical software to carry out the final optimization using this characterization as a nontrivial optimization constraint. The first step is to have a characterization of the smallest noise that will give $y_0 = y_1$ for a given u. Recall, this is the inner minimization problem. Given a test signal u we must characterize

$$\min_{\mu_i,\nu_i,x_i(0)} \Gamma_0^2 + \Gamma_1^2 \tag{4.82a}$$

$$x_0' = A_0 x_0 + B_0 u + M_0 \nu_0 \tag{4.82b}$$

$$x_1' = A_1 x_1 + B_1 u + M_1 \nu_1 \tag{4.82c}$$

$$0 = C_0 x_0 + N_0 \mu_0 - (C_1 x_1 + N_1 \mu_1) \tag{4.82d}$$

We assume that the N_i are invertible. Note that this is not restrictive since it just means that we allow more noise and make the solution a bit more robust. Then we can use (4.82d) to eliminate one of the μ_i. Suppose that we eliminate μ_1 so that

$$\begin{aligned} \mu_1 &= N_1^{-1} (C_0 x_0 + N_0 \mu_0 - C_1 x_1) \\ &= \bar{C}_0 x_0 + \bar{N}_0 \mu_0 - \bar{C}_1 x_1. \end{aligned} \tag{4.83a}$$

We drop the bars in the rest of the section. Then the inner minimization problem is

$$\min_{\eta,\nu_i,x(0)} \frac{1}{2} x(0)^T P x(0) + \frac{1}{2} \int_0^T x^T S_1 x + 2 x^T S_2 \eta + \eta^T S_3 \eta \, dt \tag{4.84a}$$

where

$$x' = Ax + Bu + M\eta \tag{4.84b}$$

and

$$\eta = \begin{bmatrix} \nu_0 \\ \nu_1 \\ \eta_0 \end{bmatrix},\ M = \begin{bmatrix} M_0 & 0 & 0 \\ 0 & M_1 & 0 \end{bmatrix},\ P = \begin{bmatrix} P_0 & 0 \\ 0 & P_1 \end{bmatrix}$$

$$S_1 = \begin{bmatrix} C_0^T C_0 & -C_0^T C_1 \\ -C_1^T C_0 & C_1^T C_1 \end{bmatrix},\ S_2 = \begin{bmatrix} 0 & 0 & C_0^T N_0 \\ 0 & 0 & -C_1^T N_0 \end{bmatrix},$$

and $S_3 = \text{diag}(I, I, I + N_0^T N_0)$.

The Hamiltonian is

$$H = \frac{1}{2}(x^T S_1 x + 2x^T S_2 \eta + \eta^T S_3 \eta) + \lambda^T (Ax + Bu + M\eta).$$

Then the value of the minimum noise consistent with the same output for both models is $\eta = -S_3^{-1}(M^T\lambda + S_2^T x)$ and this makes the necessary conditions for the inner minimization problem equivalent to

$$x' = (A - MS_3^{-1}S_2^T)x + Bu - MS_3^{-1}M^T\lambda \tag{4.85a}$$

$$-\lambda' = (A^T - S_2 S_3^{-1} M^T)\lambda + (S_1 - S_2 S_3^{-1} S_2^T)x. \tag{4.85b}$$

The boundary conditions for the inner minimization problem are $Px(0)+\lambda(0) = 0$ and $\lambda(T) = 0$.

Then the optimization problem $\text{Prob}_{0,T}^{r,s}$ to be solved is

$$\text{Find } J = \inf_u ||u||^2 \tag{4.86a}$$

where u is zero outside of $[r\ s]$ and the following constraints hold

$$x' = (A - MS_3^{-1}S_2^T)x + Bu - MS_3^{-1}M^T\lambda \tag{4.86b}$$

$$-\lambda' = (A^T - S_2 S_3^{-1} M^T)\lambda + (S_1 - S_2 S_3^{-1} S_2^T)x \tag{4.86c}$$

$$\psi' = \frac{1}{2}(x^T S_1 x + 2x^T S_2 \eta + \eta^T S_3 \eta) \tag{4.86d}$$

with the boundary conditions

$$Px(0) + \lambda(0) = 0,\ \lambda(T) = 0 \tag{4.87a}$$

$$\psi(0) = \frac{1}{2}x^T(0)Px(0),\ \psi(T) \geq 1. \tag{4.87b}$$

(4.86d) is a convenient way to compute the noise measure and (4.87b) ensures the signal we find will be proper.

There are several different ways to implement a zero u outside of the interval $[r\ s]$. While

they are all mathematically equivalent, they are not computationally equivalent. One method, if the software being used has this feature, is to use phases corresponding to the intervals $[0\ r]$, $[r\ s]$, and $[s\ T]$. Another method is to have B be zero outside of the interval $[r\ s]$ and use the cost $\|u\|_{0,T}$. Then u will be zero outside of $[r\ s]$ because those values have no affect on the state but do appear in the cost. This second formulation can lead to finer grids and take more CPU time to accommodate discontinuities of u at r, s. Phases allow u to be discontinuous at r, s. We shall comment on this more later.

4.8.1.3 Γ_∞ Total Noise Measure

Now we consider the analogous problem of finding the smallest auxiliary signal, this time subject to the Γ_∞ bound in (4.78a). Let

$$\sigma(u) = \inf_{\substack{x_0(0),\, x_1(0) \\ \nu_0, \nu_1, \mu_0, \mu_1}} \max\{\Gamma_0,\, \Gamma_1\}.$$

Following [20] we can express $\sigma(u)$ as $\sigma(u) = \max_{\beta \in [0,\,1]} \phi_\beta(u)$ where

$$\phi_\beta(u) = \inf_{\substack{x_0(0),\, x_1(0) \\ \nu_0, \nu_1, \mu_0, \mu_1}} \beta\Gamma_0 + (1-\beta)\Gamma_1. \tag{4.88}$$

Then the problem under consideration is

$$J = \min_u ||u|| \tag{4.89a}$$
$$x_0' = A_0 x_0 + B_0 u + M_0 \nu_0 \tag{4.89b}$$
$$x_1' = A_1 x_1 + B_1 u + M_1 \nu_1 \tag{4.89c}$$
$$0 = C_0 x_0 + N_0 \mu_0 - (C_1 x_1 + N_1 \mu_1) \tag{4.89d}$$
$$1 \le \sigma(u). \tag{4.89e}$$

Solving for μ_1, $\mu_1 = C_0 x_0 + N_0 \mu_0 - C_1 x_1$ and we can express the cost as

$$\begin{aligned}\phi_\beta(u) = \inf_{x(0),\,\eta} & \frac{1}{2} x^T(0) P_\beta x(0) \\ & + \frac{1}{2}\int_0^T x^T S_{1,\beta} x + 2x^T S_{2,\beta}\eta + \eta^T S_{3,\beta}\eta \ dt\end{aligned}$$

where

$$P_\beta = \begin{bmatrix} \beta P_0 & 0 \\ 0 & (1-\beta)P_1 \end{bmatrix},\ x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix},\ \eta = \begin{bmatrix} \nu_0 \\ \nu_1 \\ \mu_0 \end{bmatrix},$$

$$S_{1,\beta} = (1-\beta)\begin{bmatrix} C_0^T C_0 & -C_0^T C_1 \\ -C_1^T C_0 & C_1^T C_1 \end{bmatrix},$$

$$S_{2,\beta} = (1-\beta)\begin{bmatrix} 0 & 0 & C_0^T N_0 \\ 0 & 0 & -C_1^T N_0 \end{bmatrix},$$

$$S_{3,\beta} = \begin{bmatrix} \beta I & 0 & 0 \\ 0 & (1-\beta)I & 0 \\ 0 & 0 & \beta I + (1-\beta)N_0^T N_0 \end{bmatrix}.$$

We can now reformulate the problem as

$$J = \min_u ||u|| \tag{4.90a}$$

$$x' = Ax + Bu + M\eta \tag{4.90b}$$

$$1 \leq \phi_\beta(u) \tag{4.90c}$$

$$0 \leq \beta \ \leq 1 \tag{4.90d}$$

where

$$A = \begin{bmatrix} A_0 & 0 \\ 0 & A_1 \end{bmatrix},\ M = \begin{bmatrix} M_0 & 0 & 0 \\ 0 & M_1 & 0 \end{bmatrix},\ B = \begin{bmatrix} B_0 \\ B_1 \end{bmatrix}.$$

Again the restriction to r, s can be done with phases or altering B. We preferred phases.

We proceed by finding necessary conditions for the inner min that can be easily implemented in optimal control software. The necessary conditions are analogous to the ones in (4.86b)-(4.86c) except with the β parameter included. Therefore, the formulation of the problem that permits an efficient numerical solution is

$$J = \min_u ||u|| \tag{4.91a}$$

$$\begin{aligned} x' = {} & (A - MS_{3,\beta}^{-1}S_{2,\beta}^T)x + Bu \\ & - MS_{3,\beta}^{-1}M^T\lambda \end{aligned} \tag{4.91b}$$

$$\begin{aligned} -\lambda' = {} & (A^T - S_{2,\beta}S_{3,\beta}^{-1}M^T)\lambda \\ & + (S_{1,\beta} - S_{2,\beta}S_{3,\beta}^{-1}S_{2,\beta}^T)x. \end{aligned} \tag{4.91c}$$

$$\psi' = \frac{1}{2}(x^T S_{1,\beta}x + 2x^T S_{2,\beta}\eta + \eta^T S_{3,\beta}\eta) \tag{4.91d}$$

with the boundary conditions $P_\beta x(0) + \lambda(0) = 0$, $\lambda(T) = 0$, $\psi(0) = \frac{1}{2}x^T(0)P_\beta x(0)$, $\psi(T) \geq 1$, and $0 \leq \beta \leq 1$.

4.8.2 Computational Tests

In this section we examine a computational example that will illustrate several points. We suppose that the observation window is [0 3] and the normal and failed models are

$$x_0' = \begin{bmatrix} -1 & 2 \\ 1 & -2.5 \end{bmatrix} x_0 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \nu_0 \tag{4.92a}$$

$$y = \begin{bmatrix} 1 & 1 \end{bmatrix} x_0 + \mu_0 \tag{4.92b}$$

and

$$x_1' = \begin{bmatrix} -1.5 & 3 \\ -1 & -0.5 \end{bmatrix} x_1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \nu_1 \tag{4.93a}$$

$$y = \begin{bmatrix} 1 & 1 \end{bmatrix} x_1 + \mu_1 \tag{4.93b}$$

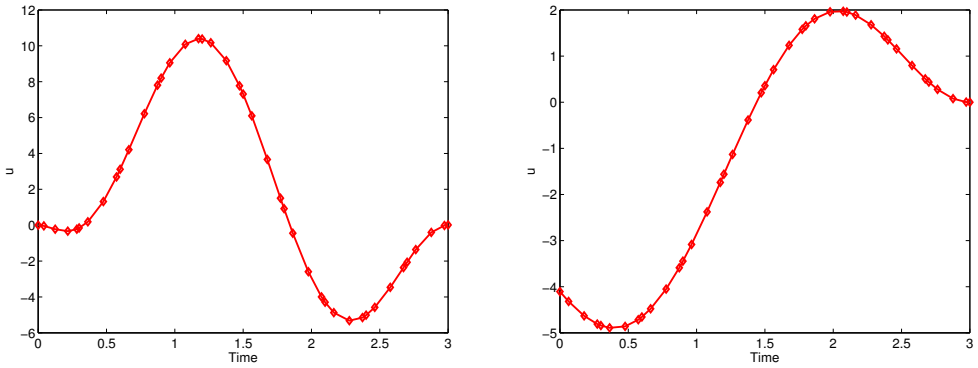
respectively so the fault is a change in A_0. We will consider the case when $P_0 = P_1 = 0$ and the case when $P_0 = P_1 = I$. Note that for every proper test signal that follows, both u and $-u$ are proper test signals of the same norm. All figures and results provided in this section are with respect to the Γ_2 total noise measure. Selected simulations for the Γ_∞ total noise measure were also performed for validation, but are not included here. Generally, the β parameter was very close to $\frac{1}{2}$ and the shape of the proper signals are independent of the choice of total noise measure for this example. The proper signal in the Γ_∞ total noise measure is bigger than the Γ_2 total noise measure by a factor of about $\sqrt{2}$. Due to the extra parameter β, the Γ_∞ formulation took considerably longer to numerically solve. A summary of the results from all tests in this section is given in Table 4.6.

4.8.3 Test 1: ($r = 0$, $s = T = 3$)

We first solve the problem $\text{Prob}_{0,3}$ to see what the test signal is if the whole observation interval can be used for applying the test signal. We compare the cases where $P_i = 0$ and $P_i = I$ for both models to simulate having some information or having no information on the state at the start of the test. The minimal proper signals are shown in Figure 4.18. As expected when $P_i > 0$, the minimal test signal is not zero at the start of the test interval.

4.8.4 Test 2: ($r = 0$, $s = T = 1$)

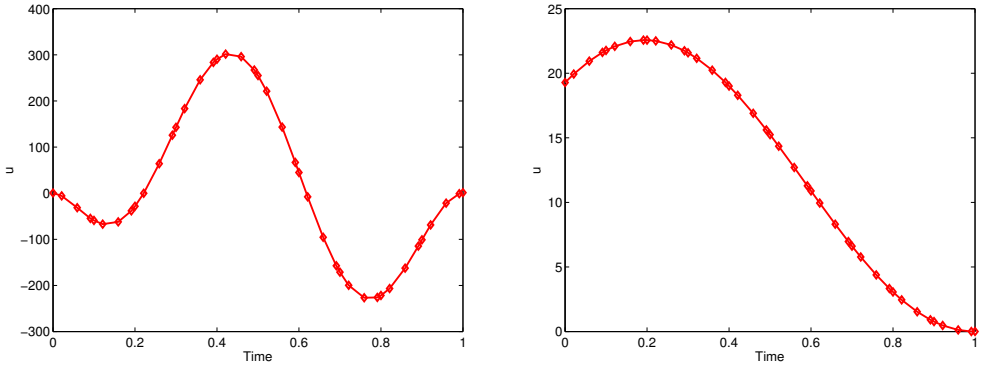
Suppose now that we can only apply the test signal for one unit of time. As is traditionally done if we want to run the test on [0 1], we solve $\text{Prob}_{0,1}^{0,1}$ and get the minimal proper test signals shown in Figure 4.19. As to be expected the shorter interval leads to larger test signals



(a) $P_i = 0$, $||u|| = 9.28$ (b) $P_i = I$, $||u|| = 4.82$

Figure 4.18: Minimal proper test signals for Test 1.

in Figure 4.19 than in Figure 4.18. The shapes can also be different as Figures 4.18b and 4.19b show.



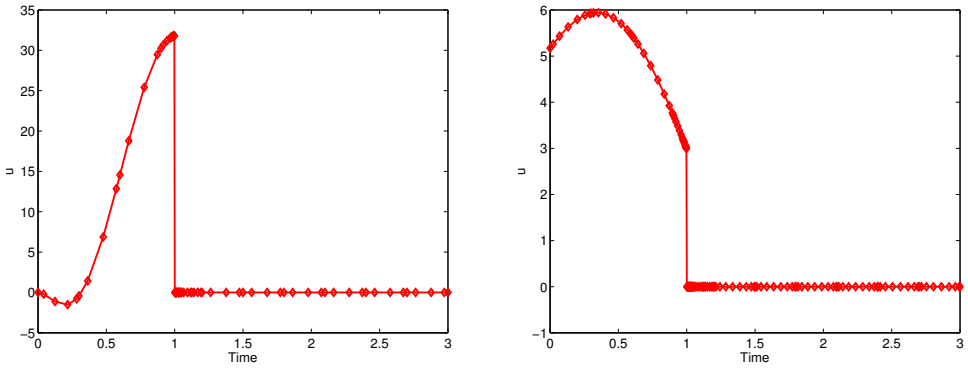
(a) $P_i = 0$, $||u|| = 162.84$ (b) $P_i = I$, $||u|| = 15.53$

Figure 4.19: Minimal proper test signals for Test 2.

4.8.5 Test 3: ($r = 0$, $s = 1$, $T = 3$)

Now suppose that we can only apply the test signal for one time unit as in Test 1, but that we can observe the output for three time units. That is, we will solve $\text{Prob}_{0,3}^{0,1}$. The result is in Figure 4.20.

Comparing Figure 4.20 to Figures 4.18 and 4.19, two things are immediately apparent. For



(a) $P_i = 0$, $||u|| = 16.89$ (b) $P_i = I$, $\|u\| = 5.23$

Figure 4.20: Minimal proper test signals for Test 3.

one, the minimal test signals have dramatically different shapes. For example, the minimum proper test signals are no longer zero at the end of their application at time s. Secondly, there is a definite benefit to the extra observation time. In the $P_i = 0$ case, the size of the test signal has dropped from 162.84 to 16.89 which is much closer to the 9.28 when a test signal is applied over the full [0 3]. In the $P_i = I$ case the size of the test signal has dropped from 15.53 to 5.23 which is close to the 4.82 of applying the test signal over the full [0 3]. The same pattern holds if we are interested in the maximum value of our test signals.

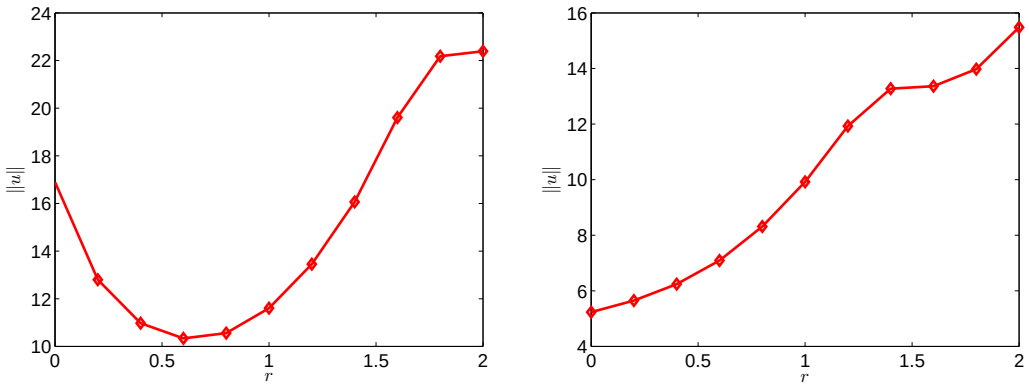
These three examples clearly show that there is an advantage of conducting extra observations. But if it is possible to start the observations earlier than the start of the test signal is there an advantage of doing so? In Figure 4.21 we plot the L^2 norm of the minimal proper test signal for $\text{Prob}_{0,3}^{r,s}$ with $0 \leq r \leq 2$ and $s = r + 1$.

We see dramatically different behavior in the two cases. With $P_i = I$, where there is restriction on the initial condition, we see that the smallest test signal is gotten by taking $r = 0$ as in Test 3. However, with $P_i = 0$, where there is no information on the initial condition we see that there is a definite advantage in applying the test signal a little later. In fact the minimum is with r around 0.6.

4.8.6 Test 4: $(s - r = 1,\ T = 3)$

In this example, we treat r as an optimization parameter and fix the length of the testing interval at 1 so that we have $s = 1 + r$ when we find the minimal u. Including this free parameter r adds one more degree of freedom and the problem becomes

$$\text{Find } J = \inf_{u,r} ||u||^2 \tag{4.94a}$$



(a) $P_i = 0$ (b) $P_i = I$

Figure 4.21: Norm of minimal proper test signals for different r in Test 4.

$$x' = (A - MS_3^{-1}S_2^T)x + B(r;t)u - MS_3^{-1}M^T\lambda \tag{4.94b}$$

$$-\lambda' = (A^T - S_2S_3^{-1}M^T)\lambda + (S_1 - S_2S_3^{-1}S_2^T)x \tag{4.94c}$$

$$\psi' = \frac{1}{2}(x^TS_1x + 2x^TS_2\eta + \eta^TS_3\eta) \tag{4.94d}$$

with the boundary conditions

$$Px(0) + \lambda(0) = 0,\ \lambda(T) = 0 \tag{4.95a}$$

$$\psi(0) = \frac{1}{2}x^T(0)Px(0),\ \psi(T) \geq 1 \tag{4.95b}$$

$$0 \leq r \leq 2 \tag{4.95c}$$

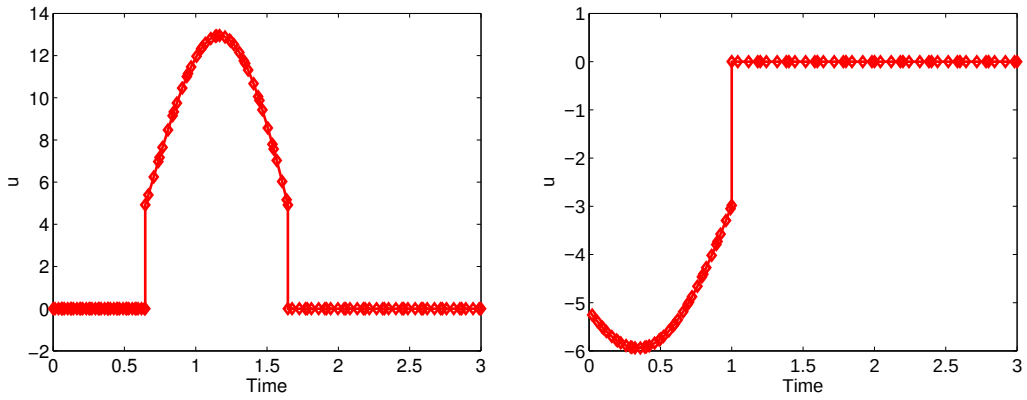
and

$$B = \begin{cases} \begin{bmatrix} B_0 \\ B_1 \end{bmatrix}, & t \in [r,\ r+1] \\ 0, & t \notin [r,\ r+1]. \end{cases}$$

From Figure 4.21 we know that for $P_i = I$, the minimum test signal of length one should be applied at the start of the interval with $r = 0$. However, the situation is different with $P_i = 0$. When we optimize over r we get the result in Figure 4.22 with an r value of 0.65. By shifting the start of the test to $r = 0.65$ instead of $r = 0$ we have reduced the size of the test signal needed with a [0 3] observation window from 16.89 on [0 1] to 10.31 on [0.65 1.65], which is a

major improvement.

As expected Figure 4.22b is essentially the same as Figure 4.20b. In these problems u is the minimum proper test signal if and only if $-u$ is also a minimum proper test signal. The optimization software may find either of the two solutions. Note that Figure 4.22b is the negative of Figure 4.20b. For a given computed u it might be more practical to apply u or $-u$ depending on the way the test signal is generated.



(a) (b)

Figure 4.22: Minimum proper signals over all possible r for Test 4. $r = 0.65$, $P_i = 0$, $||u|| = 10.31$ in (a) and $r = 0$, $P_i = I$, $||u|| = 5.23$ in (b).

Table 4.6: Norm of the optimal test signal for Tests 1-4 and each case of P_i.

	Test 1	Test 2	Test 3	Test 4
Problem	$P_{0,3}$	$P_{0,1}$	$P_{0,3}^{0,1}$	$P_{0,3}^{r,r+1}$
$P_i = 0$	**9.28**	**162.84**	**16.89**	**10.31**
$P_i = I$	**4.82**	**15.53**	**5.23**	**5.23**

4.8.7 Comments on Computations

Unlike the DAE models in Section 4.1, these models are simpler ODE models; however, the optimization problems are still fairly complex. For example, they include inequality constraints on integrals. Also, as pointed out in [1], it may be desirable to put bounds or other restrictions

on the test signals. When choosing software it is desirable to avoid codes that rely on the necessary conditions. Accordingly, as in Section 4.1, we use the direct transcription software package GPOPS-II [35, 62]. Direct transcription software avoids the necessary conditions and good transcription software can handle very complex problems. The problem is fully discretized on a grid and then solved using a nonlinear programming algorithm. The solution is evaluated and, if necessary, the grid is refined and the problem solved again in an iterative manner until tolerances are met.

In a direct transcription approach the problem can be described differently on different time intervals. In particular, controls can abruptly change when going from one interval to another without forcing fine grids near the change. These intervals are called phases. Both GPOPS-II and SOCX [6] have the ability to work with problems defined in phases. Thus (4.94)–(4.95) can be implemented as either a single problem with a discontinuous B matrix or as a problem in three phases with the control only active in the middle phase. We found the three phase version to be much faster computationally.

Neither SOCX nor GPOPS-II permit the calling of themselves. When optimizing over r, as in Test 4, it is required to nest the optimization problems with another optimizer. For the problems discussed here, the optimizer fmincon from MATLAB suffices. We define a function $h(r)$ that solves $\text{Prob}_{0,T}^{r,s}$ for a fixed r using GPOPS. Then, fmincon optimizes $h(r)$ over r. When solving the problem this way it is important that the minimization of $h(r)$ for a given r be accurate enough to permit fmincon to find a good enough r value. On the other hand, too tight of tolerances can greatly increase problem solution time by making each function evaluation of $h(r)$ very expensive in terms of CPU time. For this problem we found a GPOPS-II relative mesh tolerance of 10^{-3} and a fmincon Tolfun of 0.1 worked well.

4.9 Conclusions for Chapter 4

This chapter has presented work on the study of auxiliary signal design in systems modeled by DAEs for fault detection purposes, originally given in [21, 66, 67]. While we have pointed out some references on auxiliary signal design for ODE systems, the analogous DAE problem has been rarely studied, making it a novel consideration.

In Section 4.1, we give a procedure for the reformulation of Problem 4.1.1, a bi-level optimization problem with inequality and high index LTI DAE constraints, into Problem 4.1.2, a straightforward optimization problem with LTI ODE constraints. The goal is to compute a minimal proper auxiliary signal that facilitates fault detection. The minimal proper signal is shown to always exist under the assumption that a Riccati equation has a solution and a matrix inequality holds in Section 4.1.4. This treatment is under the presence of bounded additive uncertainty. An extension of this procedure to models with model uncertainty is presented in

Section 4.6.

A technique for efficient on-line detection and identification has been introduced in Section 4.2. Theorem 4.2.1 is a new contribution and is an integral part of the realizability test because it yields a simple and efficient method for computing the past cost on the noise. If measurements are being conducted in real-time, the past cost for each model can be computed in real-time, meaning we can determine a non-realizable model as soon as the past cost rises above a threshold. Without the theorem, we would have to solve constrained optimization problems (see (4.29)) at every time we would like the past cost. Obviously, this is too expensive computationally for real-time detection.

We have summarized and given steps for implementation of our approaches in the algorithms in Section 4.3. The numerical examples of Section 4.4 and 4.5 are also helpful guides for implementation purposes. Even nonlinear problems can benefit from this approach, as discussed in the experiments of Section 4.5. We were able to find a proper signal u, and implement the model identification results on a nonlinear model of a robot arm with elastic joints.

The conditions given in Section 4.1.4 are sufficient for the existence of a minimal proper signal. When they do not hold, there may or may not be a minimal proper signal and the optimization routine may fail. In Section 4.7.1, we give examples of this failure and observe that a common characteristic in the models that exhibit optimization failure is the presence of derivatives of the test signal appearing in the output. The failure of the algorithm appears to be due to an issue of weak convergence and nonexistence of a minimum in L^2. This is verified with three examples. Section 4.7.2 gives an easy to compute test that enables one to determine a priori when a failure might occur.

Finally, a new algorithm is introduced in Section 4.7.3 to find a nearly minimal proper signal in the cases where the previous algorithm fails. The modification is an alteration of the cost on the test signal, given in (4.73). By introducing u' into the cost, we can prevent $\lim_{\alpha\to 0} \|u'_\alpha\|_2 = \infty$ from causing a problem in the optimization. Implementation of the new algorithm on the failed examples shows convergence of the new algorithm, and testing on an example that did not fail shows the algorithm produces test signals that are close to the true optimal signal and get closer as we let $\alpha \to 0$.

In Sections 4.1-4.7, the interval on which the test signal is applied is the same as the window over which the system is observed. However, this doesn't have to be the case. Section 4.8 explores asynchronous signal design, or designing test signals on intervals that are subsets of the observation intervals. Experimentation shows that allowing for a longer output collection interval proves advantageous in that it permits the use of smaller test signals. If there is information on the state at the start of the test interval, it is often best to apply the test signal as soon as possible. However, if there is no information available on the initial state, it can be advantageous to take measurements prior to the onset of the test signal. These results are

important because the restrictions on the application of the test signal are often much more stringent than restrictions on the observation window.

Chapter 5

Contributions and Future Work

5.1 Publications

Much of Chapters 2-3 can be found in the first two publications. The main ideas of Chapter 4 were originally given in the last three publications.

[65]: Scott, J.R. & Campbell, S.L. "Observer based fault detection in differential-algebraic equations." *Proc. 2013 SIAM Conference on Control and its Applications*. San Diego, CA, 2013, pp. 176-183.

[64]: Scott, J.R. & Campbell, S.L. "Observer based fault detection and identification in differential-algebraic equations." *Proc. 2013 ASME Dynamic Systems and Control Conference*. Palo Alto, CA, 2013.

[66]: Scott, J.R. & Campbell, S.L. "Auxiliary Signal Design for Failure Detection in Differential-Algebraic Equations." *Numerical Algebra, Control, and Optimization* **4** (2014), pp. 151-179.

[67]: Scott, J.R. & Campbell, S.L. "Auxiliary signal design for failure detection in high index differential-algebraic equations." *Proc. 2014 IEEE Conference on Decision and Control*. San Diego, CA, 2014.

[21]: Campbell, S.L. & Scott, J.R. "Asynchronous auxiliary signal design for failure detection." *Proc. 2014 IEEE Conference on Systems, Man, and Cybernetics*. San Diego, CA, 2014.

5.2 Presentations Outside of NCSU

- *Observer Based Fault Detection in Differential Algebraic Equations*
  2013 SIAM Conference on Control and its Applications
  San Diego, CA, July 2013

- *Observer Based Fault Detection and Identification in Differential Algebraic Equations*
  2013 ASME Dynamic Systems and Control Conference
  Palo Alto, CA, October 2013

- *Fault Detection in High Index DAE*
  2014 IEEE Conference on Decision and Control
  Los Angeles, CA, December 2014

5.3 Contributions

In this thesis, we have given the following:

Chapter 2

- (Proposition 2.4.1, Lemma 2.4.1-2.4.4) previously unknown properties of LTI completions;

Chapter 3

- a novel method of generating residuals and preforming observer-based fault detection, using Luenberger observers and results on completion techniques, applied to linear DAEs, where no assumptions are made on the index of the DAE;
- (Proposition 3.2.1) a lower bound on the size of the fault, in terms of κ, for detection by time t, given bounds on the noise;
- (Proposition 3.2.2) a lower bound on the time to detection;
- a lower bound on the size of the user-specified threshold vector τ so that no false alarms will occur;
- (Propositions 3.3.1-3.3.3) procedures to create filtering matrices so that filtered residuals have unidirectional properties facilitating identification, given a library of faults;
- an extension of the identification results, to include faults that are a linear combination of faults;
- an estimator of the the scaling factor, κ, for each fault in the combination;
- a method of disturbance attenuation using a frequency filtering technique;

Chapter 4

- a method to determine the optimal inputs for guaranteed active fault diagnosis by solving a bilevel optimization problem in the case where the constraint dynamics are linear DAE;

- an extension to the model uncertainty case;
- sufficient conditions on the existence of a minimal proper signal;
- a method to test whether a given signal is close to minimal proper;
- an efficient test to correctly identify faulty models given an output (Theorem 4.2.1);
- the algorithms needed to efficiently implement the results of this thesis;
- evidence of the utility of our approach on problems with nonlinearities (robot arm example in Section 4.5);
- evidence of a class of problems for which the reduction procedure and resulting optimal control problem of Section 4.1 fails and criteria to determine when said method has a greater chance of failure;
- a variation of the approach in Section 4.1 enabling one to still compute useful, nearly optimal, proper test signals in the cases where the previous method does not;
- examples showing the test interval and observation window need not be equivalent and that a longer output collection interval can prove advantageous by permitting the use of smaller test signals; *and*
- examples showing that starting the test signal as soon as possible is best when there is information on the initial state, but taking output information prior to implementing the test signal may be advantageous when there is no information on the initial state.

5.4 Future Research

1. OBSERVER-BASED DETECTION FOR NONLINEAR AND LTV DAES: We made the assumption in Chapter 3 that the DAE in question is solvable and assumptions I.-IV. are satisfied. No assumptions were made on the structural form, such as Hessenburg form or index restrictions were made. Therefore, the approach chosen is one that also holds great promise for LTV and nonlinear systems. Recently, advances have been made in observer design for LTV [8] and nonlinear [73] DAEs. Moreover, there have been recent discoveries in completion theory for LTV [57] and nonlinear [18] DAE systems. The techniques of this chapter are general enough to be extended to these two more complex classes of DAEs and it is hoped that future work in this area includes these extensions.

2. DELAY DIFFERENTIAL ALGEBRAIC EQUATIONS: Many physical systems also possess delays either in the dynamics or in the application of the control. In [29], fault detection is discussed in the context of linear systems with delays. This leads us to believe the methods presented in this work could be extended to DDAE (delayed differential algebraic

equation) models. A typical FDI problem for DDAE models could be to minimize (5.1a) with constraints (5.1b)-(5.1f).

$$J = \|u\|^2 \tag{5.1a}$$
$$\gamma^2 \leq \inf \Gamma^2(u) \tag{5.1b}$$
$$0 = f_i(x_i', x_i, t, x_i(\omega_i(t)), u, u(\psi(t)), \mu_i), \;\; i = 1, 2, \;\; t_0 \leq t \leq t_f \tag{5.1c}$$
$$0 = y_0 - y_1, \;\; t_0 \leq t \leq t_f \tag{5.1d}$$
$$x_i = \alpha_i(t), \;\; -r \leq t < t_0 \tag{5.1e}$$
$$u = \beta(t), \;\; -s \leq t < t_0 \tag{5.1f}$$

$\omega_i(t)$ and $\psi_i(t)$ are time delay functions. In the case of single constant state and control delays, $\omega(t) = t - r$ and $\psi(t) = t - s$. $\alpha_i(t)$ and $\beta(t)$ are prehistory functions active on the specified interval where $r, s > 0$. The dynamics, f_i, are, in general, DDAEs. The rest of the notation here is consistent with Chapter 4.

Recall, in our simulations, we used a direct transcription approach–specifically GPOPS-II– to compute the minimal auxiliary signal. However, GPOPS-II does not allow for delays. It does allow for phases. For some special cases, we may be able to put the problem in an undulated form using the method of steps (MOS) and still use GPOPS. Another direct transcription approach, SOCX, part of the SOS suite (http://www.appliedmathematicalanalysis.com), does allow for delays, but doesn't allow for phases in delayed models. One goal is to suggest a reformulation that SOCX can solve. Ideally, the DAE would be index one, but SOCX can sometimes work with higher index DAEs depending on the cost function [31]. Another area of future work is to develop a reformulation of these higher index problems that can be implemented in direct transcription software, such as GPOPS-II or SOCX.

3. MULTIPLE FAULTY MODELS: In Chapter 4, we assumed the existence of only two models–one nominal, the other faulty. The approach in this paper can be applied to problems with more than one type of fault by designing a test signal for each pair of models and applying the test signals sequentially. This process could be formalized for DAE models. Another interesting, related area of future work is to design a proper signal for more than two concurrent models and identify the correct failure using only one test signal.

4. AUXILIARY SIGNAL DESIGN FOR NONLINEAR MODELS: The theory behind using linearized models to get test signals for nonlinear ODE models has been discussed formally in [68]. The example in 4.5, shows that this approach, at least sometimes, is applicable to nonlinear DAE models as well. While the robotic arm example behaved well, the different,

at times unpleasant, behavior of linearized DAE models compared with those of ODE models may result in more complicated theory. In general, one expects a linearized DAE to provide local information on the nonlinear DAE it originates from. However, linearized DAEs can feature quite different phenomena. A linearized DAE may show lower or higher index than the original DAE, suggest incorrect stability, observability, etc., and fail to remain regular [49]. Formalizing the theory and developing a characterization of when using linearized models to get test signals is possible is an important area of future work.

5. SUFFICIENT CONDITION FOR ALGORITHMIC FAILURE: In Section 4.7, we presented several examples where the original method for auxiliary signal design failed and a test to determine a priori when this failure might occur. However, it was also shown that even a positive outcome of the test does not guarantee failure. Future work should be to determine a verifiable condition for this failure, i.e. the nonexistence of a minimum proper test signal.

6. EXTENSIONS OF ASYNCHRONOUS SIGNAL DESIGN: In the area of asynchronous signal design, there might be circumstances where it is desirable for at least part of the test interval to lie outside of the observation interval. Fault detection applied to DDAE or DDE (delayed differential equations) models, discussed above, is a natural place where this could occur. Control delays, for example, would most naturally imply an observation interval that starts after the application of the test signal. The approach given in Section 4.8 cannot be directly used on such problems because (4.83a) cannot be used on all of the test interval.

   All of the discussion in Section 4.8 considers ODE models. It may also be possible to extend these ideas to differential algebraic equations.

BIBLIOGRAPHY

[1] Andjelkovic, I. & Campbell, S. L. "Direct optimization determination of auxiliary test signals for linear problems with model uncertainty". *Proc. Decision and Control and European Control Conference*. 2011, pp. 909–914.

[2] Andjelkovic, I. et al. "Active fault detection in nonlinear systems using auxiliary signals". *Proc. American Control Conference*. Seattle, WA, 2008.

[3] Astorga-Zaragoza, C. M. et al. "Fault Diagnosis for a Class of Descriptor Linear Parameter Varying Systems". *Int. Journal of Adaptive Control and Signal Processing* **26**.3 (2012), pp. 208–223.

[4] Bakiotis, C. et al. "Parameter identification and discriminant analysis for jet eingine mechanical state diagnosis". *Proc. IEEE Conference on Decision and Control*. Fort Lauderdale, FL, 1979.

[5] Bellman, R. E. *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[6] Betts, J. T. *Practical Methods for Optimal Control and Estimation using Nonlinear Programming*. 2nd Ed. SIAM, 2010.

[7] Bobinyec, K. et al. "Maximally reduced observers for linear time varying DAEs". *Proc. IEEE Multiconference on Systems and Control*. Denver, CO, 2011, pp. 1373–1378.

[8] Bobinyec, K. et al. "Constructing observers for linear time varying DAEs". *Proc. IEEE Conference on Decision and Control*. Maui, HI, 2012, pp. 5749–5754.

[9] Boggess, A. & Narcowich, F. J. *A First Course in Wavelets with Fourier Analysis*. John Wiley & Sons, 2009.

[10] Brenan, K. et al. *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*. Philadelphia, PA: SIAM, 1996.

[11] Bryson, A. E. & Ho, Y. C. *Applied Optimal Control*. New York: Hemisphere, 1975.

[12] Campbell, S. L. "The Numerical Solution of Higher Index Linear Time Varying Singular Systems of Differential Equations". *SIAM Jour. of Sci. and Stat. Computing* **6** (1985), pp. 334–348.

[13] Campbell, S. L. "Uniqueness of Completions for Linear Time Varying Differential Algebraic Equations". *Linear Algebra and its Applications* **161**.15 (1992), pp. 55–67.

[14] Campbell, S. L. "Least Squares Completions for Nonlinear Differential Algebraic Equations". *Numerical Mathematics* **65** (1993), pp. 77–94.

[15] Campbell, S. L. "Linearization of DAEs Along Trajectories". *Z. angew Mathematical Physics (ZAMP)* **46** (1995), pp. 70–84.

[16] Campbell, S. L. & Griepentrog, E. "Solvability of General Differential Algebraic Equations". *SIAM Jour. Scientific Computation* **16** (1995), pp. 257–270.

[17] Campbell, S. L. & Holte, L. "Eigenvalue placement in completions of DAEs". *Electronic Journal of Linear Algebra* **26**.33 (2012), pp. 520–534.

[18] Campbell, S. L. & Kunkel, P. "Completions of Nonlinear DAE Flows Based on Index Reduction Techniques and their Stabilization". *Journal of Computational and Applied Mathematics* **233** (2009), pp. 1021–1034.

[19] Campbell, S. L. & Meyer Jr., C. D. *Generalized Inverses of Linear Transformations*. SIAM, 2011.

[20] Campbell, S. L. & Nikoukhah, R. *Auxiliary Signal Design for Failure Detection*. Princeton, New Jersey: Princeton University Press, 2004.

[21] Campbell, S. L. & Scott, J. R. "Asynchronous auxiliary signal design for failure detection". *Proc. 2014 IEEE International Conference on Systems, Man, and Cybernetics*. San Diego, CA, 2014.

[22] Castillo, I. et al. "Robust Model-Based Fault Detection and Isolation for Nonlinear Processes Using Sliding Modes". *International Journal of Robust and Nonlinear Control* **22** (2012), pp. 89–104.

[23] Chandola, V. et al. *Anomaly Detection: A Survey*. Tech. rep. Minneapolis, MN: Department of Computer Science and Engineering, University of Minnesota, 2007.

[24] Chen, J. et al. "Design of Unknown Input Observers and Robust Fault Detection Filters". *Int. Journal of Control* **63** (1996), pp. 85–105.

[25] Choe, D. et al. "A comparison of optimal and suboptimal auxiliary signal design approaches for robust failure detection". *Proc. IEEE Conference on Control Applications*. Toronto, 2005.

[26] Corradini, M. L. et al. "Robust FDI filters and fault sensitivity analysis in continuous-time descriptor systems". *Proc. 51st IEEE Conference on Decision and Control*. Maui, HI, 2012, pp. 1220–1225.

[27] De Luca, A. "Control properties of robot arms with joint elasticity". *Proc. International Symposium on the Mathematical Theory of Networks and Systems*. Phoenix, AZ, 1987.

[28] De Luca, A. and Isidori, A. "Feedback linearization of invertible systems". *2nd Duisburger Kolloquium Automation und Robotik*. Duisburg, Germany, 1987.

[29] Drake, K. "Analysis of Numerical Methods for Fault Detection and Model Identification in Linear Systems with Delays". PhD thesis. North Carolina State University, 2008.

[30] Duan, G. R. et al. “Robust Fault Detection in Descriptor Linear Systems via Generalized Unknown Input Observers”. *International Journal of Systems Science* **33**.5 (2010), pp. 369–377.

[31] Engelsone, A. et al. “Direct Transcription Solution of Higher-Index Optimal Control Problems and the Virtual Index”. *Applied Numerical Mathematics* **57** (2007), pp. 281–296.

[32] Gantmatcher, F. R. *The Theory of Matrices*. Vol. II. Providence, RI: AMS Chelsea, 1964.

[33] Gao, Z. & Wang, H. “Descriptor Observer Approaches for Multivariable Systems with Measurement Noises and Application in Fault Detection and Diagnosis”. *Systems and Control Letters* (2006), pp. 304–313.

[34] García, E. A. & Frank, P. M. “Deterministic Nonlinear Observer-based Approaches to Fault Diagnosis: a Survey”. *Control Engineering Practice* **5** (1997), pp. 663–670.

[35] Garg, D. et al. “A Unified Framework for the Numerical Solution of Optimal Control Problems Using Pseudospectral Methods”. *Automatica* **46**.11 (2010), pp. 1843–1851.

[36] Garg, D. et al. “Direct Trajectory Optimization and Costate Estimation of Finite-Horizon and Infinite-Horizon Optimal Control Problems via a Radau Pseudospectral Method”. *Computational Optimization and Applications* **49**.2 (2011), pp. 335–358.

[37] Garg, D. et al. “Pseudospectral Methods for Solving Infinite-Horizon Optimal Control Problems”. *Automatica* **47**.4 (2011), pp. 829–837.

[38] Geiger, G. “Monitoring of an electrical driven pump using continuous-time parameter estimation models”. *Proc. 6th IFAC Symposium on Identification and Parameter Estimation*. 1982.

[39] Gerdin, M. et al. “Parameter estimation in linear differential-algebraic equations”. *Proc. 13th IFAC Symposium on System Identification*. 2003.

[40] Gerdts, M. *Parameter Identification in Higher DAE Systems*. Tech. rep. Department of Mathematics, Universität Hamburg, 2005.

[41] Gertler, J. “Analytical redundancy methods in fault detection and isolation”. *Proc. IFAC SAFEPROCESS Symposium*. Baden-Baden, 1991.

[42] Isermann, R. “Process Fault Detection Based on Modeling and Estimation Methods: A Survey”. *Automatica* **20** (1984).

[43] Isermann, R. “Supervision, Fault-Detection and Fault Diagnosis Methods – an Introduction”. *Control Engineering Practice* **5**.5 (1997), pp. 639–652.

[44] Isermann, R. *Fault Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Berlin, Germany: Springer, 2006.

[45] Kircheis, R. & Körkel, S. "Parameter estimation for DAE models in a multiple experiment context". *Proc. 82nd Annual Meeting of the International Association of Applied Mathematics and Mechanics*. 2011.

[46] Kunkel, P. & Mehrmann, V. *Differential-Algebraic Equations. Analysis and Numerical Solution*. Zürich, Switzerland: EMS Publishing House, 2006.

[47] Lee, D. et al. "Robust H_∞ Sliding Mode Descriptor Oobserver for Fault and Output Disturbance Estimation of Uncertain Systems". *IEEE Transactions on Automatic Control* **57**.11 (2012), pp. 2928–2934.

[48] Leonhardt, S. & Ayoubi, M. "Methods of Fault Diagnosis". *Control Engineering Practice* **5**.5 (1997), pp. 683–692.

[49] März, R. "Notes on Linearization of DAEs and on Optimization with Differential-Algebraic Constraints". *Control and Optimiation with Differential-Algebraic Constraints*. Ed. by Biegler, L. T. et al. Advances in Design and Control. SIAM, 2012. Chap. 3, pp. 37–58.

[50] Niemann, H. H. "A Setup for Active Fault Diagnosis". *IEEE Transactions on Automatic Control* **51**.9 (2006), pp. 1572–1578.

[51] Niemann, H. H. "Active fault diagnosis in closed-loop uncertain systems". *6th IFAC Symposium on Fault Detection Supervision and Safety for Technical Processes*. Beijing, China, 2006, pp. 587–592.

[52] Nikoukhah, R. & Campbell, S. L. "Auxiliary Signal Design for Active Failure Detection in Uncertain Linear System with A Priori Information". *Automatica* **42**.2 (2006), pp. 219–228.

[53] Nikoukhah, R. & Campbell, S. L. "On the Detection of Small Parameter Variations in Linear Uncertain Systems". *European Journal of Control* **14**.2 (2008), pp. 158–171.

[54] Nikoukhah, R. et al. "Auxiliary Signal Design for Robust Multimodel Identification". *IEEE Transactions on Automatic Control* **47**.1 (2002), pp. 158–164.

[55] Okay, I. "The Additional Dynamics of the Least Squares Completions of Linear Differential Algebraic Equations". PhD thesis. North Carolina State University, 2008.

[56] Okay, I. et al. "The Additional Dynamics of Least Squares Completions for Linear Differential Algebraic Equations". *Linear Algebra and its Applications* **425** (2007), pp. 471–485.

[57] Okay, I. et al. "Completions of Implicitly Defined Linear Time Varying Vector Fields". *Linear Algebra and its Applications* **431** (2009), pp. 1422–1438.

[58] Patterson, M. A. & Rao, A. V. "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Continuous-Time Optimal Control Problems". *Journal of Spacecraft and Rockets* **49**.2 (2012), pp. 364–377.

[59] Patton, R. J. & Chen, J. "A review of parity space approaches to fault diagnosis". *Proc. IFAC SAFEPROCESS Symposium*. Baden-Baden, 1991.

[60] Poulsen, N. K. & Niemann, H. H. "Active fault diagnosis–a stochastic approach". *7th IFAC Symposium on Fault Detection Supervision and Safety for Technical Processes*. 2009.

[61] Raimondo, D. M. et al. "Active fault diagnosis using moving horizon input design". *Proc. European Control Conference*. Zürich, Switzerland, 2013.

[62] Rao, A. V. et al. "Algorithm 902: GPOPS, a Matlab Software for Solving Multiple-phase Optimal Control Problems using the Gauss Pseudospectral Method". *ACM Transactions on Mathematical Software* **37**.37 (2010), 22:1–22:39.

[63] Riaza, R. *Differential-Algebraic Systems. Analytical Aspects and Circuit Applications*. Hackensack, NJ: Word Scientific, 2008.

[64] Scott, J. R. & Campbell, S. L. "Observer based fault detection and identification in differential algebraic equations". *Proc. 2013 ASME Dynamical Systems and Control*. Palo Alto, CA, 2013.

[65] Scott, J. R. & Campbell, S. L. "Observer based fault detection in differential algebraic equations". *Proc. SIAM Conference on Control and its Applications*. San Diego, CA, 2013, pp. 176–183.

[66] Scott, J. R. & Campbell, S. L. "Auxiliary Signal Design for Failure Detection in Differential-Algebraic Equations". *Numerical Algebra, Control and Optimization* **4** (2014), pp. 151–179.

[67] Scott, J. R. & Campbell, S. L. "Auxiliary signal design for failure detection in high index differential-algebraic equations". *Proc. 2014 IEEE Conference on Decision and Control*. Los Angeles, CA, 2014.

[68] Sweetingham, K. "Auxiliary Signal Design for Fault Detection in Nonlinear Systems". PhD Thesis. Raleigh, NC: North Carolina State University, 2008.

[69] Varga, A. "Descriptor System Techniques in Solving $\mathcal{H}_{2/\infty}$-optimal Fault Detection and Isolation Problems". Ed. by Biegler, L. T. et al. Vol. 23. Advances in Design and Control. SIAM, 2012. Chap. 6, pp. 107–128.

[70] Wahrburg, A. & Adamy, J. "Fault isolation for linear non-minimum phase systems using dynamically extended observers". *Prc. 51st IEEE Conference on Decision and Control*. Maui, HI, 2012.

[71] Yeu, T. K. & Kawaji, S. "Sliding mode observer based fault detection and isolation in descriptor systems". *Proc. American Control Conference*. Anchorage, AK, 2002.

[72] Yeu, T. K. et al. "Fault Detection, Isolation and Reconstruction for Descriptor Systems". *Asian Journal of Control* **7**.4 (2005), pp. 356–367.

[73] Zemouche, A. & Boutayeb, M. "Observers for continuous-time lipschitz nonlinear systems. Analysis and comparisions". *Proc. 51st IEEE Conference on Decision and Control*. Maui, HI, 2012, pp. 4774–4779.

APPENDIX

Appendix A

MATLAB Code

A.1 Section 4.4 Code

Explanation of code listings:

- paper3exec.m - The main file that should be executed to produce the results of Section 4.4.
- Build_Matrices_Reduction.m - Performs the reduction procedure in Section 4.1.2.
- fullrow_to_invertible.m - Given $M = [A\ B]$ where M has full row rank and A has full column rank, this function returns columns of B that make A concatenated with the extra columns invertible. This is used during the reduction procedure.
- Find_properu.m- Finds a minimal proper u using the L^2 noise bound. Solves Problem 4.1.2.
- OPTu_checker.m - Returns $\min \|y_0 - y_1\|^2$ given a signal u. Solves Problem 4.3.1.
- Model_ID.m - Tests the model identification algorithm. This function takes a proper u and its time grid t and calculates the minimum noise necessary to produce an output y. The output y would usually be measured. In the absence of actual measurements, we randomly generate noises and generate an output for model i. This function returns the results shown in Section 4.4.2.
- scaleu_alg1.m- Scales u down to make it closer to minimal proper. It is the implementation of Algorithm 4.1.

```
%This is the main file that should be executed to produce the results
%of Section 4.4.

% Explanation of variables created:
% u_opt - Optimal signal
% t_opt - time grid optimal signal is defined on
% obj   - ||u||^2 (in L2)

clear all
close all
clc


% load CampbellNik2004_t1.mat %Example 1 (ODE Example: Proper u only no ID)
 load Paper3_DAE_ex.mat %Example 2

 %build matrices and do reduction procedure
auxdata = Build_Matrices_Reduction(matrices,params);

%Section 4.4.1
% 1) Find the proper u, the time grid its defined on
[u_opt,t_opt,obj]=Find_properu(auxdata);

figure
plot(t_opt,u_opt,'-rd','LineWidth',2)
xlabel('Time','FontSize',14)
ylabel('u','FontSize',14)
ax = gca;
set(ax,'FontSize',14)


%Proper u check - Finds J =||y_0 - y_1||^2
%select bound type (L2 or infty)
bound_type = 'L2';

J = [];
for alpha = [.6 .9 .99 1.01 1.1 1.8]
temp = OPTu_checker( t_opt,u_opt,matrices,params,bound_type, alpha);
J = [J temp];
end
% end section 4.4.1

```

```
%Real-Time Identification Section 4.4.2
u_opt = sqrt(2)*u_opt; %make u proper for L_infty

%model on which the output is based (the active model in the simulation)
true_model = 1;

%run a function to calculate W_i, the past cost
%on the noise for both models
[W_hist,t] = Model_ID(u_opt,t_opt,auxdata,true_model);

%plot ID results
tspan = linspace(params.t0,params.tf);
gamma_plot = params.gamma*ones(100,1);
figure()
plot(t,W_hist(:,1),'k--',t,W_hist(:,2),'k:',tspan,gamma_plot,'LineWidth',2)
xlabel('Time','FontSize',14)
ax = gca;
set(ax,'FontSize',14)
legend('W_0','W_1','\gamma')
```

```
function [auxdata] = Build_Matrices_Reduction(matrices,params)
%
% function [auxdata] = Build_Matrices_Reduction(matrices,params)
% performs the reduction procedure in Section 4.1.2 and builds matrices
% necessary to solve Problem 4.1.2
%
% Input:
% matrices - matrices that define the problem for both models, i = 0 and i
% =1.
% params - parameters for the problem:
%   t_0, t_f = initial and final time of the test interval
%    gamma   = noise bound
%


[n0,unused] = size(matrices.E0);
[n1,unused] = size(matrices.E1);
r0 = rank(matrices.E0);
r1 = rank(matrices.E1);
[m0,unused] = size(matrices.C0);
[m1,unused] = size(matrices.C1);
```

```
22
23
24 params.n0=n0;
25 params.n1=n1;
26 params.r0=r0;
27 params.r1=r1;
28 params.m0=m0;
29 params.m1=m1;
30
31 %-------------------------------------------------------------------------%
32 % Perform Reduction
33 %-------------------------------------------------------------------------%
34
35 % compute SVD of E_i
36 [U0, Sigma0, V0] = svd(matrices.E0);
37 [U1, Sigma1, V1] = svd(matrices.E1);
38 Sigmatilde0 = [inv(Sigma0(1:r0,1:r0)) zeros(r0,n0-r0); zeros(n0-r0,r0) ...
39     eye(n0-r0,n0-r0) ];
40 Sigmatilde1 = [inv(Sigma1(1:r1,1:r1)) zeros(r1,n1-r1); zeros(n1-r1,r1) ...
41     eye(n1-r1,n1-r1) ];
42
43 % ----------  Perform change of variables on C matrices and break up-----%
44 C0 = matrices.C0*V0;
45 C1 = matrices.C1*V1;
46 C01 = C0(:,1:r0);
47 C02 = C0(:,r0+1:n0);
48 C11 = C1(:,1:r1);
49 C12 = C1(:,r1+1:n1);
50 %-------------------------------------------------------------------------%
51
52 %Create A_ijk matrices
53 tempA0 = -Sigmatilde0*U0'*matrices.F0*V0;
54 A011 = tempA0(1:r0,1:r0);
55 A012 = tempA0(1:r0,r0+1:n0);
56 A021 = tempA0(r0+1:n0,1:r0); A022 = tempA0(r0+1:n0,r0+1:n0);
57
58 tempA1 = -Sigmatilde1*U1'*matrices.F1*V1;
59 A111 = tempA1(1:r1,1:r1);
60 A112 = tempA1(1:r1,r1+1:n1);
61 A121 = tempA1(r1+1:n1,1:r1); A122 = tempA1(r1+1:n1,r1+1:n1);
62
63
64 %Create M_ijk matrices
```

```
tempM0 = Sigmatilde0*U0'*matrices.M0;
tempM1 = Sigmatilde1*U1'*matrices.M1;
[unused1,unused2,Vm0]=svd(tempM0(r0+1:r0+n0-r0,:));
[unused,unused,Vm1]=svd(tempM1(r1+1:r1+n1-r1,:));
perm0 = [zeros(n0-r0,r0) eye(n0-r0); eye(r0) zeros(r0,n0-r0)];
perm1 = [zeros(n1-r1,r1) eye(n1-r1); eye(r1) zeros(r1,n1-r1)];
tempM0 = tempM0*Vm0;
tempM0 = tempM0*perm0;
tempM1 = tempM1*Vm1*perm1;
M011 = tempM0(1:r0,1:r0);
M012= tempM0(1:r0,r0+1:n0);
M022 = tempM0(r0+1:n0,r0+1:n0);
M111 = tempM1(1:r1,1:r1); M112= tempM1(1:r1,r1+1:n1);
M122 = tempM1(r1+1:n1,r1+1:n1);

%Create B_ij matrices
B0 = Sigmatilde0*U0'*matrices.B0;
B1 = Sigmatilde1*U1'*matrices.B1;
B01 = B0(1:r0,:);
B02 = B0(r0+1:n0,:);
B11 = B1(1:r1,:);
B12 = B1(r1+1:n1,:);

%make hat matrices and tilde matrices in 4.13
Ahat = [A011 zeros(r0,r1); zeros(r1,r0) A111];
Dhat = [A012 zeros(r0,n1-r1); zeros(r1,n0-r0) A112];
Nhat = [M012 zeros(r0,n1-r1+m0); zeros(r1,n0-r0) M112 zeros(r1,m0)];
Bhat = [B01;B11];
Mhat = [M011 zeros(r0,r1+m1) ; zeros(r1,r0) M111 zeros(r1,m1)];
Atilde = [ A021 zeros(n0-r0,r1); zeros(n1-r1,r0) A121; C01 -C11];
Dtilde = [A022 zeros(n0-r0,n1-r1); zeros(n1-r1,n0-r0) A122; C02 -C12];
Btilde = [B02;B12;matrices.D0-matrices.D1];
Mtilde = [zeros(n0-r0+n1-r1,r0+r1+m1); zeros(m1,r0+r1) -matrices.N1];
Ntilde = blkdiag(M022,M122,matrices.N0);


%------- Pick subset of columns of Ntilde so that [Dtilde Ntilde] --------%
%--- is invertible.  This subset exists because [Dtilde Ntilde] ----------%
% -------- has full row rank. ----------------------------------------%
[H,PermH] = fullrow_to_invertible(Dtilde, Ntilde);
%-------------------------------------------------------------------------%
Ntilde = Ntilde*PermH;
Nhat = Nhat*PermH;
```

```
Atilde = [Dtilde H]\Atilde;
Btilde = [Dtilde H]\Btilde;
Mtilde = [Dtilde H]\Mtilde;
Gtilde = [Dtilde H]\Ntilde(:,m0+1:end);


H2 = Nhat(:,1:m0);
Rhat = [Dhat H2];
Ghat = Nhat(:,m0+1:end);

A = Ahat - Rhat*Atilde;
B = Bhat-Rhat*Btilde;
M = Mhat - Rhat*Mtilde;
G = Ghat - Rhat*Gtilde;


A0 = Atilde(n0-r0+n1-r1+1:end,:);
G0 = Gtilde(n0-r0+n1-r1+1:end,:);
B0 = Btilde(n0-r0+n1-r1+1:end,:);
M0 = Mtilde(n0-r0+n1-r1+1:end,:);

Q = [eye(length(G0(1,:)),length(G0(1,:)))+G0'*G0 G0'*M0; ...
    M0'*G0 eye(length(M0(1,:)),length(M0(1,:)))+M0'*M0];


%system matrices from 4.17
Az1 = A-[G M]*(Q\[G0'*A0; M0'*A0]);
Au = B - [G M]*(Q\[G0'*B0; M0'*B0]);
Alam = -[G M]*(Q\[G';M']);
Bz1 = A0'*A0 - [A0'*G0 A0'*M0]*(Q\[G0'*A0; M0'*A0]);
Blam = A'- [A0'*G0 A0'*M0]*(Q\[G';M']);
Bu = A0'*B0 - [A0'*G0 A0'*M0]*(Q\[G0'*B0;M0'*B0]);
%----------------- End reduction ----------------------------------------%
%store system matrices for dynamic equations when finding proper u
auxdata.Az1 = Az1;
auxdata.Au = Au;
auxdata.Alam = Alam;
auxdata.Bz1 = Bz1;
auxdata.Blam = Blam;
auxdata.Bu = Bu;
%------------------------------------------------------------------------%

%------------------------------------------------------------------------%
```

```
% some matrices are needed to compute I in the DE for \psi
auxdata.inv_noise = Q;
auxdata.G = G;
auxdata.M = M;
auxdata.G0 = G0;
auxdata.M0 = M0;
auxdata.A0 = A0;
auxdata.B0 = B0;
%-------------------------------------------------------------------------&
auxdata.P = matrices.P;
%-------------------------------------------------------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%END 4.1 MATRICES


%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                     matrices for ID test  Sec. 4.2
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%

% model 0
Dtilde0= [A022 ; C02];
Ntilde0 = [zeros(m0+n0-r0,r0) blkdiag(M022,matrices.N0)];
Btilde0 = [B02;matrices.D0];
Atilde0 = [A021;C01];
Nhat0 = [M011 M012 zeros(r0,m0)];
Bhat0 = B01;
A11 = A011;
Dhat0 = A012;


[H,PermH] = fullrow_to_invertible(Dtilde, Ntilde);
%-------------------------------------------------------------------------%
Ntilde = Ntilde*PermH;
Nhat = Nhat*PermH;
Atilde = [Dtilde H]\Atilde;
Btilde = [Dtilde H]\Btilde;
Mtilde = [Dtilde H]\Mtilde;
Gtilde = [Dtilde H]\Ntilde(:,m0+1:end);


H2 = Nhat(:,1:m0);
```

```
194 Rhat = [Dhat H2];
195 Ghat = Nhat(:,m0+1:end);
196
197
198 [Ntilde01, PermN]= fullrow_to_invertible(Dtilde0,Ntilde0);
199 Ntilde0 = Ntilde0*PermN;
200 Ntilde02 = Ntilde0(:,m0+1:end);
201 Nhat0 = Nhat0*PermN;
202 Nhat01 = Nhat0(:,1:m0);
203 Nhat02 = Nhat0(:,m0+1:end);
204 Rhat0 = [Dhat0 Nhat01];
205
206 Q1 = -[Dtilde0 Ntilde01]\Ntilde02;
207 Q2 = -[Dtilde0 Ntilde01]\Atilde0;
208
209 %-------------------------------------------------------------------------%
210 % define matrices needed to solve DE for K,v,phi
211 auxdata.model0.A = A11- Rhat0*([Dtilde0 Ntilde01]\Atilde0);
212 auxdata.model0.N = Nhat02 - Rhat0*([Dtilde0 Ntilde01]\Ntilde02);
213 auxdata.model0.Bhat0 = Bhat0;
214 auxdata.model0.Btilde0 = Btilde0;
215 auxdata.model0.Rhat0 = Rhat0;
216 auxdata.model0.Dtilde0 = Dtilde0;
217 auxdata.model0.Ntilde01 = Ntilde01;
218 auxdata.model0.Q1 = Q1(n0-r0+1:end,:);
219 auxdata.model0.Q2 = Q2(n0-r0+1:end,:);
220 auxdata.model0.Q = auxdata.model0.Q1'*auxdata.model0.Q1+ ...
221     eye(size(auxdata.model0.Q1'*auxdata.model0.Q1));
222 auxdata.model0.hatP = matrices.P0(1:r0,1:r0);
223 %end model 0
224
225
226 %-------------------------------------------------------------------------%
227 % model 1
228 Dtilde0= [A122 ; C12];
229 Ntilde0 = [zeros(m1+n1-r1,r1) blkdiag(M122,matrices.N1)];
230 Btilde0 = [B12;matrices.D1];
231 Atilde0 = [A121;C11];
232 Nhat0 = [M111 M112 zeros(r1,m1)];
233 Bhat0 = B11;
234 A11 = A111;
235 Dhat0 = A112;
236
```

```
237 [Ntilde01, PermN]= fullrow_to_invertible(Dtilde0,Ntilde0);
238 Ntilde0 = Ntilde0*PermN;
239 Ntilde02 = Ntilde0(:,m1+1:end);
240 Nhat0 = Nhat0*PermN;
241 Nhat01 = Nhat0(:,1:m1);
242 Nhat02 = Nhat0(:,m1+1:end);
243 Rhat0 = [Dhat0 Nhat01];
244
245 Q1 = -[Dtilde0 Ntilde01]\Ntilde02;
246 Q2 = -[Dtilde0 Ntilde01]\Atilde0;
247
248 %-------------------------------------------------------------------------%
249 % define matrices needed to solve DE for K,v,phi
250 auxdata.model1.A = A11- Rhat0*([Dtilde0 Ntilde01]\Atilde0);
251 auxdata.model1.N = Nhat02 - Rhat0*([Dtilde0 Ntilde01]\Ntilde02);
252 auxdata.model1.Bhat0 = Bhat0;
253 auxdata.model1.Btilde0 = Btilde0;
254 auxdata.model1.Rhat0 = Rhat0;
255 auxdata.model1.Dtilde0 = Dtilde0;
256 auxdata.model1.Ntilde01 = Ntilde01;
257 auxdata.model1.Q1 = Q1(n1-r1+1:end,:);
258 auxdata.model1.Q2 = Q2(n1-r1+1:end,:);
259 auxdata.model1.Q = auxdata.model1.Q1'*auxdata.model1.Q1+...
260     eye(size(auxdata.model1.Q1'*auxdata.model1.Q1));
261 auxdata.model1.hatP = matrices.P1(1:r1,1:r1);
262 %end model 1
263
264 %-------------------------------------------------------------------------%
265 % grab matrices/parameters too
266 auxdata.params =  params;
267 auxdata.matrices = matrices;
268 %-------------------------------------------------------------------------%
269 %END ID matrices
270 end
```

```
1 function [ cols, P] = fullrow_to_invertible( A,B )
2 %
3 % function [ cols, P] = fullrow_to_invertible(A,B)
4 %   This function takes Dtilde (A) and Ntilde (B) and returns cols
5 % N_1tilde  and a perumutation matrix P that reorders Ntilde so that its
6 % first columns are N_1tilde
```

```
%INPUT:
%   A - a matrix with full column rank (Dtilde from section 4.1)
%   B - a matrix so that [A B] has full row rank (Ntilde from section 4.1)
%
%OUTPUT:
  % cols - The first n-m columns of B that make [A cols] an invertible matrix.
  % P - the permutation matrix so that B*P orders the columns so that the
  % first n-m columns of B*P are cols.

cols =[];
[n,m]=size(A); % we assume n>m
[m2,n2] =size(B);
I= [];
P = zeros(n2);

%the rank of A is j.  Need to add columns to A until
%rank(A) = n
for j = m+1:n
    i=1;

%if the next column of B does not add a rank,
    %then skip it
while rank([A B(:,i)]) < j
    i = i + 1;
end
%if it does add a rank, then tack it onto cols
        A = [A B(:,i)];
        cols = [cols B(:,i)];
        P(i,j-m)=1; %record which column it was in the permutation matrix
        I = [I i];
end

%finish making P
for j = n-m+1:n2
    for i = 1:n2
        if max(i==I) == 1
        else
            P(i,j) = 1;
            I = [I i];
            break;
        end
    end
end
```

```
end
```

```
function [u_opt,t_opt, obj] = Find_properu(auxdata)
%
% function [ u_opt, t_opt, obj ] = Find_properu(auxdata)
%
% Find_properu finds a minimal proper u using the L^2 noise for a problem
% defined by auxdata (see Build_Matrices_Reduction.m for an explanation).
% INPUT:
%       auxdata - problem structure

% Ouput =
%            u_opt - minimal proper u that is defined on
%            t_opt - the time grid for u
%            obj  - ||u||^ 2 (in L2)
%
% This function uses the helper functions properu_cont.m,
% properu_end.m, properu_settings.m as needed by GPOPSII


global r0 % dimension of dynamic states in model 0
global r1 % dimension of dynamic states in model 1
global n0 % dimension of states in model 0
global n1 % dimension of states in model 1
global m0 % dimension of outputs in model 0
global m1 % dimension of outputs in model 0

% Prepare auxdata, settings and parameters to pass to GpopsII
%---------------Get all matrices and parameters -------------------------%
params = auxdata.params;
matrices = auxdata.matrices;
r0 = auxdata.params.r0;
r1 = params.r1;
n0 = params.n0;
n1 = params.n1;
m0 = params.m0;
m1 = params.m1;

%Compute all other quantities from user given params
auxdata.params = params;
auxdata.matrices = matrices;
```

```
40
41
42 % Get settings for GPOPSII%
43 setup = properu_settings(params,auxdata.P);
44 setup.auxdata=auxdata;
45 % ------------------------------------------------------------ %
46
47 % Solve with GPOPSII
48 output = gpops2(setup);
49
50 % gather output
51 t_opt = output.result.solution.phase.time;
52 u_opt = output.result.solution.phase.control;
53
54 %the states for model 0 and model 1
55 z1 = output.result.solution.phase.state(:,1:r0+r1);
56 lambda = output.result.solution.phase.state(:,r0+r1+1:end-1);
57 obj = output.result.objective;
58 end
59 %-------------------------------------------------------------------------%
60
61
62
63 % Helper functions
64
65
66
67 function phaseout = properu_cont(input)
68 global r0
69 global r1
70 global n0
71 global n1
72 t = input.phase.time;
73 x = input.phase.state;
74 u = input.phase.control;
75
76 %-------------------------------------------------------------------------%
77 %---------------------- Define needed Params -----------------------------%
78 %-------------------------------------------------------------------------%
79 G0= input.auxdata.G0;
80 M0=input.auxdata.M0;
81 G = input.auxdata.G;
82 M= input.auxdata.M;
```

```
A0 = input.auxdata.A0;
B0 = input.auxdata.B0;
inv_noise = input.auxdata.inv_noise;
Az1 = input.auxdata.Az1;
Alam = input.auxdata.Alam;
Au = input.auxdata.Au;
Bz1 = input.auxdata.Bz1;
Blam = input.auxdata.Blam;
Bu = input.auxdata.Bu;
%--------------------------------------------------------------------%
%-------------- Dynamics ----------------------------------------%
%--------------------------------------------------------------------%
%need to include dynamics \psi dot = I() as well
z1 = x(:,1:r0+r1);
lambda = x(:,r0+r1+1:end-1);

dz1 = Az1*z1' + Au*u' + Alam*lambda';
dlam = -Bz1 *z1' - Blam*lambda' - Bu*u';
c12c2 = -inv_noise\([G';M']*lambda'+[G0'*A0; M0'*A0]*z1' + [G0'*B0; M0'*B0]*u');
c11 = -(A0*z1'+[G0 M0]*c12c2 + B0*u');
c = [c11;c12c2];
dpsi = 1/2*(sum(c.^2,1));
integrand = sum(u'.^2,1)';

xdot=[dz1' dlam' dpsi'];


phaseout.dynamics = xdot;
phaseout.integrand = integrand;
end
%------------------------------------------------------------- %

function output = properu_end(input)
global r0
global r1
P = input.auxdata.P;
q  = input.phase.integral;
%-------------- Input Linear BC ---------------------------------------%

output.objective = q;

x0 = input.phase.initialstate;
xf = input.phase.finalstate;
```

```
126 output.eventgroup.event = [(P*x0(1:r0+r1)'+x0(r0+r1+1:end-1)')', ...
127     x0(end)-1/2*x0(1:r0+r1)*P*x0(1:r0+r1)'];
128 end
129 %-------------------------------------------------------------- %
130
131 function [ setup ] =properu_settings(params,P)
132 %function [setup] = Define_GPOPS_settings(params,P)
133 %
134 % This function defines the settings to pass into GPOPSII.
135 %
136 %Input:
137 % params.t0 = initial time
138 % params.tf = final time
139 % params.gamma = noise bound
140 % P  - initial uncertainty matrix
141 %
142 %
143 % Output;
144 % setup - settings to pass to gpopsII
145
146 global r1
147 global r0
148 global n0
149 global n1
150 %-------------------------------------------------------------------------%
151 %---------------------- Setup for Problem Bounds -------------------------%
152 %-------------------------------------------------------------------------%
153 bounds.phase.initialtime.lower = params.t0;
154 bounds.phase.initialtime.upper = params.t0;
155 bounds.phase.finaltime.lower = params.tf;
156 bounds.phase.finaltime.upper = params.tf;
157
158     n=2*(r1+r0);
159 bounds.phase.initialstate.lower = [-1e8*ones(1,n) 0];
160 bounds.phase.initialstate.upper = [1e8*ones(1,n) 1e8];
161 bounds.phase.state.lower = [-1e8*ones(1,n) 0];
162 bounds.phase.state.upper = [1e8*ones(1,n) 10000];
163 bounds.phase.finalstate.lower = [-1e8*ones(1,r0+r1) zeros(1,r0+r1) ...
164     params.gamma^2];
165 bounds.phase.finalstate.upper = [1e8*ones(1,r0+r1) zeros(1,r0+r1) 1e9];
166 bounds.eventgroup.lower              = zeros(1,r0+r1+1);
167 bounds.eventgroup.upper              = zeros(1,r0+r1+1);
168
```

```
bounds.phase.control.lower = -1e9;
bounds.phase.control.upper = 1e9;
bounds.phase.integral.lower = 0;
bounds.phase.integral.upper = 1e10;



%------------------------------------------------------------------------%
%---------------------- Provide Guess of Solution -----------------------%
%------------------------------------------------------------------------%
    guessx = [1*ones(r0+r1+r0+r1,1)' 1/2*ones(r0+r1,1)'*P*ones(r0+r1,1);...
                ones(r0+r1+r0+r1,1)' 100];
    guess.phase.time     = [params.t0;params.tf];
    guess.phase.state    = guessx;
    guess.phase.control  = [1;1];
    guess.phase.integral = 1;



%------------------------------------------------------------------------%
%----------Provide Mesh Refinement Method and Initial Mesh ---------------%
%
%optional fields
mesh.method          = 'hp1';
mesh.tolerance       = 1e-3;
mesh.maxiteration    = 10;
mesh.colpointsmin    = 4;
mesh.colpointsmax    = 12;
mesh.phase.colpoints = 4*ones(1,10);
mesh.phase.fraction  = 0.1*ones(1,10);
%}
%------------------------------------------------------------------------%
%------------- Assemble Information into Problem Structure --------------%
%------------------------------------------------------------------------%
setup.name = 'Find minimal proper u';
setup.functions.continuous = @properu_cont;
setup.functions.endpoint = @properu_end;
setup.bounds = bounds;
setup.guess = guess;
setup.mesh = mesh;
setup.nlp.solver = 'ipopt';
setup.nlp.options.ipopt.linear_solver = 'ma57';
```

```
%setup.derivatives.supplier = 'sparseBD';
setup.derivatives.derivativelevel = 'second';
%setup.scales.method = 'automatic-bounds';
setup.method = 'RPMintegration';


end
```

```
function [ J ] = OPTu_checker( t_opt,u_opt,matrices,params,bound_type, alpha)
%function [J] = OPTu_checker(t_opt,u_opt,matrices,params,bound_type,alpha)
%
% This function
% INPUT:
% u_opt - optimal control (minimal proper)u
% t_opt - time grid for u_opt
% matrices - matrices that define the problem
% params -
    % t0 - start time
    % tf - end time
    % gamma - bound for noise \Gamma_{infty}<\gamma \Gamma{L^2} <
    %         \gamma
% bound_type - 'infty', 'L2'  infinity or L^2 bound for noise
% alpha - scaling factor on u
%
% OUTPUT:
% J = min || y_0 - y_1||^2


%error handling
if strcmp(bound_type,'infty')==0 && strcmp(bound_type,'L2')==0
    error('Error.  bound_type must either be "infty" or "L2"')
end
J = [];


auxdata = OPTu_checker_auxdata(matrices,params);
auxdata.u=u_opt;
auxdata.t = t_opt;
auxdata.bound_type = bound_type;
setup = OPTu_checker_settings(auxdata,params,bound_type);

```

```
auxdata.params = params;
auxdata.alpha = alpha;
setup.auxdata = auxdata;
output = gpops2(setup);
J = [J output.result.objective];
end


function [ setup ] = OPTu_checker_settings(auxdata,params,bound_type)
% This function defines the settings to pass into GPOPSII.

n0 =auxdata.n0;
n1 =auxdata.n1;
r0 =auxdata.r0;
r1 =auxdata.r1;
m0 =auxdata.m0;
m1 = auxdata.m1;
bounds.phase.initialtime.lower = params.t0;
bounds.phase.initialtime.upper = params.t0;
bounds.phase.finaltime.lower = params.tf;
bounds.phase.finaltime.upper = params.tf;
    if strcmp(bound_type,'L2')==1
     bounds.phase.initialstate.upper = [1e12*ones(1,r0+r1) params.gamma^2];
     bounds.phase.initialstate.lower = [-1e12*ones(1,r0+r1) 0];
     bounds.phase.state.lower = [-1e12*ones(1,r0+r1) 0];
     bounds.phase.state.upper = [1e12*ones(1,r0+r1) params.gamma^2];
     bounds.phase.finalstate.lower = [-1e12*ones(1,r0+r1) 0];
     bounds.phase.finalstate.upper = [1e12*ones(1,r0+r1) params.gamma^2];
     bounds.eventgroup.lower              = 0;
     bounds.eventgroup.upper              = 0;
    elseif strcmp(bound_type,'infty') ==1
     bounds.phase.initialstate.upper = [1e12*ones(1,r0+r1) ...
                                        params.gamma^2 params.gamma^2];
     bounds.phase.initialstate.lower = [-1e12*ones(1,r0+r1) 0 0];
     bounds.phase.state.lower = [-1e12*ones(1,r0+r1) 0 0];
     bounds.phase.state.upper = [1e12*ones(1,r0+r1) params.gamma^2 ...
                                    params.gamma^2];
     bounds.phase.finalstate.lower = [-1e12*ones(1,r0+r1) 0 0];
     bounds.phase.finalstate.upper = [1e12*ones(1,r0+r1) params.gamma^2 ...
                                        params.gamma^2];
     bounds.eventgroup.lower              = [0 0];
     bounds.eventgroup.upper              = [0 0];
    end
```

```
77  bounds.phase.control.lower = [-1e10*ones(1,n0-r0+n1-r1) ...
78                                -200*params.gamma^2*ones(1,r0+r1+m1+m0)];
79  bounds.phase.control.upper = [1e10*ones(1,n0-r0+n1-r1) ...
80                                200*params.gamma^2*ones(1,r0+r1+m1+m0)];
81  bounds.phase.integral.lower = 0;
82  bounds.phase.integral.upper = 1e12;
83
84
85
86  %------------------------------------------------------------------------%
87  %---------------------- Provide Guess of Solution -----------------------%
88  %------------------------------------------------------------------------%
89
90      if strcmp(bound_type,'L2')==1
91          guessx = [2*ones(r0+r1,1)' params.gamma^2/2; ...
92                      ones(r0+r1,1)' params.gamma^2/2 ];
93      elseif strcmp(bound_type,'infty')==1
94          guessx = [2*ones(r0+r1,1)' params.gamma^2/2 params.gamma^2/2; ...
95                      ones(r0+r1,1)' params.gamma^2/2 params.gamma^2/2 ];
96      end
97      guess.phase.control  = [1*ones(1,n0-r0+n1-r1) ...
98                               .01*ones(1,r0+r1+m1+m0); ...
99                              1*ones(1,n0-r0+n1-r1) .01*ones(r0+r1+m1+m0,1)'];
100 guess.phase.time     = [params.t0; params.tf];
101 guess.phase.state    = guessx;
102 guess.phase.integral = 1;
103 %------------------------------------------------------------------------%
104 %----------Provide Mesh Refinement Method and Initial Mesh ---------------%
105 %
106 %optional fields
107 mesh.method          = 'hp1';
108 mesh.tolerance       = 1e-3;
109 mesh.maxiteration    = 5;
110 mesh.colpointsmin    = 4;
111 mesh.colpointsmax    = 16;
112 mesh.phase.colpoints = 4*ones(1,10);
113 mesh.phase.fraction  = 0.1*ones(1,10);
114 %}
115 %------------------------------------------------------------------------%
116 %------------- Assemble Information into Problem Structure --------------%
117 %------------------------------------------------------------------------%
118 setup.name = 'fix u min J';
119 setup.functions.continuous = @OPTu_checker_cont;
```

```
120 setup.functions.endpoint = @OPTu_checker_end;
121 setup.bounds = bounds;
122 setup.guess = guess;
123 setup.mesh = mesh;
124 setup.nlp.solver = 'ipopt';
125 %setup.nlp.options.ipopt.linear_solver = 'ma57';
126 setup.derivatives.supplier = 'sparseCD';
127 setup.derivatives.derivativelevel = 'second';
128 setup.method = 'RPMintegration';
129
130
131 end
132
133 %-------------------------------------%
134 % BEGIN:fixu_minnoiseContinuous.m %
135 %-------------------------------------%
136 function phaseout = OPTu_checker_cont(input)
137
138 alpha = input.auxdata.alpha;
139 u_opt = input.auxdata.u;
140 t_opt = input.auxdata.t;
141
142 r1=input.auxdata.r1; %rank of E1
143 r0 = input.auxdata.r0; %rank of E0
144 n0 = input.auxdata.n0; % size of x0
145 n1 = input.auxdata.n1; % size of x1
146 m0 = input.auxdata.m0; %number of outputs model 0
147 m1 = input.auxdata.m1; % number of outputs model 1
148
149 t = input.phase.time;
150 %------------ get state separated ---------%
151 x = input.phase.state;
152 z1 = x(:,1:r0+r1);
153 % ----------------- get control separated------------%
154 noise = input.phase.control;
155 z2 = noise(:,1:n0-r0+n1-r1);
156 zeta2 = noise(:,n0-r0+n1-r1+1:n0+n1+m1);
157 eta1 = zeta2(:,r0+r1+1:end);
158 eta0 = noise(:,n0+n1+m1+1:n0+n1+m1+m0);
159 %-------------------------------------------------------------------------%
160
161 %--------------------------recover needed matrices -------------------%
162 M = input.auxdata.M;
```

```
163 G = input.auxdata.G;
164 B = input.auxdata.B;
165 A = input.auxdata.A;
166 N0 = input.auxdata.N0;
167 N1 = input.auxdata.N1;
168 D0 = input.auxdata.D0;
169 D1 = input.auxdata.D1;
170 C = input.auxdata.C;
171 Atilde = input.auxdata.Atilde;
172 Btilde = input.auxdata.Btilde;
173 Dtilde = input.auxdata.Dtilde;
174 %--------------------------------------------------------------------------%
175 %---------------------------------------------------------------------%
176 %--------------- Dynamics ----------------------------------------%
177 %---------------------------------------------------------------------%
178 %GPOPS sometimes passes time values < t0 and > tf but only by tf+\epsilon
179 % and t0-\epsilon where \epsilon is very small.  Since we don't have values
180 % of u there, we set these small time perturbations to t0 and tf
181 tu  = (t>=input.auxdata.params.t0).*t.*(t<=input.auxdata.params.tf);
182
183 %interpolate u to get values at the time points needed by GPOPS
184 u = alpha*interp1(t_opt,u_opt,tu);
185
186 [pts, n] = size(x);
187 xdot = zeros(pts,n);
188 integrand = zeros(pts,1);
189
190 %dynamics
191 dz1 = (A*z1' + B*u' + G*z2' + M*zeta2')';
192 mu02m12 = (-Atilde*z1' - Dtilde*z2' - Btilde*u')';
193 if strcmp(input.auxdata.bound_type,'L2')==1
194     dpsi = (1/2*(sum(eta0'.^2,1) + sum(zeta2'.^2,1)+sum(mu02m12'.^2,1)))';
195 elseif strcmp(input.auxdata.bound_type,'infty')==1
196    dpsi = [(1/2*(sum(eta0'.^2,1)+sum(zeta2(:,1:r0)'.^2,1)+ ...
197              sum(mu02m12(:,1:(n0-r0))'.^2,1)))', ...
198          (1/2*(sum(zeta2(:,r0+1:end)'.^2,1)+ ...
199          sum(mu02m12(:,n0-r0+1:end,:)'.^2,1)))'];
200 end
201 xdot = [dz1 dpsi];
202 phaseout.dynamics = xdot;
203
204 %cost integrand
205 integrand = C*[z1';z2'] + [D0 -D1]*[u';u'] + [N0 -N1]*[eta0';eta1'];
```

```
integrand = sum(integrand.^2,1);
phaseout.integrand =  integrand';
end


function [auxdata] = OPTu_checker_auxdata( input,params )
% This function accepts the user inputs outputs all needed
% matrices and parameters in auxdata.  This function is very much like
% Build_Matrices_Reduction.m.

[n0,unused] = size(input.E0); %E_i is square by assumption
[n1,unused] = size(input.E1);
r0 = rank(input.E0);
r1 = rank(input.E1);
n = r0+r1+1;
[U0, Sigma0, V0] = svd(input.E0);
[U1, Sigma1, V1] = svd(input.E1);

[m0,unused] = size(input.C0);
[m1,unused] = size(input.C1);
Sigmatilde0 = [inv(Sigma0(1:r0,1:r0)) zeros(r0,n0-r0); ...
                zeros(n0-r0,r0) eye(n0-r0,n0-r0) ];
Sigmatilde1 = [inv(Sigma1(1:r1,1:r1)) zeros(r1,n1-r1); ...
                zeros(n1-r1,r1) eye(n1-r1,n1-r1) ];

% ----------  Perform change of variables on C matrices and break up-----%
C0 = input.C0*V0;
C1 = input.C1*V1;
I1 = [eye(r0) zeros(r0,r1+n0-r0+n1-r1); zeros(r1,r0+n0-r0) eye(r1) ...
    zeros(r1,n1-r1); zeros(n0-r0,r0) eye(n0-r0) zeros(n0-r0,r1+n1-r1); ...
            zeros(n1-r1,r0+n0-r0+r1) eye(n1-r1)];
C = [C0 -C1]*I1;
%-------------------------------------------------------------------------%


tempA0 = -Sigmatilde0*U0'*input.F0*V0;
A011 = tempA0(1:r0,1:r0); A012 = tempA0(1:r0,r0+1:n0);
A021 = tempA0(r0+1:n0,1:r0); A022 = tempA0(r0+1:n0,r0+1:n0);


tempA1 = -Sigmatilde1*U1'*input.F1*V1;
A111 = tempA1(1:r1,1:r1); A112 = tempA1(1:r1,r1+1:n1);
A121 = tempA1(r1+1:n1,1:r1); A122 = tempA1(r1+1:n1,r1+1:n1);
```

```
249
250 tempM0 = Sigmatilde0*U0'*input.M0;
251 tempM1 = Sigmatilde1*U1'*input.M1;
252 [unused,unused,Vm0]=svd(tempM0(r0+1:r0+n0-r0,:));
253 [unused,unused,Vm1]=svd(tempM1(r1+1:r1+n1-r1,:));
254 perm0 = [zeros(n0-r0,r0) eye(n0-r0); eye(r0) zeros(r0,n0-r0)];
255 perm1 = [zeros(n1-r1,r1) eye(n1-r1); eye(r1) zeros(r1,n1-r1)];
256 tempM0 = tempM0*Vm0*perm0;
257 tempM1 = tempM1*Vm1*perm1;
258 M011 = tempM0(1:r0,1:r0); M012= tempM0(1:r0,r0+1:n0);
259 M022 = tempM0(r0+1:n0,r0+1:n0);
260 M111 = tempM1(1:r1,1:r1); M112= tempM1(1:r1,r1+1:n1);
261 M122 = tempM1(r1+1:n1,r1+1:n1);
262
263 B0 = Sigmatilde0*U0'*input.B0;
264 B1 = Sigmatilde1*U1'*input.B1;
265 B01 = B0(1:r0,:);
266 B02 = B0(r0+1:n0,:);
267 B11 = B1(1:r1,:);
268 B12 = B1(r1+1:n1,:);
269
270 Ahat = [A011 zeros(r0,r1); zeros(r1,r0) A111];
271 Dhat = [A012 zeros(r0,n1-r1); zeros(r1,n0-r0) A112];
272 Nhat = [M012 zeros(r0,n1-r1); zeros(r1,n0-r0) M112];
273 Bhat = [B01;B11];
274 Mhat = [M011 zeros(r0,r1+m1) ; zeros(r1,r0) M111 zeros(r1,m1)];
275 Atilde = [ A021 zeros(n0-r0,r1); zeros(n1-r1,r0) A121];
276 Dtilde = [A022 zeros(n0-r0,n1-r1); zeros(n1-r1,n0-r0) A122];
277 Btilde = [B02;B12];
278
279 Ntilde = blkdiag(M022,M122);
280 %-----------------------------------------------------%
281
282 Atilde = Ntilde\Atilde;
283 Btilde = Ntilde\Btilde;
284 Mtilde = 0;
285 Dtilde = Ntilde\Dtilde;
286
287 A = Ahat - Nhat*Atilde;
288 B = Bhat-Nhat*Btilde;
289 M = Mhat;
290 G = Dhat - Nhat*Dtilde;
291
```

```
292
293 % --------- end system matrices ----------------------------------------%
294 %----------------------------------------------------------------------%
295
296 %----------------------- Compile the information into Auxdata -----------%
297 % MATRICES                          % PARAMS
298 auxdata.P = input.P;                auxdata.m0 = m0;
299 auxdata.C = C;                      auxdata.m1 = m1;
300 auxdata.D0 = input.D0;              auxdata.r0 = r0;
301 auxdata.D1 = input.D1;              auxdata.r1 = r1;
302 auxdata.N0 = input.N0;              auxdata.n0 = n0;
303 auxdata.N1 = input.N1;              auxdata.n1 = n1;
304 auxdata.A = A;                      auxdata.n = n;
305 auxdata.G = G;
306 auxdata.B = B;
307 auxdata.M = M;
308 auxdata.Atilde = Atilde;
309 auxdata.Dtilde = Dtilde;
310 auxdata.Btilde = Btilde;
311 auxdata.hatP0 = input.P0;
312 auxdata.hatP1 = input.P1;
313 end
314
315 %----------------------------------------------------------------------%
316 function [ output ] = OPTu_checker_end(input)
317 bound_type = input.auxdata.bound_type;
318 P = input.auxdata.P;
319 q  = input.phase.integral;
320 r0 = input.auxdata.r0;
321 r1 = input.auxdata.r1;
322 output.objective = q;
323 hatP0 = input.auxdata.hatP0(1:r0,1:r0);
324 hatP1 = input.auxdata.hatP1(1:r1,1:r1);
325
326 %-------------- Linear BC ---------------------------------------%
327 x0 = input.phase.initialstate;
328 xf = input.phase.finalstate;
329     if strcmp(bound_type,'L2')==1
330         output.eventgroup.event = [x0(end)-1/2*...
331     x0(1:r0+r1)*P*x0(1:r0+r1)'];
332     elseif strcmp(bound_type,'infty')==1
333         output.eventgroup.event = [x0(end-1)-1/2*...
334          x0(1:r0+r1)*blkdiag(hatP0,zeros(r1))*x0(1:r0+r1)', ...
```

```
335           x0(end) - 1/2*x0(1:r0+r1)*blkdiag(zeros(r0),hatP1)*x0(1:r0+r1)'];
336     end
337 end
338 %}
```

```
1  function [last_known_proper,last_known_alpha, history] = ...
2              scaleu_alg1(t,u,bound_type,tol,max_iterations,matrices,params)
3  % function newu =
4  %           scaleu_alg1(t,u,bound_type,tol,max_iterations,matrices,params)
5  %
6  % This function takes a proper u for a particular bound and scales it to
7  % make it smaller and closer to minimal proper.  It is the code for
8  % Algorithm 4.1.
9  %
10 % INPUTS:
11 %   u - proper u
12 %   t - time grid where u is defined
13 %   bound_type = 'infty' or 'L2'   what bound type we are scaling with
14 %                   respect to
15 %   tol = algorithm will terminate when
16 %    (|| last_known_proper - utest ||)||last_known_proper||_\infty < tol
17 %           equivalently, when alpha < tol.
18 %   max_iterations = maximum number of iterations
19 %   matrices - matrices that define the problem:
20 %               E x ' + F x = Bu + M \mu
21 %               y   = C x + D u + N \nu
22 %   params - parameters for the problem
23 %   % t0 - start time
24     % tf - end time
25     % gamma - bound for noise \Gamma_{infty}<\gamma \Gamma{L^2} <
26     %         \gamma
27 %
28 % OUTPUTS:
29 %       last_known_proper - the scaled signal
30 %       history - iteration history
31
32 history = [];
33
34 %Optu_checker requires a scaling factor on the control.  We don't want to
35 %scale so alpha = 1.
36 alpha = 1;
```

```
37
38 n = 0;%initialie iteration counter
39 n0 = 1; %everytime we get J =0, we need to decrease alpha by a smaller
40 %           amount.  n0 keeps track of how much to decrease alpha.
41 last_known_proper = u;
42 last_known_alpha = 1;
43 while n < max_iterations
44     alpha = alpha - 10^(-n0); %decrease alpha to scale u down
45     utest = alpha*u;
46
47     %find J = \min ||y_0 - y_1||^2
48     [ J ] = OPTu_checker(t,u,matrices,params,bound_type,alpha);
49
50     %increment iteration counter and store history
51     n = n+1;
52     history = [history [n;alpha;J]];
53
54     if J < 10^(-7) % if J is zero
55         if ((last_known_alpha - alpha)/last_known_alpha) < tol
56             disp('Tolerance met.  Scaled u has been found.')
57             return
58         else %if tolerance has not been met but J = 0 then need to take
59                   %back up to what it was and increment n0
60             alpha = alpha + 10^(-n0);
61             n0 = n0 + 1;
62         end
63     else
64         last_known_proper = utest;
65         last_known_alpha = alpha;
66     end
67 end
68 disp('Maximum Iterations have been reached.  || y0 - y1 || = ')
69 disp(J)
70 end
```

A.2 Section 4.8 Code

Explanation of code listings:

- exec1.m - The main file that should be executed to produce the results of Section 4.8.

- define_problem.m - Stores matrices that define the problem and calculates matrices necessary for a solution.
- Find_u_asynch_phases.m - Finds the minimal proper u using GPOPS II.
- cost_for_r.m - Calculates the norm of the optimal control for a given r.

```
% main exec file for asynchronous signal design
% exec1.m
clc
clear all
close all
tol = 1e-3; %relative mesh tolerance for gpops
P_0 = zeros(2,2); %initial weight
P_1=zeros(2,2);

%pick noise measure
%noise_measure = 'infty';
noise_measure='L2';


%
%Example 1-3 (change r and s as needed):

r=1;%begin test signal
s=2;%end test window

auxdata = auxiliary_function(P_0,P_1,r,s); %builds some matrices
[u1,t1,x1,obj,betaa] = Find_u_asynch_phases(auxdata,tol,noise_measure);
obj
betaa
%}
figure
plot(t1,u1,'-rd','LineWidth',2)
xlabel('Time','FontSize',14)
ylabel('u','FontSize',14)
ax = gca;
set(ax,'FontSize',14)
%
% Example 4: s-r=1, T=3
r0=.2;
options = optimset('TolFun',2);
```

```
f = @(x)cost_for_r(x,P_0,P_1,noise_measure);
[r,fval,exitflag]=fmincon(f,r0,[],[],[],[],0,2,[],options);

auxdata = auxiliary_function(P_0,P_1,r,r+1); %builds some matrices
[u2,t2,x2,obj,betaa] = Find_u_asynch_phases(auxdata,1e-6,noise_measure);

obj
betaa
%}

figure
plot(t2,u2,'-rd','LineWidth',2)
xlabel('Time','FontSize',14)
ylabel('u','FontSize',14)
ax = gca;
set(ax,'FontSize',14)
%}
```

```
function [auxdata] = define_problem(P0,P1,r,s)
%This file defines matrices for the problem and constructs matrices
%necessary for solution.

params.t0 = 0;
params.r = r;
params.s = s;
params.tf = 3;
params.gamma = 1;
%s-r = a
params.a = 1;

A0 = [-1 2; 1 -2.5];
A1 = [-1.5 3; -1 -0.5];
%A0 = [0 1; -9 0];
%A1 = [0 1; -4 0];
[n,unused] = size(A0);
[m,unused] = size(A1);
n = n+m;
params.n = n;
B0 = [0;1];
B1 = [0;1];

```

```
N0 = eye(1);
N1 = eye(1);
M0 = eye(2);
M1 = eye(2);

C0 = [1 1];
C1 = [1 1];

Cbar0= N1\C0;
Nbar0 = N1\N0;
Cbar1 = N1\C1;

A = blkdiag(A0,A1);
B = [B0;B1];
[unused,n ] = size(N0);
M = blkdiag(M0,M1);
[m,unused] = size(M);
M = [M zeros(m,n)];
[unused,n] = size(Cbar0'*Cbar0);
[m,unused] = size(Cbar1'*Cbar1);
S1 = [Cbar0'*Cbar0 -Cbar0'*Cbar1; -Cbar1'*Cbar0 Cbar1'*Cbar1];
%S1 = [Cbar0'*Cbar0 -2*Cbar0'*Cbar1; zeros(m,n) Cbar1'*Cbar1];
[unused,n] = size([M0 M1]);
[m,unused] = size([Cbar0'*Nbar0; -Cbar1'*Nbar0]);
S2 = [zeros(m,n) [Cbar0'*Nbar0; -Cbar1'*Nbar0]];
[m,unused] = size(Nbar0'*Nbar0);
S3 = blkdiag(eye(n), eye(m)+Nbar0'*Nbar0);



P = blkdiag(P0,P1);
%P=.1;
auxdata.params = params;
auxdata.A = A;
auxdata.B = B;
auxdata.P = P;
auxdata.S1 = S1;
auxdata.S2 = S2;
auxdata.S3 = S3;
auxdata.M = M;
[unused,auxdata.nx0]=size(A0);
[unused,auxdata.nx1]=size(A1);
[auxdata.ny,unused] = size(C0);
```

```
auxdata.Nbar0 = Nbar0;
auxdata.P = P;
auxdata.P0= P0;
auxdata.P1 = P1;
```

```
function [u_opt,t_opt,x, obj,betaa] = Find_u_asynch_phases(auxdata,...
    tol,noise_measure)
%
% function [ u_opt,t_opt ] = Find_u_asynch_phases()
%
% This function finds a minimal proper u.
% INPUT:
%        auxdata - parameters and matrices created to solve the problem
%        tol - mesh tolerance for gpops II
%        noise_measure - 'L2'or 'infty'
% Ouput =
%            u_opt - minimal proper u that is defined on   t_opt
%            t_opt - the time grid for u
%               x - optimal state
%               obj - optimal objective value ||u||^2
%               betaa - if using max norm for noise, it's the optimal beta
%

% Step 0: Prepare auxdata, settings and parameters to pass to GpopsII

%---------------Get all matrices and parameters -------------------------%
params = auxdata.params;
%-----------------------------------------------------------------------%
%get which case we are in
if -1e-8 < params.r && 1e-8> params.r
    params.r=0;
end
if params.r == params.t0 && params.s == params.tf
    phase_case=1;
elseif params.r == params.t0 && params.s < params.tf
    phase_case=2;
elseif params.t0<params.r && params.s < params.tf
    phase_case=3;
elseif params.t0< params.r && params.s == params.tf
    phase_case = 4;
else
```

```
37      disp('Error')
38      params.r
39      params.t0
40      params.s
41      params.tf
42  end
43  params.phase_case=phase_case;
44
45  params.noise_measure = noise_measure;
46
47  auxdata.params=params;
48
49  % Get settings for GPOPSII%
50  setup = properu_settings_phases(params,tol,phase_case);
51  %------------------------------------------------------------------------%
52  if strcmp(noise_measure,'infty')==1
53      setup.bounds.parameter.lower = .3;
54      setup.bounds.parameter.upper = .7;
55      setup.guess.parameter = .6;
56  end
57  setup.auxdata=auxdata;
58  % ---------------------------------------------------------- %
59
60  % Solve with GPOPSII
61  output = gpops2(setup);
62
63  %gather output
64  t_opt = [output.result.solution.phase(1).time];
65  u_opt = [output.result.solution.phase(1).control];
66  x = [output.result.solution.phase(1).state];
67  obj = output.result.objective;
68  if phase_case > 1
69      t_opt = [t_opt;output.result.solution.phase(2).time];
70      u_opt = [u_opt;output.result.solution.phase(2).control];
71      x = [x;output.result.solution.phase(2).state];
72  end
73
74  if phase_case == 3
75      t_opt = [t_opt;output.result.solution.phase(3).time];
76      u_opt = [u_opt;output.result.solution.phase(3).control];
77      x = [x;output.result.solution.phase(3).state];
78  end
79
```

```
80
81 if strcmp(noise_measure,'infty') == 1
82     betaa = output.result.solution.parameter;
83 else
84     betaa='There is no beta when using L^2 norm for noise measure.';
85 end
86
87 end
88 %-------------------------------------------------------------------------%
89
90
91
92 % Helper functions
93
94
95
96 function phaseout = properu_cont_phases(input)
97 %-------------------------------------------------------------------------%
98 %------------------------ Define needed Params ---------------------------%
99 %-------------------------------------------------------------------------%
100 n = input.auxdata.params.n;
101 P = input.auxdata.P;
102 S1= input.auxdata.S1;
103 S2 = input.auxdata.S2;
104 S3 = input.auxdata.S3;
105 A = input.auxdata.A;
106 B = input.auxdata.B;
107 M = input.auxdata.M;
108 phase_case = input.auxdata.params.phase_case;
109 nx0 = input.auxdata.nx0;
110 nx1 = input.auxdata.nx1;
111 ny = input.auxdata.ny;
112 Nbar0 = input.auxdata.Nbar0;
113
114 %if using Linfty noise measure need to modify some matrices
115 if strcmp(input.auxdata.params.noise_measure,'infty')==1
116     betaa = input.phase.parameter;
117     betaa = betaa(1);
118     S1 = (1-betaa)*S1;
119     S2 = (1-betaa)*S2;
120     S3 = blkdiag(betaa*eye(nx0), (1-betaa)*eye(nx1), ...
121         betaa*eye(ny)+(1-betaa)*Nbar0'*Nbar0);
122 end
```

```
123
124 switch phase_case
125     case 1
126         %phase 1
127         t1 = input.phase.time;
128         z1= input.phase.state;
129         u1 = input.phase.control;
130
131         %--------------------------------------------------------------------%
132         x = z1(:,1:n); %state (x_0,x_1)
133         lambda = z1(:,n+1:end-1); %costate
134
135         [tel,unused]=size(x);
136
137
138         dx = (A-M*(S3\S2'))*x' + B*u1' - (M*(S3\M'))*lambda';
139         dlam = -(A'-S2*(S3\M'))*lambda'-(S1-S2*(S3\S2'))*x';
140
141         dpsi = zeros(tel,1);
142         for i = 1:tel
143             eta = -S3\(M'*lambda(i,:)'+S2'*x(i,:)');
144             dpsi(i) = 1/2*(x(i,:)*S1*x(i,:)' + 2*x(i,:)*S2*eta + ...
145                 eta'*S3*eta);
146         end
147
148
149         integrand = sum(u1'.^2,1)';
150
151         xdot = [dx' dlam' dpsi];
152         phaseout.dynamics = xdot;
153         phaseout.integrand = integrand;
154         %--------------------------------------------------------------------%
155
156     case 2
157
158         %--------------------------------------------------------------------%
159         %phase 1
160         t1 = input.phase(1).time;
161         z1= input.phase(1).state;
162         u1 = input.phase(1).control;
163
164         %--------------------------------------------------------------------%
165         x = z1(:,1:n); %state (x_0,x_1)
```

```
166         lambda = z1(:,n+1:end-1); %costate
167
168         [tel,unused]=size(x);
169
170
171         dx = (A-M*(S3\S2'))*x'+ B*u1' - (M*(S3\M'))*lambda';
172         dlam = -(A'-S2*(S3\M'))*lambda'-(S1-S2*(S3\S2'))*x';
173
174         dpsi = zeros(tel,1);
175         for i = 1:tel
176             eta = -S3\(M'*lambda(i,:)'+S2'*x(i,:)');
177             dpsi(i) = 1/2*(x(i,:)*S1*x(i,:)' + 2*x(i,:)*S2*eta +...
178                 eta'*S3*eta);
179         end
180
181
182         integrand = sum(u1'.^2,1)';
183
184         xdot = [dx' dlam' dpsi];
185         phaseout(1).dynamics = xdot;
186         phaseout(1).integrand = integrand;
187         %--------------------------------------------------------------------%
188
189         %--------------------------------------------------------------------%
190         %phase 3
191         t2 = input.phase(2).time;
192         z2= input.phase(2).state;
193         u2 = input.phase(2).control;
194
195         %--------------------------------------------------------------------%
196         x = z2(:,1:n); %state (x_0,x_1)
197         lambda = z2(:,n+1:end-1); %costate
198
199         [tel,unused]=size(x);
200
201
202         dx = (A-M*(S3\S2'))*x'- (M*(S3\M'))*lambda';
203         dlam = -(A'-S2*(S3\M'))*lambda'-(S1-S2*(S3\S2'))*x';
204
205         dpsi = zeros(tel,1);
206         for i = 1:tel
207             eta = -S3\(M'*lambda(i,:)'+S2'*x(i,:)');
208             dpsi(i) = 1/2*(x(i,:)*S1*x(i,:)' + 2*x(i,:)*S2*eta + ...
```

```
209                 eta'*S3*eta);
210         end
211
212
213         integrand = sum(u2'.^2,1)';
214
215         xdot = [dx' dlam' dpsi];
216         phaseout(2).dynamics = xdot;
217         phaseout(2).integrand = integrand;
218     case 3
219         %phase 1
220         t1 = input.phase(1).time;
221         z1= input.phase(1).state;
222         u1 = input.phase(1).control;
223
224         %--------------------------------------------------------------------%
225         x = z1(:,1:n); %state (x_0,x_1)
226         lambda = z1(:,n+1:end-1); %costate
227
228         [tel,unused]=size(x);
229
230
231         dx = (A-M*(S3\S2'))*x' - (M*(S3\M'))*lambda';
232         dlam = -(A'-S2*(S3\M'))*lambda'-(S1-S2*(S3\S2'))*x';
233
234         dpsi = zeros(tel,1);
235         for i = 1:tel
236             eta = -S3\(M'*lambda(i,:)'+S2'*x(i,:)');
237             dpsi(i) = 1/2*(x(i,:)*S1*x(i,:)' + 2*x(i,:)*S2*eta + ...
238                 eta'*S3*eta);
239         end
240
241
242         integrand = sum(u1'.^2,1)';
243
244         xdot = [dx' dlam' dpsi];
245         phaseout(1).dynamics = xdot;
246         phaseout(1).integrand = integrand;
247         %--------------------------------------------------------------------%
248
249         %--------------------------------------------------------------------%
250         %phase 2
251         t1 = input.phase(2).time;
```

```
252         z1= input.phase(2).state;
253         u1 = input.phase(2).control;
254 
255         %--------------------------------------------------------------------%
256         x = z1(:,1:n); %state (x_0,x_1)
257         lambda = z1(:,n+1:end-1); %costate
258 
259         [tel,unused]=size(x);
260 
261 
262         dx = (A-M*(S3\S2'))*x'+ B*u1' - (M*(S3\M'))*lambda';
263         dlam = -(A'-S2*(S3\M'))*lambda'-(S1-S2*(S3\S2'))*x';
264 
265         dpsi = zeros(tel,1);
266         for i = 1:tel
267             eta = -S3\(M'*lambda(i,:)'+S2'*x(i,:)');
268             dpsi(i) = 1/2*(x(i,:)*S1*x(i,:)' + 2*x(i,:)*S2*eta +...
269                 eta'*S3*eta);
270         end
271 
272 
273         integrand = sum(u1'.^2,1)';
274 
275         xdot = [dx' dlam' dpsi];
276         phaseout(2).dynamics = xdot;
277         phaseout(2).integrand = integrand;
278         %--------------------------------------------------------------------%
279 
280         %--------------------------------------------------------------------%
281         %phase 3
282         t2 = input.phase(3).time;
283         z2= input.phase(3).state;
284         u2 = input.phase(3).control;
285 
286         %--------------------------------------------------------------------%
287         x = z2(:,1:n); %state (x_0,x_1)
288         lambda = z2(:,n+1:end-1); %costate
289 
290         [tel,unused]=size(x);
291 
292 
293         dx = (A-M*(S3\S2'))*x'- (M*(S3\M'))*lambda';
294         dlam = -(A'-S2*(S3\M'))*lambda'-(S1-S2*(S3\S2'))*x';
```

```
295
296         dpsi = zeros(tel,1);
297         for i = 1:tel
298             eta = -S3\(M'*lambda(i,:)'+S2'*x(i,:)');
299             dpsi(i) = 1/2*(x(i,:)*S1*x(i,:)' + 2*x(i,:)*S2*eta +...
300                 eta'*S3*eta);
301         end
302
303
304         integrand = sum(u2'.^2,1)';
305
306         xdot = [dx' dlam' dpsi];
307         phaseout(3).dynamics = xdot;
308         phaseout(3).integrand = integrand;
309     case 4
310         %----------------------------------------------------------------%
311         %phase 1
312         t1 = input.phase(1).time;
313         z1= input.phase(1).state;
314         u1 = input.phase(1).control;
315
316         %----------------------------------------------------------------%
317         x = z1(:,1:n); %state (x_0,x_1)
318         lambda = z1(:,n+1:end-1); %costate
319
320         [tel,unused]=size(x);
321
322
323         dx = (A-M*(S3\S2'))*x' - (M*(S3\M'))*lambda';
324         dlam = -(A'-S2*(S3\M'))*lambda'-(S1-S2*(S3\S2'))*x';
325
326         dpsi = zeros(tel,1);
327         for i = 1:tel
328             eta = -S3\(M'*lambda(i,:)'+S2'*x(i,:)');
329             dpsi(i) = 1/2*(x(i,:)*S1*x(i,:)' + 2*x(i,:)*S2*eta +...
330                 eta'*S3*eta);
331         end
332
333
334         integrand = sum(u1'.^2,1)';
335
336         xdot = [dx' dlam' dpsi];
337         phaseout(1).dynamics = xdot;
```

```
338         phaseout(1).integrand = integrand;
339         %--------------------------------------------------------------------%
340 
341         %--------------------------------------------------------------------%
342         %phase 2
343         t2 = input.phase(2).time;
344         z2= input.phase(2).state;
345         u2 = input.phase(2).control;
346 
347         %--------------------------------------------------------------------%
348         x = z2(:,1:n); %state (x_0,x_1)
349         lambda = z2(:,n+1:end-1); %costate
350 
351         [tel,unused]=size(x);
352 
353 
354         dx = (A-M*(S3\S2'))*x'+ B*u2'- (M*(S3\M'))*lambda';
355         dlam = -(A'-S2*(S3\M'))*lambda'-(S1-S2*(S3\S2'))*x';
356 
357         dpsi = zeros(tel,1);
358         for i = 1:tel
359             eta = -S3\(M'*lambda(i,:)'+S2'*x(i,:)');
360             dpsi(i) = 1/2*(x(i,:)*S1*x(i,:)' + 2*x(i,:)*S2*eta +...
361                 eta'*S3*eta);
362         end
363 
364 
365         integrand = sum(u2'.^2,1)';
366 
367         xdot = [dx' dlam' dpsi];
368         phaseout(2).dynamics = xdot;
369         phaseout(2).integrand = integrand;
370 end
371 
372 end
373 %------------------------------------------------------------------------ %
374 
375 function output = properu_end_phases(input)
376 P = input.auxdata.P;
377 if strcmp(input.auxdata.params.noise_measure,'infty')==1
378     betaa = input.parameter;
379     P0 = input.auxdata.P0;
380     P1 = input.auxdata.P1;
```

```
381     P = blkdiag(betaa*P0,(1-betaa)*P1);
382 end
383 phase_case = input.auxdata.params.phase_case;
384 n = input.auxdata.params.n;
385 switch phase_case
386     case 1
387         q = input.phase.integral;
388         x0 = input.phase.initialstate;
389         xf = input.phase.finalstate;
390         output.eventgroup.event = [(P*x0(1:n)'+x0(n+1:end-1)')', ...
391             x0(end)-1/2*x0(1:n)*P*x0(1:n)'];
392     case 2
393         q1 = input.phase(1).integral;
394         q2  = input.phase(2).integral;
395         q = q1+q2;
396         % Variables at Start and Terminus of Phase 1
397         x0{1} = input.phase(1).initialstate;
398         xf{1} = input.phase(1).finalstate;
399         % Variables at Start and Terminus of Phase 2
400         x0{2} = input.phase(2).initialstate;
401         xf{2} = input.phase(2).finalstate;
402         % Event Group 1: Linkage Constraints Between Phases 1 and 2
403         output.eventgroup(1).event = [(P*x0{1}(1:n)'+x0{1}(n+1:end-1)')'...
404             ,x0{1}(end)-1/2*x0{1}(1:n)*P*x0{1}(1:n)', x0{2}-xf{1}];
405     case 3
406         q1 = input.phase(1).integral;
407         q2  = input.phase(2).integral;
408         q3 = input.phase(3).integral;
409         q = q1+q2 + q3;
410         % Variables at Start and Terminus of Phase 1
411         x0{1} = input.phase(1).initialstate;
412         xf{1} = input.phase(1).finalstate;
413         % Variables at Start and Terminus of Phase 2
414         x0{2} = input.phase(2).initialstate;
415         xf{2} = input.phase(2).finalstate;
416         % Variables at Start and Terminus of Phase 2
417         x0{3} = input.phase(3).initialstate;
418         xf{3} = input.phase(3).finalstate;
419 
420         % Event Group 1: Linkage Constraints Between Phases 1 and 2
421         output.eventgroup(1).event = [(P*x0{1}(1:n)'+x0{1}(n+1:end-1)')'...
422             ,x0{1}(end)-1/2*x0{1}(1:n)*P*x0{1}(1:n)', x0{2}-xf{1}];
423         output.eventgroup(2).event = [ x0{3}-xf{2}];
```

```
424     case 4
425         q1 = input.phase(1).integral;
426         q2  = input.phase(2).integral;
427         q = q1+q2;
428         % Variables at Start and Terminus of Phase 1
429         x0{1} = input.phase(1).initialstate;
430         xf{1} = input.phase(1).finalstate;
431         % Variables at Start and Terminus of Phase 2
432         x0{2} = input.phase(2).initialstate;
433         xf{2} = input.phase(2).finalstate;
434 
435         % Event Group 1: Linkage Constraints Between Phases 1 and 2
436         output.eventgroup(1).event = [(P*x0{1}(1:n)'+x0{1}(n+1:end-1)')'...
437             ,x0{1}(end)-1/2*x0{1}(1:n)*P*x0{1}(1:n)', x0{2}-xf{1}];
438 end
439 
440 %-------------------------------------------------------
441 output.objective = q;
442 end
443 %------------------------------------------------------------- %
444 
445 function [ setup ] =properu_settings_phases(params,tol,phase_case)
446 % This function defines the settings to pass into GPOPSII.
447 %
448 %Input:
449 % params.t0 = initial time
450 % params.tf = final time
451 % params.n = dimension of state (x_0 and x_1)
452 % P = initial weight matrix
453 % tol = mesh tolerance
454 % phase_case = switch variable 1-4 that signals ordering and how many phases
455 %
456 %
457 %
458 % Output;
459 % setup - settings to pass to gpopsII
460 
461 %------------------------------------------------------------------------%
462 %---------------------- Setup for Problem Bounds ------------------------%
463 %------------------------------------------------------------------------%
464 switch phase_case
465     case 1
466         [bounds,mesh,guess] = settings_1(params);
```

```
467     case 2
468         [bounds,mesh,guess] = settings_2(params);
469     case 3
470         [bounds,mesh,guess] = settings_3(params);
471     case 4
472         [bounds,mesh,guess] = settings_4(params);
473 end
474
475
476
477
478
479
480 %-------------------------------------------------------------------------%
481 %----------Provide Mesh Refinement Method and Initial Mesh ---------------%
482 %
483 %optional fields
484 mesh.method          = 'hp1';
485 mesh.tolerance       = tol;
486 mesh.maxiteration    = 10;
487 mesh.colpointsmin    = 4;
488 mesh.colpointsmax    = 12;
489 %-------------------------------------------------------------------------%
490 %------------- Assemble Information into Problem Structure ---------------%
491 %-------------------------------------------------------------------------%
492 setup.name = 'Find minimal proper u';
493 setup.functions.continuous = @properu_cont_phases;
494 setup.functions.endpoint = @properu_end_phases;
495 setup.bounds = bounds;
496 setup.guess = guess;
497 setup.mesh = mesh;
498 setup.nlp.solver = 'ipopt';
499 setup.nlp.options.ipopt.linear_solver = 'ma57';
500 %setup.derivatives.supplier = 'sparseBD';
501 setup.derivatives.derivativelevel = 'second';
502 %setup.scales.method = 'automatic-bounds';
503 setup.method = 'RPMintegration';
504 end
505 % ------------------------------------------------------------------ %
506
507
508 function [bounds, mesh, guess] = settings_1(params)
509
```

```
%case 1 only 1 phase - t_0 = r  and t_f = s
bounds.phase.initialtime.lower = params.t0;
bounds.phase.initialtime.upper = params.t0;
bounds.phase.finaltime.lower = params.tf;
bounds.phase.finaltime.upper = params.tf;
n = params.n; %size of state and costate

bounds.phase.initialstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
bounds.phase.initialstate.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
bounds.phase.state.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
bounds.phase.state.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
bounds.phase.finalstate.lower = [-1e8*ones(1,n) zeros(1,n) params.gamma^2];
bounds.phase.finalstate.upper = [1e8*ones(1,n) zeros(1,n) 1e10];

bounds.phase.control.lower = -1e9;
bounds.phase.control.upper = 1e9;
bounds.phase.integral.lower = 0;
bounds.phase.integral.upper = 1e10;

guessx = [ones(1,n) ones(1,n) 0; ones(1,n) ones(1,n) 1.2*params.gamma^2];
guess.phase.time     = [params.t0; params.tf];
guess.phase.state    = guessx;
guess.phase.control  = [1;1];
guess.phase.integral = 1;
%------------------------ Events - ---------------------------------%
%link phases and initial
bounds.eventgroup.lower = [zeros(1,n+1)];
bounds.eventgroup.upper = [zeros(1,n+1)];
%-------------------------------------------------------------------%
mesh.phase.colpoints = 4*ones(1,10);
mesh.phase.fraction  = 0.1*ones(1,10);
end

function [bounds, mesh, guess] = settings_2(params)
%case 2 - 2 phases t0=r< s < tf
iphase=1;
bounds.phase(iphase).initialtime.lower = params.t0;
bounds.phase(iphase).initialtime.upper = params.t0;
bounds.phase(iphase).finaltime.lower = params.s;
bounds.phase(iphase).finaltime.upper = params.s;
n = params.n; %size of state and costate

bounds.phase(iphase).initialstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n)...
```

```
    0];
bounds.phase(iphase).initialstate.upper = [1e8*ones(1,n) 1e8*ones(1,n)...
    1e10];
bounds.phase(iphase).state.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
bounds.phase(iphase).state.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
bounds.phase(iphase).finalstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n)...
    0];
bounds.phase(iphase).finalstate.upper = [1e8*ones(1,n) 1e8*ones(1,n)...
    1e10];

bounds.phase(iphase).control.lower = -1e9;
bounds.phase(iphase).control.upper = 1e9;
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1e10;

guessx = [ones(1,n) ones(1,n) 0; ones(1,n) ones(1,n) .2*params.gamma^2];
guess.phase(iphase).time     = [params.t0; params.s];
guess.phase(iphase).state    = guessx;
guess.phase(iphase).control  = [1;1];
guess.phase(iphase).integral = 1;

%-------------------------------------------------------------------------%
iphase = 2;
bounds.phase(iphase).initialtime.lower = params.s;
bounds.phase(iphase).initialtime.upper = params.s;
bounds.phase(iphase).finaltime.lower = params.tf;
bounds.phase(iphase).finaltime.upper = params.tf;
n = params.n; %size of state and costate

bounds.phase(iphase).initialstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n)...
    0];
bounds.phase(iphase).initialstate.upper = [1e8*ones(1,n) 1e8*ones(1,n)...
    1e10];
bounds.phase(iphase).state.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
bounds.phase(iphase).state.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
bounds.phase(iphase).finalstate.lower = [-1e8*ones(1,n) zeros(1,n)...
    params.gamma^2];
bounds.phase(iphase).finalstate.upper = [1e8*ones(1,n) zeros(1,n) 1e10];

bounds.phase(iphase).control.lower = -1e9;
bounds.phase(iphase).control.upper = 1e9;
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1e10;
```

```
596
597 guessx = [ones(1,n) ones(1,n) .2*params.gamma^2; ones(1,n) ones(1,n)...
598     1.2*params.gamma^2];
599 guess.phase(iphase).time     = [params.s; params.tf];
600 guess.phase(iphase).state    = guessx;
601 guess.phase(iphase).control  = [1;1];
602 guess.phase(iphase).integral = 1;
603 %------------------------------------------------------------------------%
604
605
606 %------------------------ Events - -------------------------------%
607 %link phases and initial
608 bounds.eventgroup(1).lower = [zeros(1,n+1),zeros(1,2*n+1)];
609 bounds.eventgroup(1).upper = [zeros(1,n+1) ,zeros(1,2*n+1)];
610 %------------------------------------------------------------------------%
611 for i = 1:2
612 mesh.phase(i).colpoints = 4*ones(1,10);
613 mesh.phase(i).fraction  = 0.1*ones(1,10);
614 end
615 end
616
617
618
619 function [bounds, mesh, guess] = settings_3(params)
620 % case 3 - 3 phases, t0 < r < s < tf
621 iphase=1;
622 bounds.phase(iphase).initialtime.lower = params.t0;
623 bounds.phase(iphase).initialtime.upper = params.t0;
624 bounds.phase(iphase).finaltime.lower = params.r;
625 bounds.phase(iphase).finaltime.upper = params.r;
626 n = params.n; %size of state and costate
627
628 bounds.phase(iphase).initialstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n)...
629     0];
630 bounds.phase(iphase).initialstate.upper = [1e8*ones(1,n) 1e8*ones(1,n)...
631     1e10];
632 bounds.phase(iphase).state.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
633 bounds.phase(iphase).state.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
634 bounds.phase(iphase).finalstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
635 bounds.phase(iphase).finalstate.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
636
637 bounds.phase(iphase).control.lower = -1e9;
638 bounds.phase(iphase).control.upper = 1e9;
```

```
639 bounds.phase(iphase).integral.lower = 0;
640 bounds.phase(iphase).integral.upper = 1e10;
641
642 guessx = [ones(1,n) ones(1,n) 0; ones(1,n) ones(1,n) .2*params.gamma^2];
643 guess.phase(iphase).time     = [params.t0; params.r];
644 guess.phase(iphase).state    = guessx;
645 guess.phase(iphase).control  = [1;1];
646 guess.phase(iphase).integral = 1;
647
648
649 %--------------------------------------------------------------------%
650 iphase = 2;
651 bounds.phase(iphase).initialtime.lower = params.r;
652 bounds.phase(iphase).initialtime.upper = params.r;
653 bounds.phase(iphase).finaltime.lower = params.s;
654 bounds.phase(iphase).finaltime.upper = params.s;
655 n = params.n; %size of state and costate
656
657 bounds.phase(iphase).initialstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n)...
658     0];
659 bounds.phase(iphase).initialstate.upper = [1e8*ones(1,n) 1e8*ones(1,n)...
660     1e10];
661 bounds.phase(iphase).state.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
662 bounds.phase(iphase).state.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
663 bounds.phase(iphase).finalstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
664 bounds.phase(iphase).finalstate.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
665
666 bounds.phase(iphase).control.lower = -1e9;
667 bounds.phase(iphase).control.upper = 1e9;
668 bounds.phase(iphase).integral.lower = 0;
669 bounds.phase(iphase).integral.upper = 1e10;
670
671 guessx = [ones(1,n) ones(1,n) .2*params.gamma^2; ones(1,n) ones(1,n)...
672     1.2*params.gamma^2];
673 guess.phase(iphase).time     = [params.r; params.s];
674 guess.phase(iphase).state    = guessx;
675 guess.phase(iphase).control  = [1;1];
676 guess.phase(iphase).integral = 1;
677 %--------------------------------------------------------------------%
678
679 %--------------------------------------------------------------------%
680 iphase = 3;
681 bounds.phase(iphase).initialtime.lower = params.s;
```

```
682 bounds.phase(iphase).initialtime.upper = params.s;
683 bounds.phase(iphase).finaltime.lower = params.tf;
684 bounds.phase(iphase).finaltime.upper = params.tf;
685 n = params.n; %size of state and costate
686
687 bounds.phase(iphase).initialstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n)...
688     0];
689 bounds.phase(iphase).initialstate.upper = [1e8*ones(1,n) 1e8*ones(1,n)...
690     1e10];
691 bounds.phase(iphase).state.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
692 bounds.phase(iphase).state.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
693 bounds.phase(iphase).finalstate.lower = [-1e8*ones(1,n) zeros(1,n)...
694     params.gamma^2];
695 bounds.phase(iphase).finalstate.upper = [1e8*ones(1,n) zeros(1,n) 1e10];
696
697 bounds.phase(iphase).control.lower = -1e9;
698 bounds.phase(iphase).control.upper = 1e9;
699 bounds.phase(iphase).integral.lower = 0;
700 bounds.phase(iphase).integral.upper = 1e10;
701
702 guessx = [ones(1,n) ones(1,n) .2*params.gamma^2; ones(1,n) ones(1,n)...
703     1.2*params.gamma^2];
704 guess.phase(iphase).time     = [params.s; params.tf];
705 guess.phase(iphase).state    = guessx;
706 guess.phase(iphase).control  = [1;1];
707 guess.phase(iphase).integral = 1;
708 %------------------------------------------------------------------------%
709
710
711 %------------------------- Events - --------------------------------%
712 %link phases and initial
713 bounds.eventgroup(1).lower = [zeros(1,n+1),zeros(1,2*n+1)];
714 bounds.eventgroup(1).upper = [zeros(1,n+1) ,zeros(1,2*n+1)];
715 bounds.eventgroup(2).lower = [zeros(1,2*n+1)];
716 bounds.eventgroup(2).upper = [zeros(1,2*n+1)];
717 %------------------------------------------------------------------------%
718 for i = 1:3
719 mesh.phase(i).colpoints = 4*ones(1,10);
720 mesh.phase(i).fraction  = 0.1*ones(1,10);
721 end
722 end
723
724 function [bounds, mesh, guess] = settings_4(params)
```

```
% case 4 - 2 phases, t0 < r < s=tf
iphase=1;
bounds.phase(iphase).initialtime.lower = params.t0;
bounds.phase(iphase).initialtime.upper = params.t0;
bounds.phase(iphase).finaltime.lower = params.r;
bounds.phase(iphase).finaltime.upper = params.r;
n = params.n; %size of state and costate

bounds.phase(iphase).initialstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n)...
    0];
bounds.phase(iphase).initialstate.upper = [1e8*ones(1,n) 1e8*ones(1,n)...
    1e10];
bounds.phase(iphase).state.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
bounds.phase(iphase).state.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
bounds.phase(iphase).finalstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
bounds.phase(iphase).finalstate.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];

bounds.phase(iphase).control.lower = -1e9;
bounds.phase(iphase).control.upper = 1e9;
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1e10;

guessx = [ones(1,n) ones(1,n) 0; ones(1,n) ones(1,n) .2*params.gamma^2];
guess.phase(iphase).time     = [params.t0; params.r];
guess.phase(iphase).state    = guessx;
guess.phase(iphase).control  = [1;1];
guess.phase(iphase).integral = 1;

%------------------------------------------------------------------------%
iphase = 2;
bounds.phase(iphase).initialtime.lower = params.r;
bounds.phase(iphase).initialtime.upper = params.r;
bounds.phase(iphase).finaltime.lower = params.tf;
bounds.phase(iphase).finaltime.upper = params.tf;
n = params.n; %size of state and costate

bounds.phase(iphase).initialstate.lower = [-1e8*ones(1,n) -1e8*ones(1,n)...
    0];
bounds.phase(iphase).initialstate.upper = [1e8*ones(1,n) 1e8*ones(1,n)...
    1e10];
bounds.phase(iphase).state.lower = [-1e8*ones(1,n) -1e8*ones(1,n) 0];
bounds.phase(iphase).state.upper = [1e8*ones(1,n) 1e8*ones(1,n) 1e10];
bounds.phase(iphase).finalstate.lower = [-1e8*ones(1,n) zeros(1,n)...
```

```
    params.gamma^2];
bounds.phase(iphase).finalstate.upper = [1e8*ones(1,n) zeros(1,n) 1e10];

bounds.phase(iphase).control.lower = -1e9;
bounds.phase(iphase).control.upper = 1e9;
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1e10;

guessx = [ones(1,n) ones(1,n) .2*params.gamma^2; ones(1,n) ones(1,n)...
    1.2*params.gamma^2];
guess.phase(iphase).time     = [params.r; params.tf];
guess.phase(iphase).state    = guessx;
guess.phase(iphase).control  = [1;1];
guess.phase(iphase).integral = 1;
%------------------------------------------------------------------------%


%--------------------------- Events - ---------------------------------%
%link phases and initial
bounds.eventgroup(1).lower = [zeros(1,n+1),zeros(1,2*n+1)];
bounds.eventgroup(1).upper = [zeros(1,n+1) ,zeros(1,2*n+1)];
%------------------------------------------------------------------------%
for i = 1:2
mesh.phase(i).colpoints = 4*ones(1,10);
mesh.phase(i).fraction  = 0.1*ones(1,10);
end
end
```

```
function Obj = cost_for_r(x,P_0,P_1,noise_measure)
%This function returns ||u|| given an r

auxdata = auxiliary_function(P_0,P_1,x,x+1); %builds some matrices
[u1,t1,x1,Obj] = Find_u_asynch_phases(auxdata,1e-2,noise_measure);

end
```