System Identification Techniques and Modeling for Nonintrusive Load Diagnostics

by

Steven Robert Shaw

S.B. Massachusetts Institute of Technology (1995)
M.Eng. Massachusetts Institute of Technology (1997)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2000

Author .....
Department of Electrical Engineering and Computer Science
January 25, 2000

Certified by .....
Steven B. Leeb
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Departmental Committee on Graduate Theses

System Identification Techniques and Modeling for Nonintrusive Load Diagnostics

by

Steven Robert Shaw

Submitted to the Department of Electrical Engineering and Computer Science
on January 25, 2000, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis addresses the requirements of a system that can detect on/off transients and identify physical parameters of loads connected to a power distribution network. The thesis emphasizes three areas; a transient classifier that recognizes load transients using a pattern matching scheme, parameter estimation techniques suited for use with this classifier, and case studies of modeling and identification motivated by diagnostics and performance monitoring. Together, these areas support applications that can extract detailed load information from centralized, easily accessible parts of a distribution network.

A new approach and implementation of pattern-based nonintrusive transient classification is presented. The classifier is nonintrusive in the sense that it uses aggregated measurements at a central location and does not require instrumentation of individual loads. The classifier implementation includes a framework that integrates preprocessors for AC and DC environments, programs that present results, and load-specific parameter identification modules that are executed as their associated transients are classified. An obstacle for these parameter identification programs is that a good initial guess is needed for the iterative optimization routines typically used to find parameter estimates. Two approaches are given to overcome this problem for certain systems. The first extends conventional optimization methods to identify model parameters given a poor initial guess. The second approach treats the identification as a modeling problem and suggests ways to construct "inverse" models that map observations to parameter estimates without iteration. The techniques presented in the thesis are demonstrated with simulation data and in real world scenarios including a dormitory, an automobile, and an experimental building.

Thesis Supervisor: Steven B. Leeb
Title: Carl Richard Soderberg Associate Professor of Power Engineering

Acknowledgments

I would like to thank Professor Steven Leeb for his guidance and patience. Professor Leeb's enthusiasm is unwavering and inspiring, and his support over the last few years is much appreciated.

I am also thankful for the many contributions of my thesis readers, Professors Kirtley, Norford, and White.

The implementation reported in this thesis depends largely on freely distribed software. Tools such as gcc, g77, Octave, Perl, and routines from packages like ODEPACK, MINPACK, LAPACK and the BLAS were essential. I was consistently amazed by the rapid and effective advice provided by members of the free software community. In particular, Matt Welsh's `das1200` driver and email suggestions convinced me that the project would work as envisioned.

Thanks are due to Andy Suby and the crew at the Iowa Energy Center for their essential help in setting up experiments and collecting data. Students Craig Abler, Chris Laughman and Chris Salthouse provided important support throughout. Professors Bernard Lesieutre and George Verghese deserve thanks for their many useful suggestions.

Intel, Tektronix and Hewlett-Packard generously donated equipment used in this work. Other support was provided by the National Science Foundation; AMP incorporated, including Mr. Joseph P. Sweeney, Mr. Mike LeVan, Mr. Tom Davis, and Dr. Howard Peiffer; and Dr. Emanuel Landsman.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Consumers demand reliability from complicated, highly engineered systems like satellites, automobiles, and building energy management systems. The costs of unanticipated failure in such system are high, and even when a failure occurs, it may be difficult to find out what happened and take appropriate steps to prevent further problems. Reliability can be sometimes be achieved through redundancy or rigorous maintenance schedules, often at high cost. An alternate approach might exploit inexpensive computation and communication resources to identify, report, and target maintenance to components of systems that are about to fail. With the correct diagnostic tools, plant managers, utilities, and maintenance companies could offer enhanced reliability and decreased maintenance costs without extensive retrofit of existing systems.

An inexpensive, minimally invasive diagnostic device would be very valuable in the context of avoiding failure in complicated systems. Such a device would be nonintrusive in the sense of being self-contained, requiring minimal physical access and no reconfiguration of the system being monitored. The power distribution networks of many systems, e.g. cars, airplanes, and buildings, are particularly suitable for nonintrusive monitoring because wires carry power from a centralized location to the individual components of the system. A monitor installed at this centralized location might infer the operating conditions and "health" of the overall system by decoding the transients created by the component loads.

A diagnostic system of this scope might be realized as a specialization of the Nonintrusive Load Monitor (NILM) presented in [41, 40, 38, 33, 43, 53, 59]. Prototypes of the NILM can associate electrical transients on a power distribution network in real-time with individual loads using relatively sparse measurement schemes. With an appropriate collection of models and methods, these transients might be further analyzed to obtain physical parameters for individual loads. Given physical parameters, load diagnosis could be as simple as establishing nominal values and acceptable toler-

ances.

1.1 Contribution

The tasks required for nonintrusive load diagnostics, as outlined above, are drawn from mature fields. Because of their importance and broad application, numerical optimization algorithms have been widely explored. However, in the context of unattended nonintrusive load diagnostics, on a platform designed to handle many different models, there are unique issues.

A primary problem is that the iterative optimization techniques often used to estimate parameters require a good initial guess. Conventionally, a good initial guess is supplied by an experienced analyst's rough calculations or estimates. The analyst may even attempt to refine an initial guess in response to algorithm failure. This is not practical for an on-line device designed to handle a range of systems. Another aspect of the problem unique to the NILM environment is the availability of data from the pattern matching process. When a transient is matched by the NILM, certain scaling parameters are available. These may be of use in formulating an initial guess.

Ease of model specification is essential to make a nonintrusive diagnostic platform applicable to a wide range of systems. In solving specific problems, analysts often exploit properties of the system to modify the form of the minimization for improved performance. The resulting estimators are model-specific. Examples are plentiful and include [58, 12, 1, 47, 26, 49]. Model-specific estimators may offer tremendous advantages. For example, a separable nonlinear system can be integrated analytically to find an exact time-domain solution. This may result in a *linear* estimation problem or a problem that can be linearized for initial, approximate solution. To minimize operator effort in nonintrusive diagnostics, such problem-specific circumstances cannot be exploited. Hence, Jacobians must be approximated by finite differences and differential equations must be integrated numerically. System identification under these conditions is computationally expensive.

This thesis provides new tools for enhanced nonintrusive monitoring and system identification, creating a framework for nonintrusive diagnostics. These tools are demonstrated in selected field applications.

1.2 Organization

An implementation of nonintrusive monitoring is presented in Chapter 2, beginning with an overview of prior work in the area and culminating in a description of the new system. The description in Chapter 2 includes AC and DC monitoring systems. Chapter 3 presents the system identification and diagnostic theory, including some demonstrations of the methods on simulation data. Chapter

4 covers field applications and results, including field performance of the nonintrusive monitor, system identification, and system identification combined with monitoring. As there are many field experiments to report, experiments, models and results are presented together for clarity. Finally, Chapter 5 summarizes and gives recommendations for future work.

Chapter 2

Nonintrusive monitoring

Inexpensive, simple to install, and centralized monitoring of electric power systems is a very attractive concept. In addition to providing a starting point for diagnostics, data obtained from such a monitoring system might help building and plant managers to reduce electric power consumption and pinpoint loads introducing undesirable harmonics. For small installations or one-time problem solving efforts, nonintrusive diagnostic or monitoring capabilities might be offered as value-added services by utilities.

The conventional alternative to nonintrusive methods is the instrumentation and monitoring of individual loads at a target site. Individual submetering requires a large number of expensive transducers and a means for collecting the data from these transducers to a central location. In a car, airplane or other transportation system, the cost, weight, and complexity of the submetering approach limits its appeal. In a building level power distribution system, the physical intrusion caused by submetering in an actively used building might outweigh the value of the data collected.

The contribution of this chapter is an open, general platform for nonintrusive load monitoring and subsequent diagnostics called the nonintrusive transient classifier (NITC). NITC is more general than previous NILMs, e.g. [41, 28], in that it directly addresses transient classification in both AC and DC systems. Abstraction between the kinds of measurements and preprocessing needed for different kinds of electric power distribution systems is maintained throughout. NITC is also an open-ended system, specifically designed to adapt gracefully to both anticipated and new power transient monitoring and classification problems by interfacing easily with other programs. Another advantage of the approach in this chapter is that the system is implemented entirely with low-cost, off the shelf hardware, in contrast with the systems discussed in [32, 33, 39, 41, 38].

This chapter reviews previous work in nonintrusive load monitoring in two broad categories; simple change-of-mean detection and more sophisticated multi-time scale, transient-based approaches.

NITC falls into the later category, and operational details of NITC follow the review of prior work. The chapter concludes with a discussion of conditions under which NITC and prior load monitors are roughly equivalent.

2.1 Background

Work in nonintrusive load monitoring can be divided into two approaches. The steady-state approach localizes and labels load events according to their characteristic changes in steady-state power consumption. In the transient approach, events are labeled according to the shape and structure of the transitions between steady-states.

One difficulty with the transient approach is the possibility that the details of transitions depend heavily on unreproducible factors, for example, the exact phase in the voltage cycle when a load is switched on. Except in specific circumstances, e.g. zero voltage switching solid-state relays, it is difficult to make statements about transient reproducibility with a few measurements. An argument for the transient technique is that transients for physically different loads with similar steady consumption are often distinct [38].

2.1.1 Steady-state approaches

In the steady-state approach, loads are distinguished by their steady state power consumption. A residential refrigerator consuming 250W and 200 VAR is given as an example in the description of the "Nonintrusive Appliance Load Monitor" or NALM [28]. If the NALM observes a change of 250W and 200 VAR in the net power consumption of a residence, it identifies a refrigerator. Generally, observed step changes in real and reactive power are identified by their proximity to clusters in the "signature space" ΔP, ΔQ associated with individual loads. The signature space technique reduces the potentially complex, rich data in load transients to a two-dimensional space of changes in steady-state power consumption, as illustrated in Figure 2-1. This space has a nice graphical interpretation, but loads that generate adjoining clusters are difficult to distinguish.

In a residential environment the NALM was quite successful, to the point of motivating a paper on nonintrusive monitoring and privacy [27]. An example in [27] was the impressive discovery via NALM of whether a waterbed was covered or not based on the duty cycle of its electric heating apparatus. The NALM was strongly advocated in [8] as a technology ripe for commercialization, and a NALM is offered as a product by Enetics incorporated of Victor, New York. Acronyms used for essentially identical techniques include NALM, NIALMS, and SPEED.

An interesting approach extending the steady-state paradigm by adding a slope (or slopes) to the turn-on part of a transient can be found in [15]. The findings of [15] are tempered, however, by the

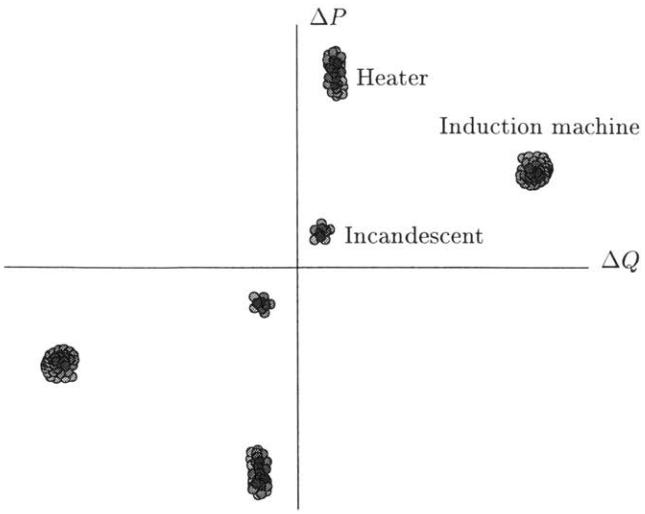


Figure 2-1: A schematic "signature space" ΔP, ΔQ as given in [28]. Possible turn-on clusters are labeled; corresponding turn-off clusters are reflected through the origin.

failure of the authors to correctly describe the method in [38]. A superlative summary, verification and presentation of the steady-state approach is found in [54].

2.1.2 Multi-time scale transient approach

One way to handle the shortcomings of steady-state data is to use transient information to distinguish loads. This is the approach adopted in [38] and subsequent publications [32, 33, 39, 41, 53].

The multi-time scale transient approach as presented in [38] characterizes load transients using relatively high-derivative portions, or v-sections, of the envelopes of power associated with the transient. This "training" step might be done off-line in Matlab, for example. Figure 2-2 gives a graphical interpretation of v-sections [38]. Detection involves matching stored v-sections to observed data, and then collating these matches into strings that are characteristic of particular loads. The method is similar to language interpretation. The first step of "lexing" or "tokenizing" corresponds to comparison of individual v-sections to the incoming data. The second step, recognizing sequences (even interleaved sequences) of tokens as transient events is parsing [45]. It is important to note in this scheme that the transient event is identified based on a stream of tokens which are admitted based on their individual quality of fit to the observed data. This is distinct from admitting a transient event based on the collective fit of the v-sections to the data.

In addition to using transient data, [38] describes a technique in which incoming data is resampled for matching transients on different time scales. This is shown in Fig. 2-3. In Fig. 2-3, a series of resampling operations results in data on several time scales with respect to the reference patterns.

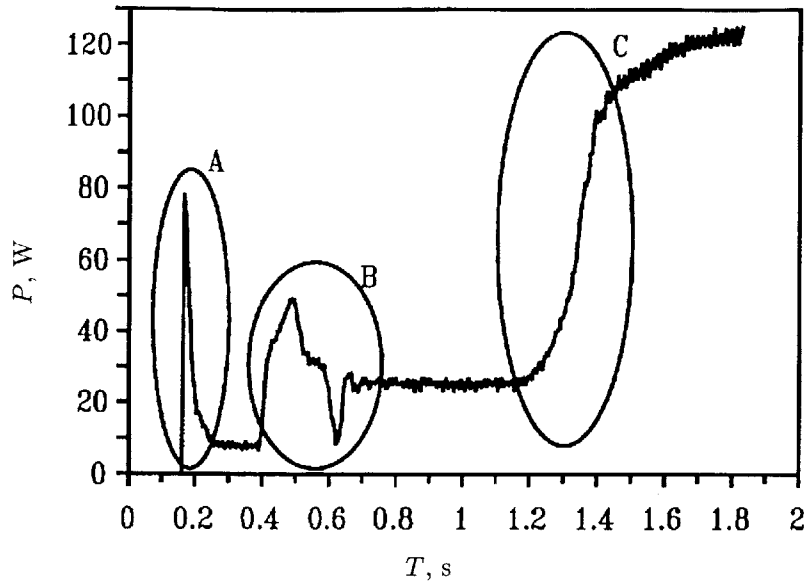


Figure 2-2: Spectral envelope input data stream is characterized by high-derivative "v-sections" in [38]. Figure is from [38].

Attempts to match the reference patterns on several times scales are reconciled before choosing the time scale on which a particular pattern matches. In effect, a time dilation parameter for each pattern is found by exhaustive search.

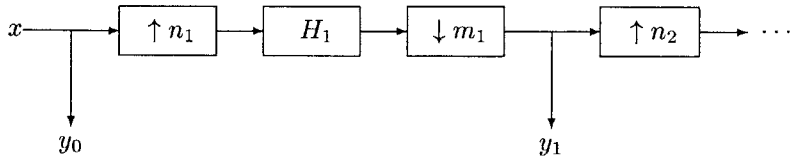


Figure 2-3: Decomposition from [38]. The H_k are filters appropriate to the up and down-sampling rates n_k and m_k. Pattern matching is performed on the new streams y_k.

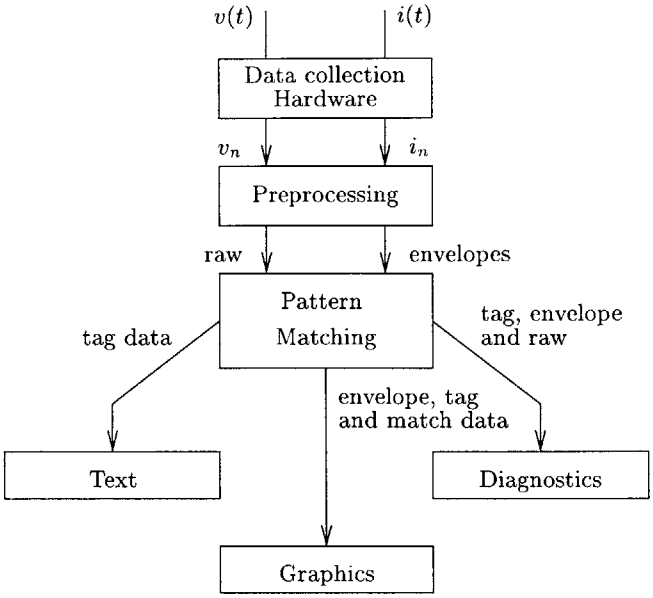


Figure 2-4: Nonintrusive transient classifier (NITC) block diagram. The NITC main program preprocesses, pattern matches, and schedules data for output on three data queues. Each data queue supports multiple streams associated with processes requiring similar kinds of data from the classifier.

2.2 The nonintrusive transient classifier

The nonintrusive transient classifier developed here is designed to perform the tasks of prior load monitors while offering a flexible base for innovation and improvement. This flexibility is in part required to apply the system identification techniques of Chapter 3, but the generalization given in this section offers other important advantages in terms of implementation cost, data recovery and presentation, and adaptability to specific problems.

An overview of the nonintrusive transient classifier is given in Fig. 2-4. Following the flow of data, selected current and voltage signals are collected with hardware and device drivers installed in the host computer. A small cache is typically associated with this device driver data collection layer. Data from the device driver enters user space when read by the preprocessing routines. Alternatively, the transient classifier can be run off-line by collecting "raw" data from a device driver, storing the data in a file, and reading the data off-line from that file. There is essentially no preprocessing other than data format conversion and error checking (depending on the device driver) in a DC environment, but in an AC environment spectral envelope estimates [43] are calculated. Data in the preprocessing layer is accumulated and processed in blocks, which then proceed to the pattern matching and dispatch part of the program. AC pattern matching and dispatch differs from the DC case only in the dimensionality of the data; AC data consists of several spectral envelopes as

a function of time, while DC data is a single function of time. Pattern matching is done with respect to a set of templates or *exemplars* that are designed separately. Finally, contacts classified by comparison of the input data to the exemplars are placed in output queues.

Output queues are of three types; *tags*, *graphics* and *diagnostics*. The type of queue depends on the kind of data it serves to post-processing programs, any number of which may be connected via independently buffered streams. Programs connected to tag streams receive a simple text string giving the time of contact, classification, scaling and error information. This string is easily parsed in a language like Perl [68]. Graphics stream data contains information necessary to produce an attractive graphical representation of the contact and its matching exemplar, including a copy of the text tag. The principle purpose of graphical streams is to provide output to users, but more sophisticated post-processing programs might also use the same data. Diagnostic streams contain text tags and raw current and voltage measurements synchronized with the preprocessed spectral envelope data used to match the exemplar.

Streams associated with the queues in Figure 2-4 have deep buffers, and serve data to attached programs in a "non-blocking" manner. Data is provided to an attached process only when the process is ready to read, which leads to smooth overall operation. Computationally intensive attached processes that run infrequently, e.g. a diagnostic program using iterative minimization, should not stop the flow of data to less computationally intensive processes. With deep buffers and enough speed, the data streams are roughly synchronized because "fast" processes run out data and surrender their time-slice to the "slow" processes that have not yet run out of data. In specific cases, it may be desirable to reprioritize processes attached to the classifier using `nice` [21].

A typical invocation of the NITC might involve the classifier, a Perl script to add up power consumed and count the number of contacts for each type of exemplar, a program to display data connected to a graphics stream, and a diagnostic dispatch program connected to a diagnostics stream. In turn the diagnostic dispatcher might execute several specialized identification programs and a graphics application to display results from these programs. In addition to dividing the overall task into manageable programs and providing "hooks" for as-yet unanticipated roles, the open structure of the NITC encourages a coarse parallelism that is adaptable to commercially available symmetric multiprocessing chipsets and motherboards.

2.2.1 Preprocessing in AC systems

Preprocessing for nonintrusive load monitoring using spectral envelopes is discussed extensively in [38, 42, 59]. Spectral envelopes are short-time averages of the harmonic content of a signal, e.g., a

signal $x(t)$ of current observed by the NILM. The in-phase spectral envelopes a_k of x are

$$a_k(t) = \frac{2}{T}\int_{t-T}^{t} x(\tau)\sin(k\omega\tau)d\tau, \tag{2.1}$$

where k is the harmonic index. Similarly, the quadrature spectral envelopes b_k are

$$b_k(t) = \frac{2}{T}\int_{t-T}^{t} x(\tau)\cos(k\omega\tau)d\tau. \tag{2.2}$$

These spectral envelopes are the coefficients of a time-varying Fourier series of the waveform $x(t)$ [43].

For transient event detection on the AC utility, the time reference is adjusted so that the term $\sin(\omega\tau)$ in (2.1) is phase-locked to the voltage measurement. The averaging interval T is typically one or more periods of the fundamental frequency of the voltage waveform. For a_k, b_k computed under these conditions, the spectral envelopes are called $P_k = a_k$ and $Q_k = b_k$. Notice that steady-state spectral envelopes P_1 and Q_1 correspond to the conventional definitions of real and reactive power, respectively.

Historically, the spectral envelope preprocessing calculations have been implemented on hardware separate from that used for pattern matching. In [38, 41], the preprocessor consisted of phase-locked oscillators, analog multipliers and analog low-pass filters estimating first, quadrature, and third harmonic spectral envelopes. The low-pass filters approximate the integrals in (2.1) and (2.2). In [32, 39], the preprocessor was improved to sixteen channels with digitally reconstructed basis functions. In [59], a preprocessor using a digital signal processor was reported.

Personal computer-type computing platforms are decreasing in cost and increasing in performance. In addition, many processors incorporate single-instruction, multiple-data (SIMD) instructions that offer DSP-like performance. In view of these changes, a natural extension of [59] is to combine preprocessing and pattern matching tasks on a single platform. The high performance and specialized architecture of the DSP in [59] was used to implement the preprocessor by simulating previous analog preprocessors. A more efficient approach is to process current and voltage data in blocks, using a short-time Fourier transform approach as in Algorithm 2.1.

Algorithm 2.1 *Procedure for computing spectral envelope estimates via short-time Fourier transform.*

1. *obtain array of sampled voltage and current measurements, $\underline{v}$ and $\underline{i}$*
2. *filter and resample $\underline{v}$, $\underline{i}$ so that each period in $\underline{v}$ contains 2^N samples*
3. *concatenate resampled data to arrays v, i*

4. compute FFT on first 2^N points of v, i to obtain V and I

5. apply rotations to I and output selected envelope estimates

6. shift v and i

7. repeat 4 until v, i contain less than 2^N points each

8. repeat 1

Steps 5 and 6 in Algorithm 2.1 deserve further explanation. While step 2 resolves frequency so that a particular number of samples corresponds to one period of the input, the phase of the voltage relative to the index is not determined. After the FFT, it is necessary to apply a corrective rotation to accommodate this phase so that the spectral envelope estimates behave as expected. If V_1 is the complex first harmonic coefficient of the voltage, define the phase

$$\phi = \frac{V_1^*}{\|V_1\|}. \tag{2.3}$$

The appropriate rotation for the k'th spectral envelope estimate is ϕ^k, i.e.

$$a_k + jb_k = \phi^k I_k, \tag{2.4}$$

where $j = \sqrt{-1}$. This is the rotation in Step 5, and can be confirmed by considering the case where the current is an in-phase copy of the voltage.

The "shift" in Step 6 controls the relative sample rate of the spectral envelope estimates. For example, if the shift is 2^N points, the estimates are supported by non-overlapping blocks of current and voltage data. If the shift is 2^{N-1} points, the sampling rate of the spectral envelope estimates is effectively doubled.

Figure 2-5 illustrates the resampling, concatenation and shifting operations described in Algorithm 2.1 by following an array of new voltage measurements. New measured data is filtered and resampled so that there are 2^N samples per period. The resampled data in $\underline{v}$ is concatenated to the data in v left over from the previous iterations. The remnant in v is less than 2^N points. Although not shown in the figure, a set of spectral envelope estimates is computed using the first 2^N samples in v. Finally, v is shifted, in this case by 2^{N-1} points. The last two steps are repeated until less than 2^N points remains in v, then the process is repeated with new set of measurements.

The spectral envelope estimates computed using Algorithm 2.1 are not precisely the same as obtained using (2.1) or the various analog preprocessors, e.g. as in [43]. However, the estimates retain the qualitative features useful for pattern matching. Figures 2-6, 2-7 and 2-8 show typical short-time Fourier transform spectral envelope estimates.

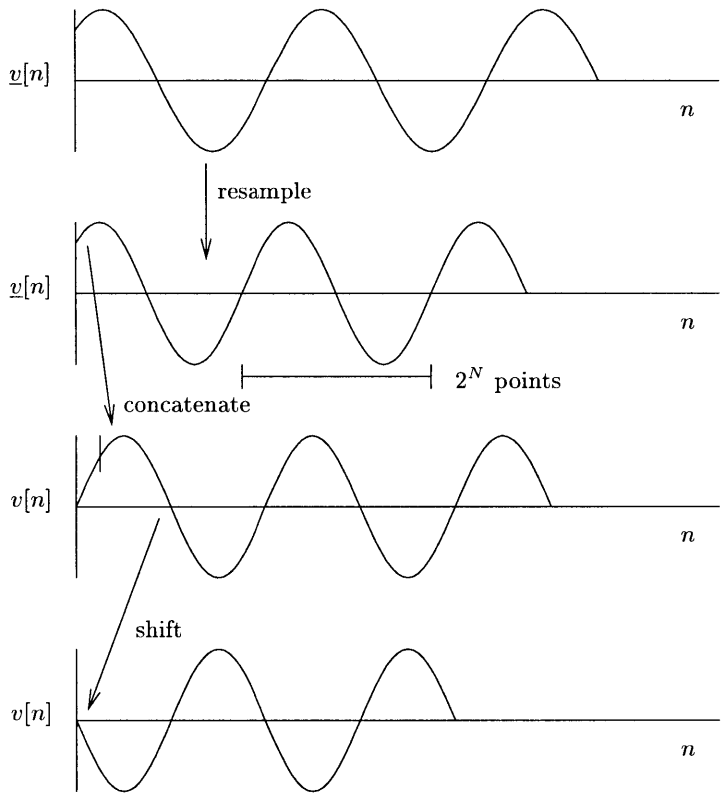


Figure 2-5: Illustration of operations in Algorithm 2.1 for a set of voltage measurements.

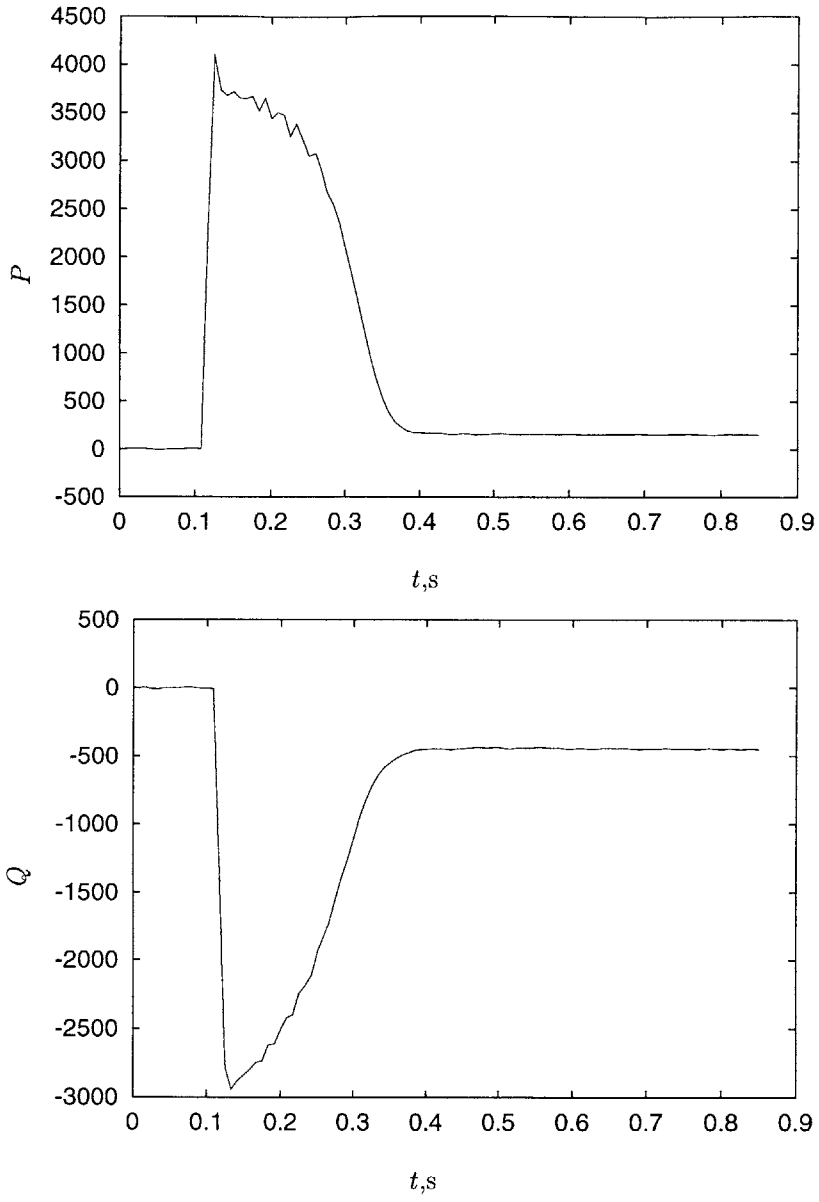


Figure 2-6: Spectral envelope estimates of P and Q for a fractional horsepower induction machine. At the beginning of the transient, the machine draws power to accelerate the rotor. When rotor comes up to speed, considerably less power is consumed. Scaling of the vertical axis is arbitrary. For P the scale is about 9 units/W, for Q 9 units/VAR, etc.

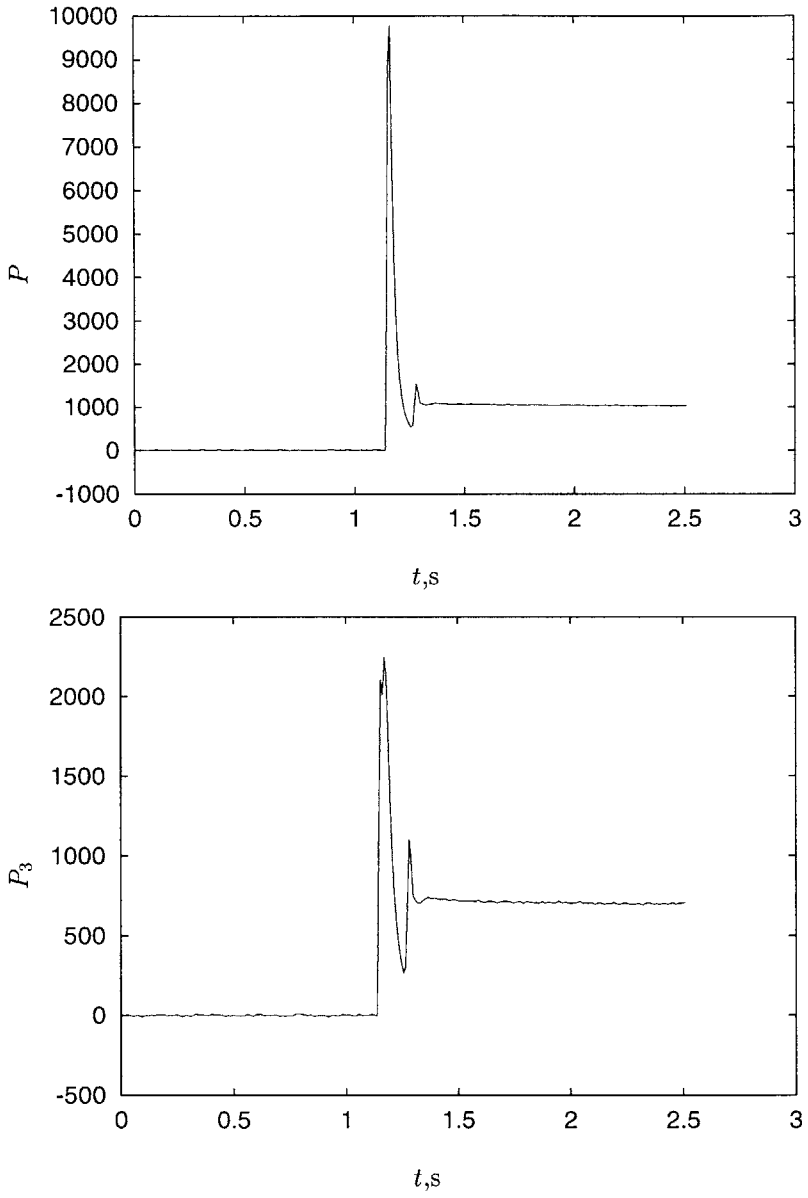


Figure 2-7: Spectral envelope estimates of P and P_3 for a rapid start fluorescent light bank.

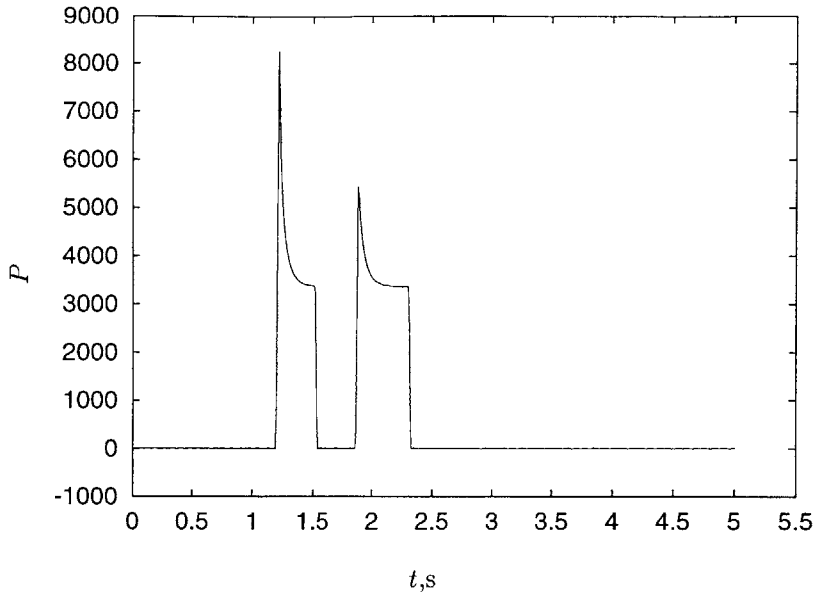


Figure 2-8: Spectral envelope estimate of P for a 400W incandescent light bulb, showing two on-off cycles in rapid succession. The first transient has a larger peak because the filament in the bulb was initially at room temperature.

2.2.2 Preprocessing in DC systems

Spectral envelope preprocessing is not necessary in a constant voltage DC system where power is directly proportional to current. However, in the DC power distribution system of the automobile considered in Chapter 4, certain disturbances and undesired signals were eliminated through careful measurement selection. These preprocessing steps apply to the specific situation, and are discussed thoroughly in §4.1.1.

2.2.3 Activity location and pattern matching

The general procedure for activity location and pattern matching is diagrammed in Fig. 2-9. Figure 2-9 shows data streams for P and Q, stacked vertically. As data arrives, the classifier shifts the data to the left in chunks called *stages*. A stage is nominally 1000 points, although this can be changed to any value (within the limits of memory). At any given time, the classifier holds five stages. Incoming data are matched against *exemplars*, or templates, which consist of one or more *sections* that characterize the transient. Matching begins when an event corresponding to an *index section* is detected in the index stage. If an event in the index stage is detected, the list of exemplars is scanned to find the first exemplar with the best match to the data stored in the stages. Other sections in the exemplar need not occur after the index section, nor need they be undetected by the event detector. Parameters varied to match the exemplar to the incoming data stream include a gain, which applies to all the sections, a "dc-term" associated with each section, and a delay or advance relative to the time when the index section was detected for each section. Sections need not consist of an uninterrupted sequence of points; a single section can be supported by an arbitrary collection of times. A section with two regions of support differs from two sections in that two sections would be fit with different offsets. The last section in the exemplar shown in Fig. 2-9 is an example of a section with two regions of support. If an exemplar is accepted as a fit to the incoming data, events associated with its sections are "pruned" from the list of detected events. Exemplars are searched from most complex to simplest, so that small exemplars do not accidently prune events belonging to large exemplars that have not had a chance to match.

Events are detected using a typical change-of-mean scheme. In particular, if the absolute value of the difference between the input and a FIR low-pass filtered version of the input exceeds a threshold, an event has occurred. Once an event has occurred, events are locked out for a small time period. This does not interfere with exemplar fitting, because the sections in the exemplars are allowed to "settle in" and find an optimal offset in time; the event detector only indicates where the search for a match should begin.

Fitting the exemplar to data is approached in an approximate manner as a series of a decoupled estimation problems rather than one large problem. Each section is characterized by an index t giving the offset relative to the index section, and a shape vector s giving the shape of the section. Assuming the event detector signals an event at time t_e, and that the spectral envelope data on the channel associated with the section is labeled d, an individual section is matched by solving the least squares problem

$$\begin{pmatrix} 1 & s \end{pmatrix} X = \begin{pmatrix} d[t_e + t - k] & d[t_e + t - k + 1] & \cdots & d[t_e + t + k] \end{pmatrix} \tag{2.5}$$

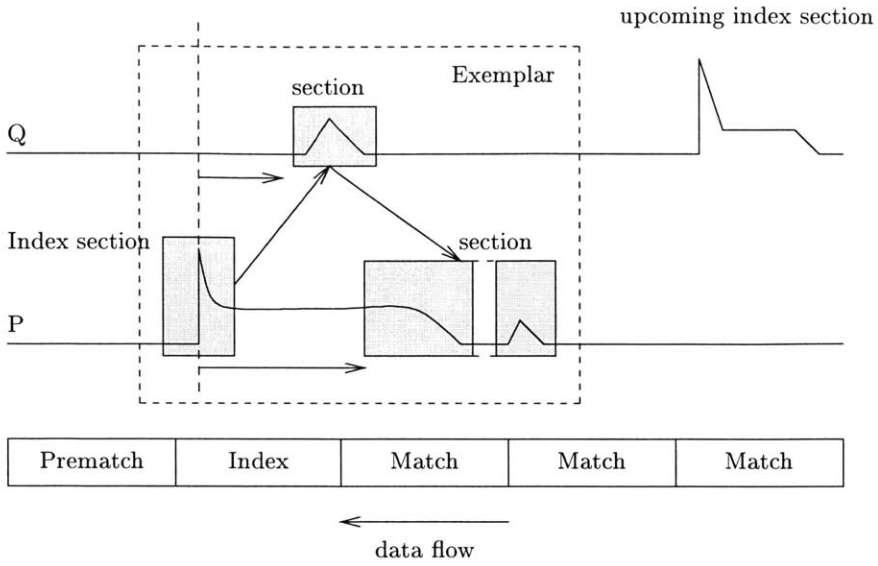


Figure 2-9: Activity location and pattern matching scheme. Data are shifted from right to left. Matching is attempted for events that have been shifted into the "index" stage. When attempting to match an exemplar, the sections of the exemplar can be supported by data before or after the index stage, as indicated by the shaded boxes. The index section determines the origin in time when matching a multi-section exemplar.

where k is a constant that defines the aperture allowed for the section to "settle in." Note that each of the items, e.g. $d[t_e + t - k]$, on the right hand side is a column vector, and that X is a matrix with $2k+1$ columns and two rows. The final shift of the section is determined by picking the column r of the right hand side and corresponding column x of X that offers the best least-squares fit. This is done for each section in the exemplar, yielding a collection of vectors x_k and r_k. The entire exemplar is fit by determining a gain that applies to all the sections. This is done by solving the least-squares problem

$$\begin{pmatrix} s_1 \\ \vdots \\ s_N \end{pmatrix} a = \begin{pmatrix} r_1 - x_1[1] \\ \vdots \\ r_N - x_N[1] \end{pmatrix} \tag{2.6}$$

where s_k is the shape vector corresponding to the k'th of N sections in the exemplar.

2.2.4 Exemplar design

Exemplars are stored in ASCII files, and can be created using software like Octave. The Octave script `vsection` in §C.3 allows a user to specify sections of an exemplar interactively using menus. Collected data are ordinarily used as a starting point for exemplar design, and are input to `vsection` as a file.

The behavior of the NITC depends on the design of the exemplars. Exemplars can be specified to capture the "v-section" concept of [38], or the regions of support can be modified so that the NITC works essentially with the steady-state information used in [28]. One approach that seems to work well in practice is to collect several transients per load to determine empirically what parts of the transient are reproducible. The reproducible parts of these transients are good starting points for sections in the exemplar characterizing that load.

Generating good exemplars by clustering or otherwise processing observed data is a promising area for future work.

2.2.5 Postprocessing

As indicated in Fig. 2-4, NITC provides three kinds of output data streams available for user post-processing; tag streams, graphical streams, and diagnostic streams. This modular approach makes the coarse parallelism of the NITC explicit to the kernel and provides a flexible means of expansion and modification. Diagnostic streams are discussed in Chapter 3.

The tag stream is especially suited for add-on, user-written postprocessing programs. A tag stream is a series of ASCII messages, one line per matched exemplar, giving information about the matched exemplar. The format is

```
[time] : [user field] : [vertical scale] : [residual].
```

Depending on the contents of the user field, of course, there may other colon-delimited fields in any particular tag. The user field would typically contain a name associated with the exemplar, e.g. "incandescent lightbulb." With a tag stream, a variety of useful programs can be written. An example is Program 2.1, a Perl script that keeps track of all exemplars matched.

Program 2.1 *Perl program to summarize matches of exemplars.*

```
#!/usr/bin/perl                                                    0
# -*-Perl-*-

%howmany = ();

while(<>) {
    ($time,$name,$otherstuff) = split(':',$_,3);
    $name =~ s/ //g;
    $howmany{$name} = $howmany{$name}+1;
    print join(' : ', $name, $howmany{$name}),"\n";
}                                                                  10
```

The output of Program 2.1 is a colon-delimited stream consisting of the exemplar user field and the number of occurrences observed.

The graphics stream is more complicated than the tag stream. The graphical stream format consists of tags, as in the tags stream, alternating with graphics blocks. A graphics block consists of a 32-bit integer, M, followed by M graphics sections. A graphics section consists of a 32-bit integer N followed by $2*N$ 32-bit floats. The first N floats are abscissa and the subsequent N are ordinates for each section. For example, a typical AC session operating on eight spectral envelope estimates would have a graphics stream with $M = 16$; for each match, raw spectral envelope data and an "overlay" showing the fit of the exemplar to the data would be included.

Two graphics stream applications, `xnilm` and `w3nilm`, were written to help users interact with data from NITC and the diagnostic codes. Both programs read data from the graphic stream, display the tag data in a window, and store a limited number of exemplar matches in an associative array. Users can access graphical data for a particular exemplar match by selecting its tag. Actively selecting interesting tags tends to prevent the "overload" of watching a live display flash by, and is helpful in studying how exemplars match. The programs differ in that `xnilm` uses the X Windows system for display, while `w3nilm` uses a graphical web browser. On a Unix system there is little practical difference between the two approaches – both can be used remotely over a network. With non-Unix platforms, however, `w3nilm` allows remote use of NITC over a network. Also, `w3nilm` allows several client connections at one time. For example, two people could watch one NITC process identify loads from different locations at the same time.

The implementation of `w3nilm` is a bit more involved that `xnilm`. When invoked, `xnilm` pops up a new window and commences using it for display. In contrast, `w3nilm` begins listening for connections on an unprivileged port on the host machine while building an in-memory database of HTML pages and graphics based on the incoming data. When a client (such as Netscape) connects, it is added to a list of active clients and served content from the database. As `w3nilm` is its own HTTP server, it can be used on a very spartan system with no conventional httpd daemon.

Figures 2-10 and 2-11 show screen shots of `xnilm` and `w3nilm`, respectively.

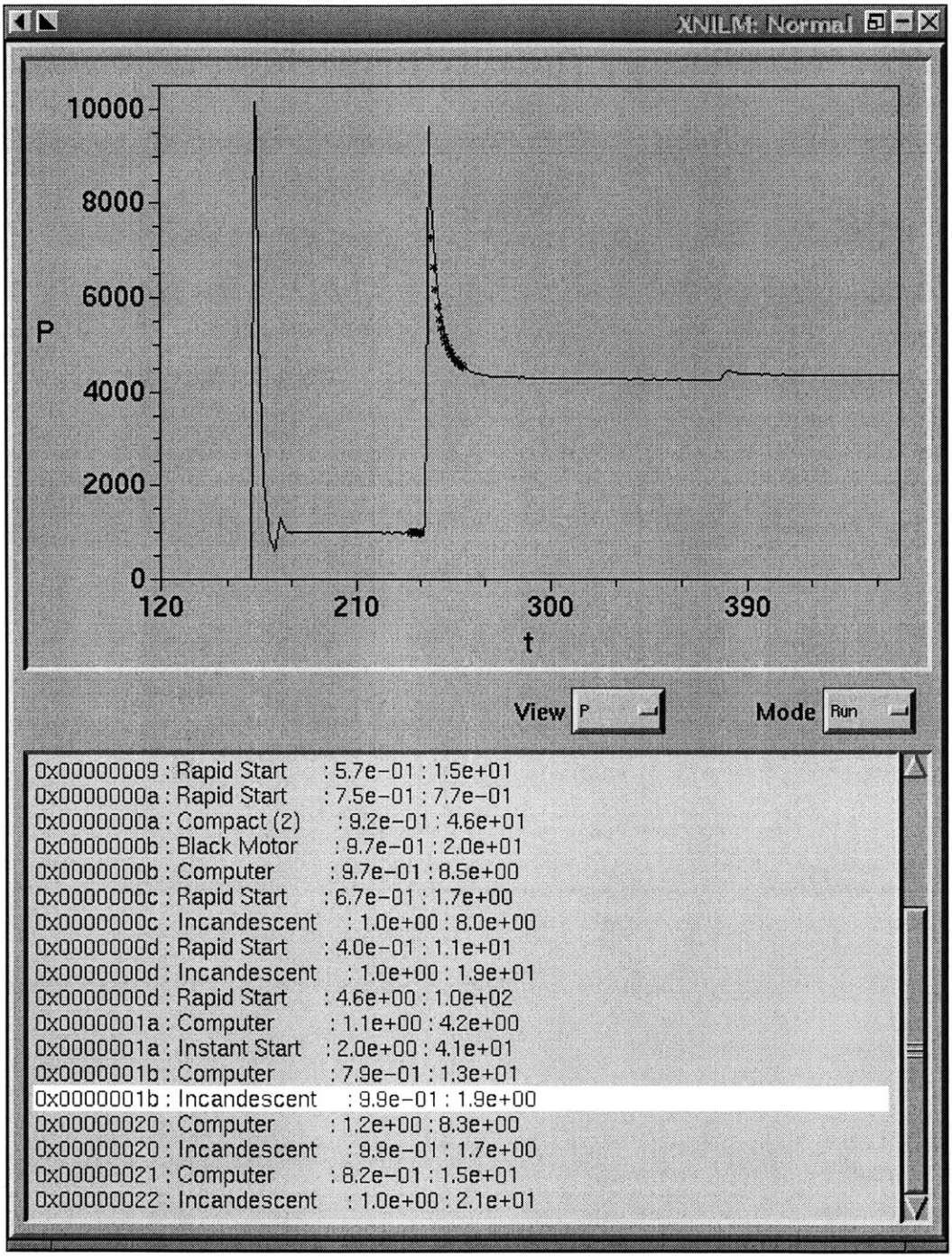


Figure 2-10: Screen shot of `xnilm` showing detection of incandescent lightbulb transient. Solid lines in the graph are spectral envelope data; the overlayed data points show the fit of exemplar to transient.

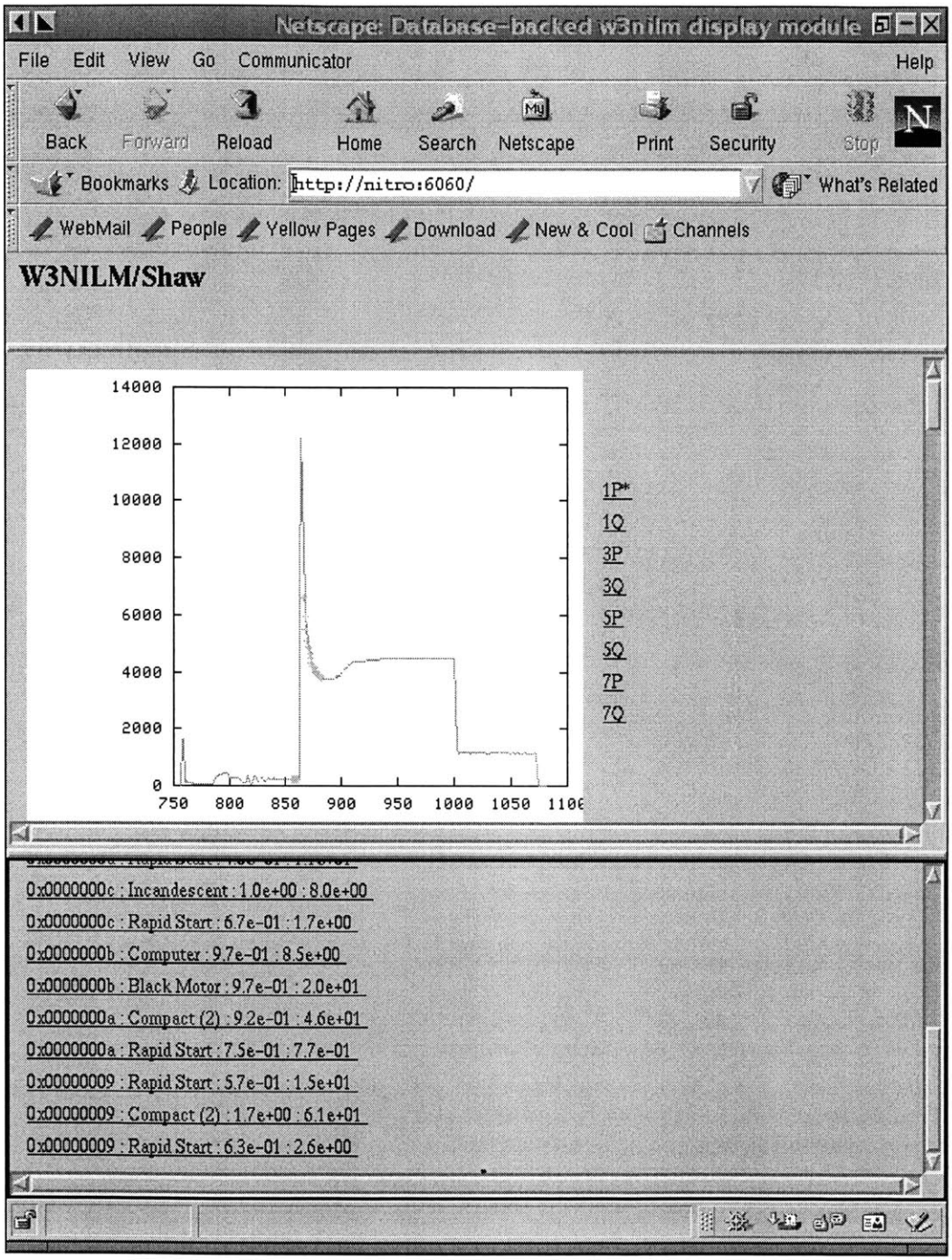


Figure 2-11: Screen shot of `w3nilm` showing detection of incandescent lightbulb transient.

2.3 Summary

Collectively, the tools presented in this chapter preprocess, classify, queue, dispatch and display transient data from AC or DC power systems.

The classifier presented in §2.2 can be made equivalent or nearly equivalent to previous classifiers. For example, exemplars can be specified so that they detect and discriminate only based on steady-state changes. This could be done by disabling vertical scaling and selecting only flat portions surrounding a transient when training. The multi-time scale approach described in [38] can be achieved by disabling the time adjusting mechanism and creating a new exemplar file consisting of the original exemplar file resampled at the desired time scales. In [43], for example, three time scales were considered.

In addition to capturing the rough behavior of previous classifiers and providing expansion options that are exploited in Chapter 3, the tools of this chapter have several unique advantages.

As a software based system, the classifier, preprocessor and associated tools leverage technological improvements in the computer industry. In contrast to previous monitoring efforts, the tools described in this chapter require only hardware that can be bought off the shelf. It is very advantageous to use mass-market computing products where prices drop rapidly as performance increases. In the time used to write this thesis, the hardware needed for a basic installation has dropped in price by nearly a factor of five. In addition, while development and debugging are done at a workstation, field installations can take advantage of commercially available miniature and rugged platforms.

The tools in this chapter address the problem of how to access and use extracted data. All of the user output modules are web accessible. Unix platforms may use X-windows, but any machine with a network connection and a suitable web browser can be used to examine results. Previous efforts required use of a dedicated console, periodic modem transmission of data, or even physical visits to monitoring sites to download data. Use of a network does not preclude these simpler strategies, however.

As modules, the tools in chapter are individually simple and thus independently tested and debugged. The modular structure also allows the kernel to transparently employ multiple processors.

Chapter 3

System identification and diagnostics

System identification is the problem of finding a descriptive model or model set for a system given observed data. Diagnostics involves interpretation of observations for the purpose of discriminating between normal and failed or about-to-fail systems. Failure modes and their manifestation in observations of the system may be complicated. In medicine, for example, this is certainly the case. Electrical loads, on the other hand, are designed and applied according to physical, electromechanical models. Design intuition and physical reasoning might be exploited, therefore, in a diagnostic approach involving the identification of a physical model.

The first two sections of this chapter give basic context for the following sections. In particular, the first section introduces notation and motivates the basic computational problems addressed in §3.3 and §3.4. The second section provides context and background for the proposed diagnostic technique. Beyond parameter estimates, the diagnostic aspect of the problem requires estimation of the size of the model set – the range of models that might also describe the data. This is the topic of §3.5.

3.1 System Identification

System identification is generally one aspect of a complicated problem. A general view of the context of system identification is shown in Fig. 3-1. The purpose of system identification is to find a model or model set describing the given data. This usually amounts to estimating the parameter values of a model. Validation is the process of assessing the usefulness and applicability of the identified model to the task at hand, and includes such tasks as confirming that the residuals meet the

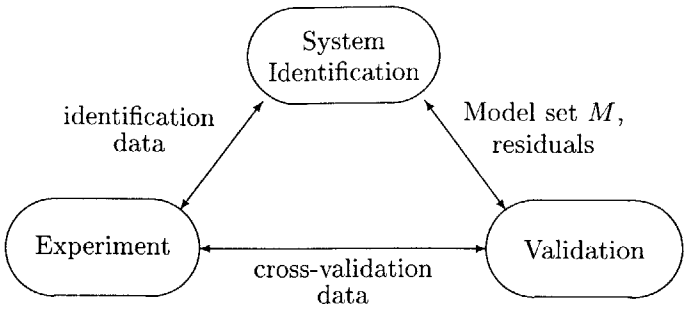


Figure 3-1: Relationships between experimental design, validation, and system identification.

assumptions of the system identification method and checking that the model generalizes to data which was not used in identification (cross-validation). Validation and identification steps often interact iteratively. For example, identification may supply a model based on *a priori* information about the disturbance in the measurements. Residual analysis may reveal that these assumptions are incorrect, and motivate identification of the system with a different model. A practical example is pseudo-linear regression. Similarly, identification and validation may interact iteratively with the measurements or experimental system. For example, if the goal of the identification is to find parameters of a particular model structure, that goal may be impossible based on the selection of measurements and require a different experiment. Similarly, validation steps like residual tests may suggest changes in the experiment or measurement procedures. Excellent overviews of these tasks and their interaction can be found in [46, 64, 63, 30]. In addition, the abstract perspective of Fig. 3-1 emphasizes the connection between black-box approaches that validate an arbitrary model and classical techniques using a physical model. Unification of these areas has been quite fruitful and the subject of a seminal overview, [61].

In the context of nonintrusive diagnostics, the situation in Fig. 3-1 can be made more specific. In particular, the details of the experiment are fixed – the measurements must be "nonintrusive", which typically means a small set of current and voltage transducers at an aggregated, central location in the power distribution system. The validation step must ensure that identified models are useful for diagnostics, as discussed in §3.2 and §3.5. Finally, the system identification step will be constrained to models drawn from physically based model structures.

A fairly general structure capturing many physical models is a set of differential and output equations

$$\begin{aligned} \dot{x} &= g(x, u; \mu) \\ y &= f(x, u; \mu) \end{aligned}$$

$$x(0) = i(\mu) \tag{3.1}$$

where the state equation g and output equation f are possibly nonlinear. The function i may be used to parameterize the initial state. In the previous equations, x is the state, which is assumed to be unmeasured, u is the input, which is assumed to be known, and y is the output, which is assumed to be measured. The quantities x, u, y and μ are all potentially vectors. Identification of this structure amounts to finding the parameter vector μ given measurements, which is generally a nonlinear problem. This thesis makes the imprecise assumption that the systems involved arise from physical models and are generally well behaved.

As a concrete example, functions g and f might describe an induction machine. The input u could be the voltages applied at the stator. The state x might include shaft speed, stator and rotor fluxes. The output y could be the currents drawn at the stator. From measurements of u and y, the problem would be to find the parameter vector μ, which might include the lumped rotor resistance, the mass of the rotor, etc.

Classically, finding μ is expressed as a minimization of some figure-of-merit or loss-function $V(\mu)$, i.e.

$$\hat{\mu} = \arg\min_{\mu} V(\mu). \tag{3.2}$$

The notation $V(\mu)$ indicates that the loss function depends *at least* on μ, which is the argument of interest in (3.2). Of course, the loss function also depends on the measurements, inputs etc. Assuming that the model is correct and the inputs are well known, or alternatively that the disturbances in the input have little effect, a typical choice is to minimize the output errors in the least squared sense, i.e.

$$V(\mu) = \frac{1}{2} r' r, \tag{3.3}$$

where the residual

$$r = y - \hat{y}|\mu \tag{3.4}$$

and $\hat{y}|\mu$ (sometimes abbreviated $\hat{y}$) is the estimate of the output, given the parameter choice μ, the model, and the input. The observed output of the system is y. If the disturbances in the measurements are normally distributed, zero-mean, independent and identically distributed, the likelihood function for the entire residual is

$$\begin{aligned} \mathcal{L}(r) &\propto e^{-\frac{r_1^2}{2\sigma^2}} e^{-\frac{r_2^2}{2\sigma^2}} \cdots e^{-\frac{r_N^2}{2\sigma^2}} \\ &= e^{-\frac{r'r}{2\sigma^2}}, \end{aligned} \tag{3.5}$$

where r_k is the k'th element of the residual r and σ is the standard deviation of the disturbance.

The likelihood function has the flavor of a probability. Formally, however, the probability of obtaining a single value from a continuous random variable with finite-valued distribution is zero. To avoid this difficulty some texts construct (3.5) by assuming a uniform uncertainty interval Δ around each element of the residual [55]. These intervals amount to a multiplicative scale factor in front of (3.5) and do not change the conclusion, which is that the parameters minimizing (3.3) maximize $\mathcal{L}$. Further discussion of maximum likelihood estimation can be found in [4, 30].

Independent of the maximum-likelihood statistical interpretation, a least squares estimate has the geometrical property of minimizing the Euclidean distance between the explanation and the observations. Even in cases where the model is inaccurate or the inputs or regressors are noisy, making statistical interpretation difficult, the least squares loss function (3.3) is often used. A more general situation is that the independent variables (e.g. the inputs or time) also contain errors; this is the orthogonal distance regression (ODR) problem [6]. Although ODR is inherently nonlinear, in the linear-model or "total least squares" case a solution via the singular value decomposition is possible and is presented in [23]. A more general discussion with proofs and examples is given by Boggs in [6], where the ODR problem is solved with Levenburg-Marquardt methods. There are algorithms that claim improved performance over Levenburg-Marquardt for sparse nonlinear systems [18], such as arise with ODR. The development in this thesis will be presented in the relatively simple framework of least-squares estimation. The generalization of ODR to unconstrained least-squares minimization of an alternate loss function implies that the methods will be of value in the ODR context as well.

For the minimization of (3.3) to provide useful results, the choice of measured quantities and data should contain sufficient information to determine μ. This identifiability problem depends on the excitation, the observations, and the model structure itself [52]. The situation with respect to choice of measured quantities is illustrated by the example in Fig. 3-2. Even if a model is theoretically identifiable given the quantities measured, there may not be sufficient excitation to accurately find one or more components of the vector μ. In the linear case, this corresponds to a singular or near-singular matrix inversion problem. One solution is to *regularize* the solution. In terms of the loss function, a term $\delta||\mu - \mu_0||$ may be added where μ_0 is a nominal parameter value and δ is small. For example, in Fig. 3-2a only the time constant $\mu_1\mu_2$ can be determined from the measurements. A regularization term $\delta||\mu||$ added to the loss function for Fig. 3-2a would produce a unique, minimum-norm solution.

Minimization of the loss function is the primary computational task in identification, given a model and measurements. If the system is *linear in the parameters*, meaning that the function

$$\hat{y} = h(\mu) \tag{3.6}$$

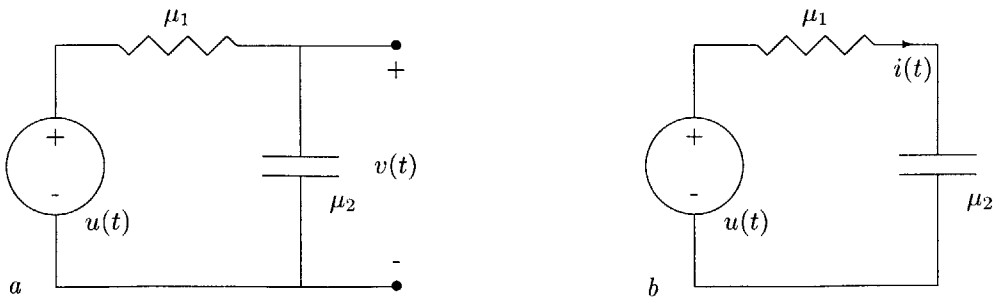


Figure 3-2: The parameters of circuit a are not identifiable with the measurements shown – the only quantity that can be determined is the time constant $\mu_1\mu_2$. The parameters of circuit b are identifiable given measurements $i(t)$ and the step excitation.

can be put in the form $\hat{y} = A\mu$ for a matrix A of known or observed quantities, the minimization (3.3) can be obtained directly as the solution of the linear problem

$$A'A\mu = A'y, \tag{3.7}$$

where y is a matrix of observations. More generally, iterative methods must be used. Note that considerable attention has been paid to iterative methods even to solve the system $Ax = b$. When A is sparse and large, the inverse of A or even a decomposition of A may be too large to store. However, since A is sparse, the product Ax is easy to compute, so x is found by iterative least-squares minimization of $b - Ax$ [19, 55]. Optimization codes for this sort of problem are plentiful because of the importance of finite difference approaches, and enjoy the advantage that the curvature of the loss function is quadratic everywhere. The idea of using the *action* of the operator rather than the operator itself has inspired hybrid methods for nonlinear systems [7].

If the function relating the parameters to the modeled observations cannot be expressed as a linear combination of the known quantities, nonlinear methods must be considered for minimization of the loss function. The standard nonlinear least-squares method is the widely used Levenburg-Marquardt method [57, 55] and its variations [74, 6]. The Levenburg-Marquardt iteration is

$$\hat{\mu}^{(k+1)} = \hat{\mu}^{(k)} + \delta_{LM}^{(k)}, \tag{3.8}$$

where $\hat{\mu}^{(k)}$ is the parameter estimate associated with the k'th iteration. Dropping explicit iteration labels for clarity, the Levenburg-Marquardt step δ_{LM} is computed as a modification of the Gauss-

Newton[1] step δ_{GN}. The Gauss-Newton step solves in the least-square sense

$$\begin{aligned} J\delta_{GN} &= r & (3.9)\\ &= y - h(\mu), & (3.10) \end{aligned}$$

where elements of Jacobian J are

$$\begin{aligned} J_{k,i} &= -\frac{\partial r_k}{\partial \mu_i}\bigg|_{\hat{\mu}} \\ &= \frac{\partial h_k(\mu)}{\partial \mu_i}\bigg|_{\hat{\mu}}, & (3.11) \end{aligned}$$

and $r = y - h(\mu)$ is the residual. The Levenburg-Marquart step δ_{LM} follows the Gauss-Newton step with the addition of a regularization term λD, so that

$$(J'J + \lambda D)\delta_{LM} = J'r, \tag{3.12}$$

where λ and sometimes D are changed on each iteration. Implementations of Levenburg-Marquardt differ primarily in the heuristics used to update D and λ. The Levenburg-Marquardt step δ_{LM} approaches the Gauss-Newton step δ_{GN} as $\lambda \to 0$. As λ increases with $D = I$, δ_{LM} goes to zero and approaches the steepest-descent direction [57, 55]. In [57], several modified Gauss-Newton updates claiming superior performance for large-residual problems are surveyed. Many of these methods address the second order terms ignored in the Gauss-Newton approximation $J'J$ of the Hessian of $V(\mu)$. Others suggest cubic approximation rather than quadratic [16] or other refinements [2]. Seber and Wild [57] suggest that Levenburg-Marquardt methods remain popular because they combine good performance with moderate complexity.

Ultimately, Levenburg-Marquardt and higher-order methods still use *local* information, and hence are susceptible to local minima in $V(\mu)$. For local extrema, the gradient

$$\nabla V(\mu) = 0. \tag{3.13}$$

Substituting the loss function (3.3) into this expression and expanding the ∇ operator yields

$$\begin{aligned} \nabla V(\mu) &= \frac{1}{2}\sum_{i=1}^{n} \left(\frac{\partial}{\partial \mu_1} \quad \frac{\partial}{\partial \mu_2} \quad \cdots \quad \frac{\partial}{\partial \mu_m} \right) r_i^2 \\ &= \sum_{i=1}^{n} \left(\frac{\partial r_i}{\partial \mu_1} \quad \frac{\partial r_i}{\partial \mu_2} \quad \cdots \quad \frac{\partial r_i}{\partial \mu_m} \right) r_i, & (3.14) \end{aligned}$$

[1] For further details on Gauss-Newton and Levenburg-Marquardt, see Appendix B.

where n is the number of observations (elements in the residual) and m is the number of parameters. With J defined as in (3.11), the summation

$$\sum_{i=1}^{n} \left(\frac{\partial r_i}{\partial \mu_1} \quad \frac{\partial r_i}{\partial \mu_2} \quad \cdots \quad \frac{\partial r_i}{\partial \mu_m} \right) r_i = -(J'r)'. \tag{3.15}$$

So, for local minima of $V(\mu)$, the term $J'r$ driving the Levenburg-Marquardt step (3.12) goes to zero and iteration fails to improve the estimate.

The problem of minimizing the least-squares criterion is a well-researched area, with many approaches. Section 3.3 does not propose a new solution to the general nonlinear least-squares problem, but rather a new way of applying existing techniques to signals involved in estimating parameters of dynamic systems. An alternate view is that the problem does not lie with the nonlinear least squares routines, but with the initial guess. In §3.4 knowledge of the system to be identified is used to construct a "pre-estimator" that computes a good initial guess for minimization given experimental data.

3.2 Diagnostics

This thesis proposes the use of models, particularly physical models, for nonintrusive diagnostics. Prior work in diagnostics includes physical model-based and other approaches, many of which are complementary in the sense that they could be easily adapted to the framework and methods of this chapter. Although not in a nonintrusive context, [11, 12] apply parameter identification of a physically based model to the problem of detecting broken rotor bars. Parameter estimation in induction machines, with possible application to diagnostics, appears in [60, 58, 51, 65, 67, 66]. Approaches other than identification of physical, model-based parameters exist and could potentially be applied in the nonintrusive context. For example, [56] demonstrates that powerful conclusions about certain low-order nonlinear dynamic systems can be extracted by appropriate pattern matching techniques. Although not directly proposed as a technique for failure detection in [56], it seems that pattern-matching may yield useful diagnostic information. Numerous "neural-network" techniques have also been proposed [13, 14, 24, 25]. Unfortunately, neural-network and other black-box approaches often suffer from less than thorough validation – it is sometimes not clear that the results are not spurious or achievable by more straightforward means. Finally, there are diagnostic approaches that take a purely second-order, statistical approach. These techniques model the test data, or some frequency or wavelet transform of the test data, as samples of a particular distribution. The assumption is made that changes in the system change the distribution of the test data, and diagnostic questions are answered by comparing test data to several reference distributions using a test metric. Often, the

issue of whether a statistical characterization of the system is appropriate is ignored. Also, results are frequently presented in terms of the number of "right" and "wrong" decisions with the authors' thresholds and test data – this approach hides the behavior of the underlying statistical measures and does not give a good indication of performance. Examples in motor diagnostics include [76, 75] and, in a survey style, [35]. These methods are not considered in the thesis, without implication that they are of no use when properly applied.

Perhaps one reason that approaches other than physical-model identification are so popular is that the system identification tasks, modeling and parameter estimation, are often daunting. In many cases a model can be written but fitting parameters requires complicated methods and, too often, excellent initial guesses. In contrast, black-box techniques usually employ tractable, general purpose model structures. A goal of the thesis is to make physical-model identification and nonintrusive diagnostics simple and practical.

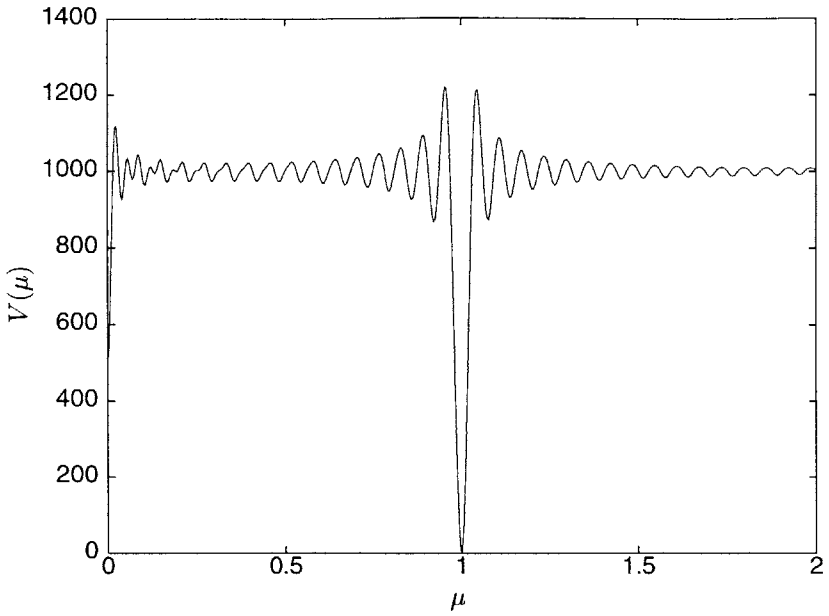


Figure 3-3: Loss function $V(\mu)$ defined in (3.17).

3.3 A method for nonlinear problems

Nonintrusive diagnostics requires parameter estimation methods that can avoid local minima and efficiently handle poor initial guesses without intervention. The iterative technique proposed in this section can be illustrated by considering identification of the model

$$y = \sin(\mu t), \tag{3.16}$$

with observations $\sin(t)$. The problem is to find an estimate $\hat{\mu} = 1$ given observations, the model above, and an initial guess. For N observations sampled with period T, the loss function $V(\mu)$ is

$$V(\mu) = \sum_{i=1}^{N} (\sin(iT) - \sin(\mu iT))^2. \tag{3.17}$$

In Fig. 3-3, $V(\mu)$ is plotted for $\mu \in [0, 2]$ with $T = 0.1$, $N = 1024$ and $\hat{\mu} = 1$. The numerous local minima in $V(\mu)$ suggest that the initial guess must be very good for conventional methods to converge on the desired estimate. The difficulty of finding $\hat{\mu}$ in this case is unintuitive, as $y(t)$ is the response of a simple, linear differential equation.

Another aspect of the difficulty in this particular example is the output error (OE) formulation

of the problem. The output error formulation minimizes the vector of errors with elements

$$e_k = y_k - \hat{y}_k|\mu, \tag{3.18}$$

where $\hat{y}|\mu$ is interpreted as the estimate of y given μ. In this formulation, the effect of noise (under the usual assumptions) is easy to handle, and the parameters can be used in a simulation to reproduce the observations. Alternatively, one could consider the vector of prediction errors with elements,

$$e_k = y_k - \hat{y}_k|y_{k-1}, \mu. \tag{3.19}$$

The prediction error formulation minimizes the error in predictions of the next output value y_k given knowledge of prior outputs and the parameters. An advantage of the prediction error approach is that it penalizes a *localized* error, i.e. the ability to predict between adjacent points rather than the ability to predict the entire output, and sometimes results in a simpler identification problem. The difficulty with the prediction error formulation is that $\hat{y}_k$ may depend on states that are not directly observed. These states may be difficult or impossible to infer from y_{k-1}.

Artificially localizing the error in loss functions like (3.17) might simplify minimization. One way to do this is to restrict the length N of the interval over which (3.17) is evaluated. In fact, temporarily discarding data in this fashion tends to produce a loss function with friendlier characteristics. The effect is shown graphically in Fig. 3-4, where the normalized loss function

$$\frac{V(\mu, N)}{N} = \frac{1}{N}\sum_{i=1}^{N}(\sin(iT) - \sin(\mu iT))^2 \tag{3.20}$$

is plotted over the μ, N plane. For small N the valley leading to the global minimum at $\hat{\mu} = 1$ is wider, implying that convergence will succeed for a wider range of initial guesses. One interpretation is that the interval N must be sufficiently small so that the difference $\sin(\mu t) - \sin(\hat{\mu} t)$ for any particular $t < NT$ is consistently related to the error in the parameters $\mu - \hat{\mu}$. This interpretation is emphasized in Fig. 3-5, which shows global and low-time responses of the model for different values of μ. To put these observations to work, coarse estimates should be refined using small N; as the estimate improves, N should be expanded to include the entire data set.

In a more general context, minimizing the loss function in a series of steps in N has other benefits. For example, if a system has a separation of time constants, minimization using an increasing number of data points may effectively resolve parameters associated with the different time scales separately. For some systems, this may involve solving a series of small dimension problems rather than one higher dimension problem that would result from identification using the entire data set at once. Also, if the model requires simulation of a dynamical system, starting with a small number of points

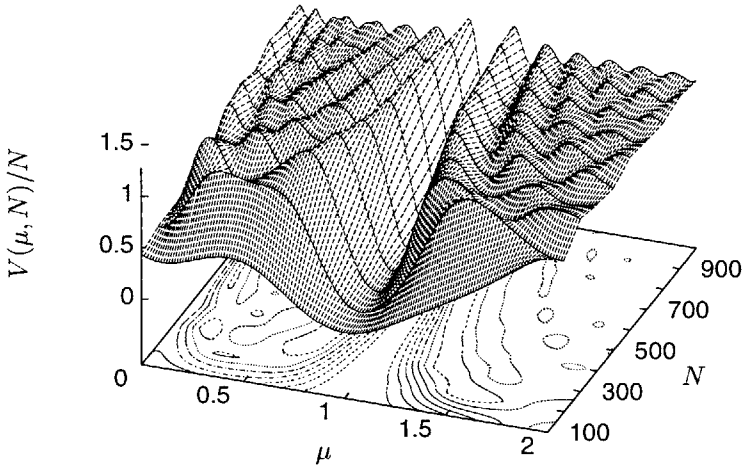
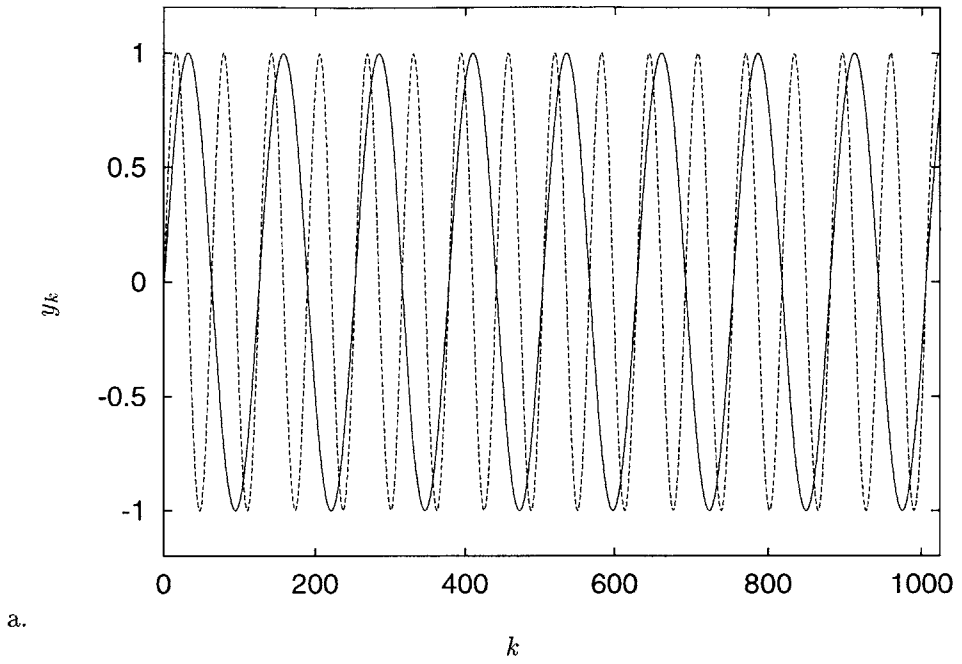


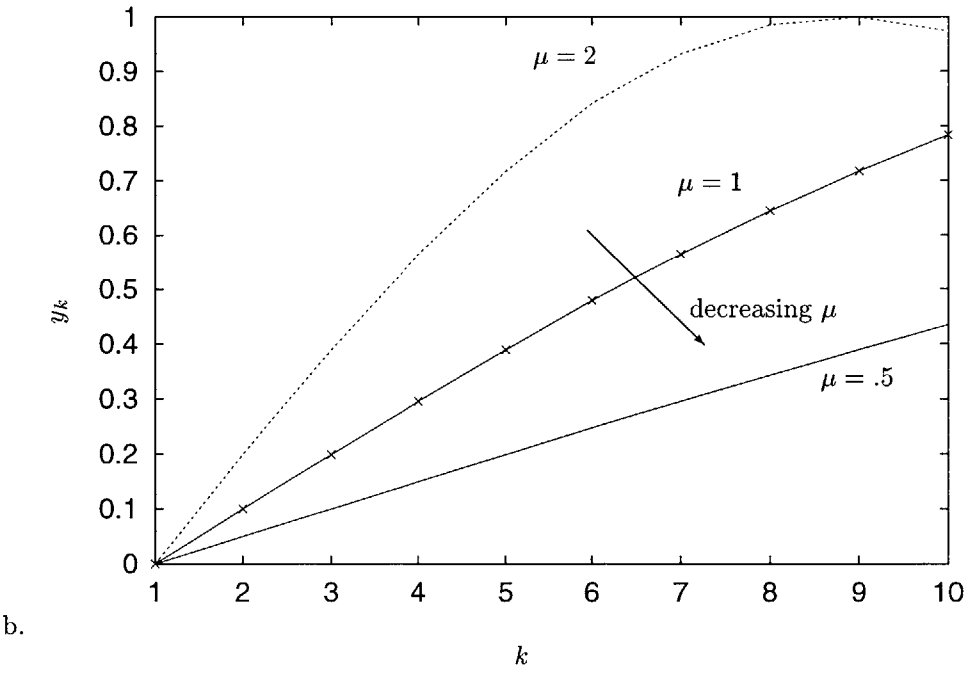
Figure 3-4: Loss function $V(\mu, N)/N$. For small N, a wider range of initial guesses in μ lead to the desired minimum $\hat{\mu} = 1$ than for large N.

may greatly reduce the simulation time required.

Subroutine 3.1 incorporates the observations from Fig. 3-4 in an Octave function minimizing a user-supplied function f that returns the residuals as a function of the parameters. Notice that the method involves repeated application of a conventional nonlinear least squares routine, `leastsq`, which implements a variation of the Levenburg-Marquardt method. Subroutine 3.1 also includes a back-tracking feature. If the criterion $V(\hat{\mu}, N)/N$ exceeds a threshold after minimization with N points, it is assumed that a local minimum was encountered because N was increased excessively on the last iteration. In this case, N is scaled back and μ is reset to its value on the previous step before continuing. The threshold in Subroutine 3.1 could be selected given knowledge of the expected value of $V(\hat{\mu}, N)/N$ in a particular problem.



a.



b.

Figure 3-5: Global (a) and low-time (b) model responses for different values of μ. Responses for $\mu = 1$, $\mu = .5$ are shown in plot (a). The relationship between μ and the model response is comparatively simpler for the low-time data shown in (b).

Subroutine 3.1 *Solution of nonlinear least squares problem by successive application of* `leastsq` *on small intervals of data.*

```
% -*-Octave-*-                                                          0
% [mu,n,thres,rval] = oldmethod('f', mu, nstart, ndat, thres)
% minimize r(1:N) = f(\mu) starting with initial guess mu
% nstart is number of points to start with
% ndat is the number of data points
% thres is the threshold for local minima
function [mu,n,thres,rval] = oldmethod(f, mu, nstart, ndat, thres)
  global N;

  rval = 0;
  if(nstart < 50)    % require at least fifty points                    10
    return;
  end;

  m1 = 0;
  m2 = nstart;

  while(1)
    N = m1+m2;
    if(m2 <= 0)
      disp('no progress');                                              20
      return;
    end;

    mu = [mu leastsq(f, mu(:,columns(mu)))];  % conventional NLLS

    % evaluate residual by calling user function
    e = eval(sprintf('%s(mu(:,columns(mu)));',f),'error');

    crit = norm(e) / sqrt(N);  % compute sqrt of loss function
                                                                        30
    if(crit > thres)       % adjust the number of points examined
      m2 = floor(m2*.75);
      mu = mu(:, 1:columns(mu)-1);
    else
      if(N == ndat)
         break;
      end;
      m1 = N;
      m2 = floor(N*1.5 - m1);
      if(m1+m2 > ndat)                                                  40
         m2 = ndat - m1;
      end;
    end;
  end;

  rval = 1;
end;
```

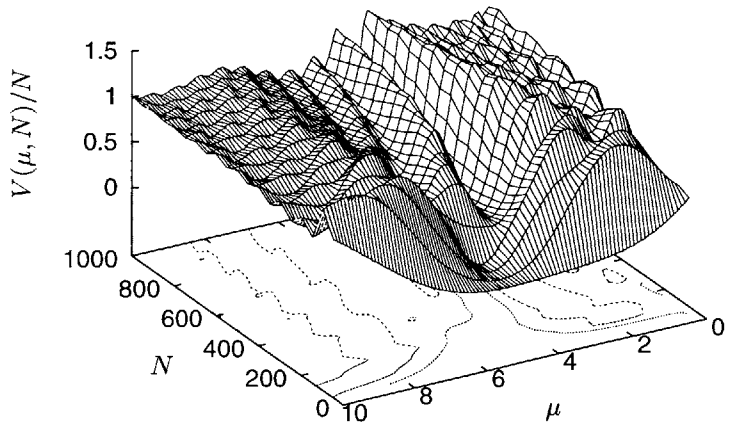


Figure 3-6: Loss function $V(\mu, N)/N$ with low-time model errors. Notice that the desired minimum shifts as N is increased.

Subroutine 3.1 is essentially the same as the "general method" reported in [60], except that a constrained nonlinear least squares routine was used. The results in [60] were quite encouraging and motivated further investigation of the technique. One problem with Subroutine 3.1 is that disturbances in low-time data, especially unmodeled correlated phenomena, sometimes "throw-off" the first couple of estimates with small N. Specifically, an unanticipated feature in the low-time data may result in a wildly incorrect intermediate estimate based on that data, rather than the rough approximation suggested by Fig. 3-4. For example, in the induction motor identification problems considered in [60], unmodeled low-time features in measured data sometimes caused problems.

The low-time error problem is illustrated in Fig. 3-6, where the loss function

$$\frac{V(\mu, N)}{N} = \frac{1}{N} \sum_{i=1}^{N} (y(iT) - \sin(\mu i T))^2 \tag{3.21}$$

with $T = 0.01$ is plotted for observations

$$y(t) = \begin{cases} \sin(5t) & \text{for } t < 2 \\ \sin(4t) & \text{otherwise} \end{cases} \tag{3.22}$$

that are contrived to have low-time errors. For illustrative purposes, the range of μ in Fig. 3-6 is greater than in Fig. 3-4. For low-times, the minimum has a parameter value of about $\mu = 5$. As the amount of data N increases, the minimum value wanders to about $\mu = 4$. Although the

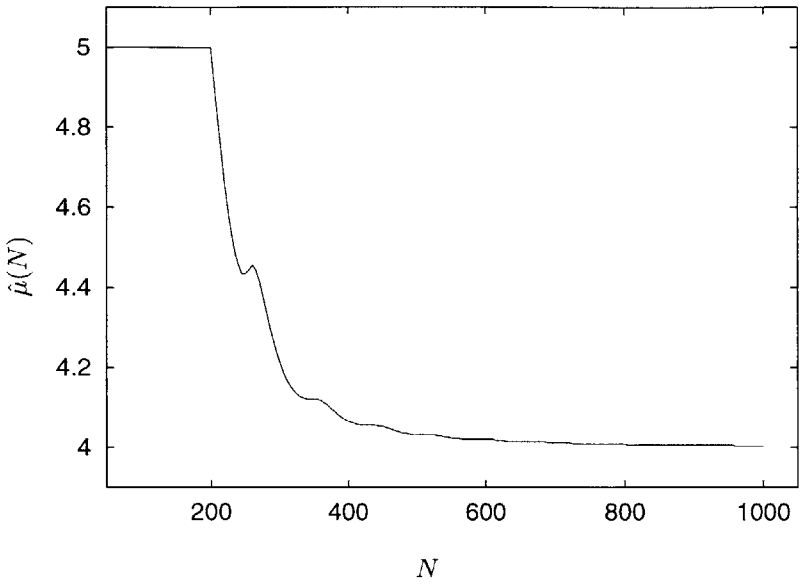


Figure 3-7: Value of the global minimum of each loss function $V(\mu, N)/N$ in Fig. 3-6 as a function of N.

model is in some sense "wrong," $\mu = 4$ provides a better overall fit of the data than any other value. Subroutine 3.1, operating on the loss function in Fig. 3-6, might well "leap" from its low-time estimate directly into the folds of the trough to the left of the trough leading to the minimum at $N = 1000$. A generic loss function might have curves like Fig. 3-6 anywhere in N. To negotiate these curves, a method starting with low-time data would need to take steps in N less than a certain critical length, or risk skipping from a route leading to the minimum into a route leading to a local minimum. Of course, steps that are too small increase overhead. Subroutine 3.2 advances in N until the error criterion $V(\hat{\mu}, N)/N$ exceeds a threshold, possibly signaling that the method has entered a local-minimum trough. If this happens, the routine searches for a shorter scale in N to attempt to avoid the trough. If no such scale can be found, the routine increases the threshold for discriminating against local minima and continues. Given knowledge of the disturbance in a particular problem, the threshold might be picked to reflect the expected value of $V(\hat{\mu}, N)/N$.

Subroutine 3.2 *A method for solving nonlinear least squares problems with improved model-error and low time disturbance rejection.*

```
% -*-Octave-*-                                                          0
% [mu,n,thres,rval] = method('f', mu, nstart, ninc, ndat, thres)
% minimize r(1:N) = f(\mu) starting with initial guess mu
% nstart is number of points to start with, ninc is nominal increment
% ndat is the number of data points, thres is threshold for local minima
% return values: n is schedule and thres is threshold actually used
function [mu,n,thres,rval] = method(f, mu, nstart, ninc, ndat, thres)
  global N;

  rval = 0;
  if(nstart < 10)                                                       10
    return;
  end;

  n = [nstart:ninc:ndat]';
  if(n(columns(n)) != ndat)
    n = [n; ndat];
  end;

  crit = zeros(size(n));
                                                                        20
  for i = 1:rows(n)
    N=n(i);

    mu = [mu leastsq(f, mu(:,columns(mu)))];  % conventional NLLS

    % evaluate user-supplied function f(mu) to get residuals
    e = eval(sprintf('%s(mu(:,columns(mu)));',f),'error');

    crit(i) = norm(e) / sqrt(n(i));   % sqrt of loss function
                                                                        30
    if(crit(i) > thres)
      [a, b, c, rval] = method(f, mu(:,1), floor(n(1)*.75),
                               floor(ninc*.75), ndat, thres);

      if(rval == 1)       % success with finer scale?
        mu = a;
        n = b;
        thres = c;
        return;
      end;                                                              40

      thres = thres*1.2;  % no success, bump up the threshold
    end;
  end;
  rval = 1;
end;
```

3.3.1 Relation to Kalman filters

The idea of refining an initial guess with low-time data is reminiscent of the Kalman filter and its specialization to recursive least-squares. For linear models, recursive solutions of Kalman filter equations are equivalent to batch solutions [62], although in practical situations recursive solutions may have numerical difficulties. In the nonlinear case, recursive solution of the extended Kalman filter is not equivalent to batch estimation. Practical experience may be that the extended Kalman filter offers some of the benefits claimed for the approaches proposed in this chapter. The following is a simplistic discussion of the Kalman and extended Kalman filters to explore the relationship between these updates and the proposed methods.

The Kalman (or Kalman-Bucy) filter is usually applied to estimation of state variables for control purposes [62, 50, 30]. Given an initial guess of the state x of the system, the Kalman filter updates an estimate of the state using observations that are a function of the state. For a linear system

$$x_{k+1} = A_k x_k \tag{3.23}$$

with linear output equation

$$y_k = C_k x_k, \tag{3.24}$$

the Kalman filter can be viewed as the system

$$\begin{pmatrix} C_0 & 0 & 0 & \cdots & & 0 \\ -A_0 & I & 0 & \cdots & & 0 \\ 0 & C_1 & 0 & \cdots & & 0 \\ 0 & -A_1 & I & \cdots & & 0 \\ \vdots & & & \ddots & & \vdots \\ & & & & C_n & 0 \\ 0 & & & \cdots & -A_n & I \end{pmatrix} \begin{pmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \vdots \\ \hat{x}_n \\ \hat{x}_{n+1} \end{pmatrix} = \begin{pmatrix} y_0 \\ 0 \\ y_1 \\ \vdots \\ y_n \\ 0 \end{pmatrix}, \tag{3.25}$$

where the $\hat{x}_k$ are successive estimates of the state and y_k are observations of the system. The structure of this matrix can be exploited to produce estimates recursively as new data arrive. One recursive solution takes the form

$$\hat{x}_{k+1} = A_k \hat{x}_k + K_k(y_k - \hat{y}_k), \tag{3.26}$$

where K_k is the "Kalman gain", y_k is the observation vector, and $\hat{y}_k = C_k \hat{x}_k$ is the estimate of the observation at time k given the estimate $\hat{x}_k$ of the state at time k. In control parlance, the prediction

error $y_k - \hat{y}_k$ is the "innovation." The errors in the state estimate are

$$\delta_{k+1} = x_{k+1} - \hat{x}_{k+1}, \tag{3.27}$$

where x_{k+1} is the true (and unknown) value of the state at time $k+1$. The evolution of these errors is

$$\delta_{k+1} = (A_k - K_k C_k)\delta_k, \tag{3.28}$$

and the Kalman gain K_k is picked to minimize δ_{k+1} by completing the square of (3.28).

In recursive linear least squares, the goal is to estimate constant but unknown parameters x of a system using incoming measurements. In the Kalman framework, this corresponds to a system where

$$x_{k+1} = x_k, \tag{3.29}$$

with linear output equation

$$y_k = C_k x_k. \tag{3.30}$$

Putting these constraints in (3.25) yields

$$\begin{pmatrix} C_0 & 0 & 0 & \cdots & & 0 \\ -I & I & 0 & \cdots & & 0 \\ 0 & C_1 & 0 & \cdots & & 0 \\ 0 & -I & I & \cdots & & 0 \\ \vdots & & & \ddots & & \vdots \\ & & & & C_n & 0 \\ 0 & & & \cdots & -I & I \end{pmatrix} \begin{pmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \vdots \\ \hat{x}_n \\ \hat{x}_{n+1} \end{pmatrix} = \begin{pmatrix} y_0 \\ 0 \\ y_1 \\ \vdots \\ y_n \\ 0 \end{pmatrix}, \tag{3.31}$$

which can be manipulated to form a recursion for $\hat{x}$. The estimates $\hat{x}_k$ are constrained so that if (3.31) is solved all at once (not recursively), then $\hat{x}_0 = \hat{x}_1 = \cdots = \hat{x}_n$. If (3.31) is rewritten in terms of $\hat{x}_n$, the familiar least-squares arrangement

$$\begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_n \end{pmatrix} \hat{x}_n = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{3.32}$$

is obtained. The ordinary and recursive least-squares estimates at time n are therefore equivalent, since they both solve (3.31).

The Kalman framework is interesting in the context of this chapter because it involves a series

of estimates based on increasing amounts of data. Like the methods of §3.3, the Kalman framework can be applied to nonlinear systems. The generalization of the Kalman filter to nonlinear systems is called the extended Kalman filter, and involves linearizations of the output and/or evolution equations at appropriate operating points. However, if the linearized analog of (3.25) is solved recursively, early linearizations are not evaluated at the correct operating points as determined by the most recent estimates. In contrast, the methods of §3.3 use linearizations based on the latest parameter estimates for all observations. With the extended Kalman filter, a qualitatively similar effect could be achieved by retaining the entire record of observations, $y_0 \cdots y_n$, and restarting the filter.

3.3.2 Example : chirp convergence test

The advantage of Subroutine 3.1 can be demonstrated by a Monte-Carlo style convergence test. The loss function for convergence of the model

$$y(t) = \sin(\mu_0 t + \mu_1 t^2) \tag{3.33}$$

using 1000 samples in $t \in [0, 4]$ to nominal parameters $\mu_0 = 10$, $\mu_1 = 5$ is depicted graphically in Figure 3-8a. Levenburg-Marquardt and Subroutine 3.1 were compared by testing convergence from 250,000 randomly selected initial guesses over the domain indicated in Figure 3-8a. The Levenburg-Marquardt method, without modification, converged successfully from the group of initial guesses superimposed on the loss function in Figure 3-8b. The method in Subroutine 3.1 was successful in all cases.

3.3.3 Example : Low-time model error avoidance

It is not difficult to find a low-time model-error situation where Subroutine 3.1 fails. A few examples based on the model

$$y(t) = \sin(\mu_0 t) + \mu_1 \tag{3.34}$$

using the initial guess $\mu_{0,1} = 1$ are shown in Fig. 3-9. The low-time error in Fig. 3-9a is an offset present in the "measurement" for $t < 1$. In Fig. 3-9b, the low-time error is in the frequency of the signal, which changes abruptly at $t = 2$. These are low-time model-errors in the sense that the model, when fit to the bulk of the data, does not describe the data at low times. The successful fits in Fig. 3-9 are the result of Subroutine 3.2, which worked in all cases. The highly correlated disturbances in Fig. 3-9 clearly do not fit the usual stochastic assumptions, and model errors this gross should probably be handled differently. On the other hand, there are situations where the

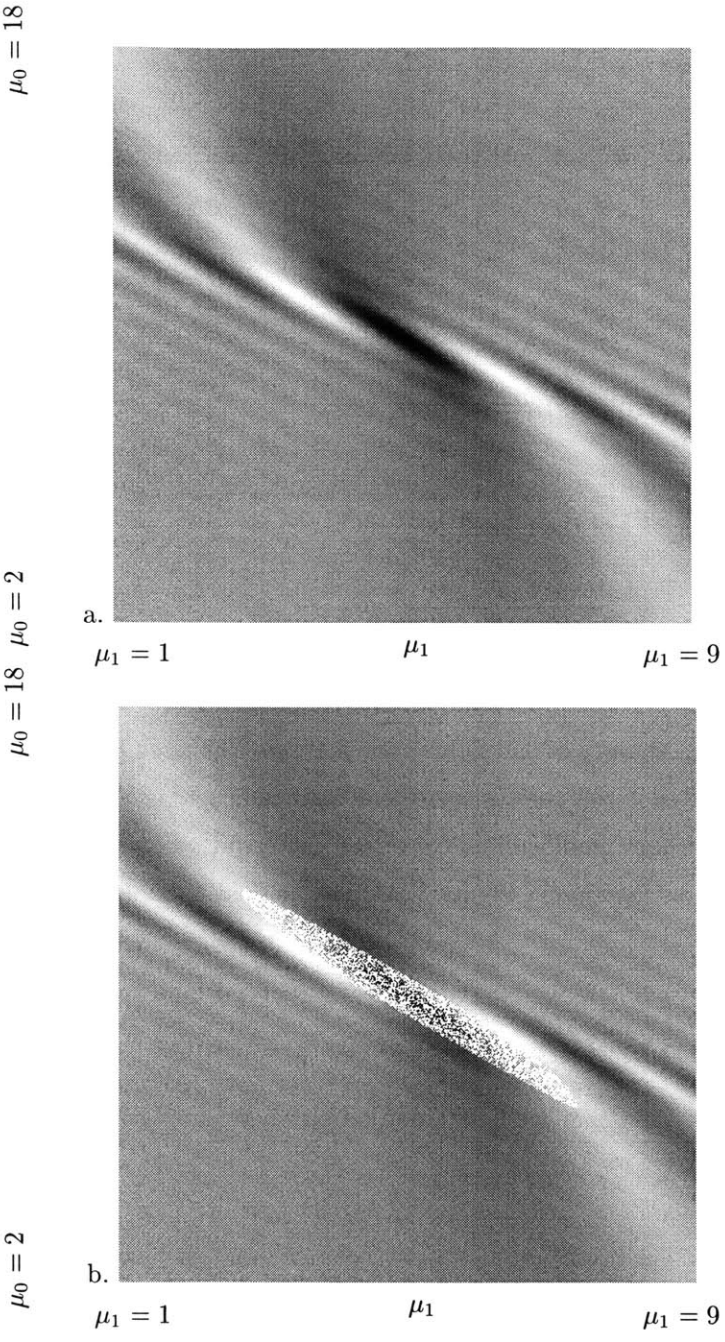


Figure 3-8: a. Loss function for a chirp signal. Darker areas are lower values of the loss function. b. Group of 8466 Levenburg-Marquardt convergent initial guesses superposed on loss function. 250,000 uniformly distributed, randomly selected initial guesses were attempted over the domain.

modeling errors are relatively small and also difficult to correct. In these cases, ignoring the model error at the expense of a parameter bias may be preferred.

It is interesting to note that direct application of the Matlab routine `leastsq` to all of these situations, including a plain sinewave with no low-time model errors, failed. However, the conventional Matlab routine required 105,000 evaluations of the scalar function sin before failing, where Subroutine 3.2 used roughly 119,000 evaluations for a plain sinewave, 248,000 for Fig. 3-9a, and 225,000 for Fig. 3-9b. Since Subroutine 3.2 calls `leastsq`, these numbers are comparable.

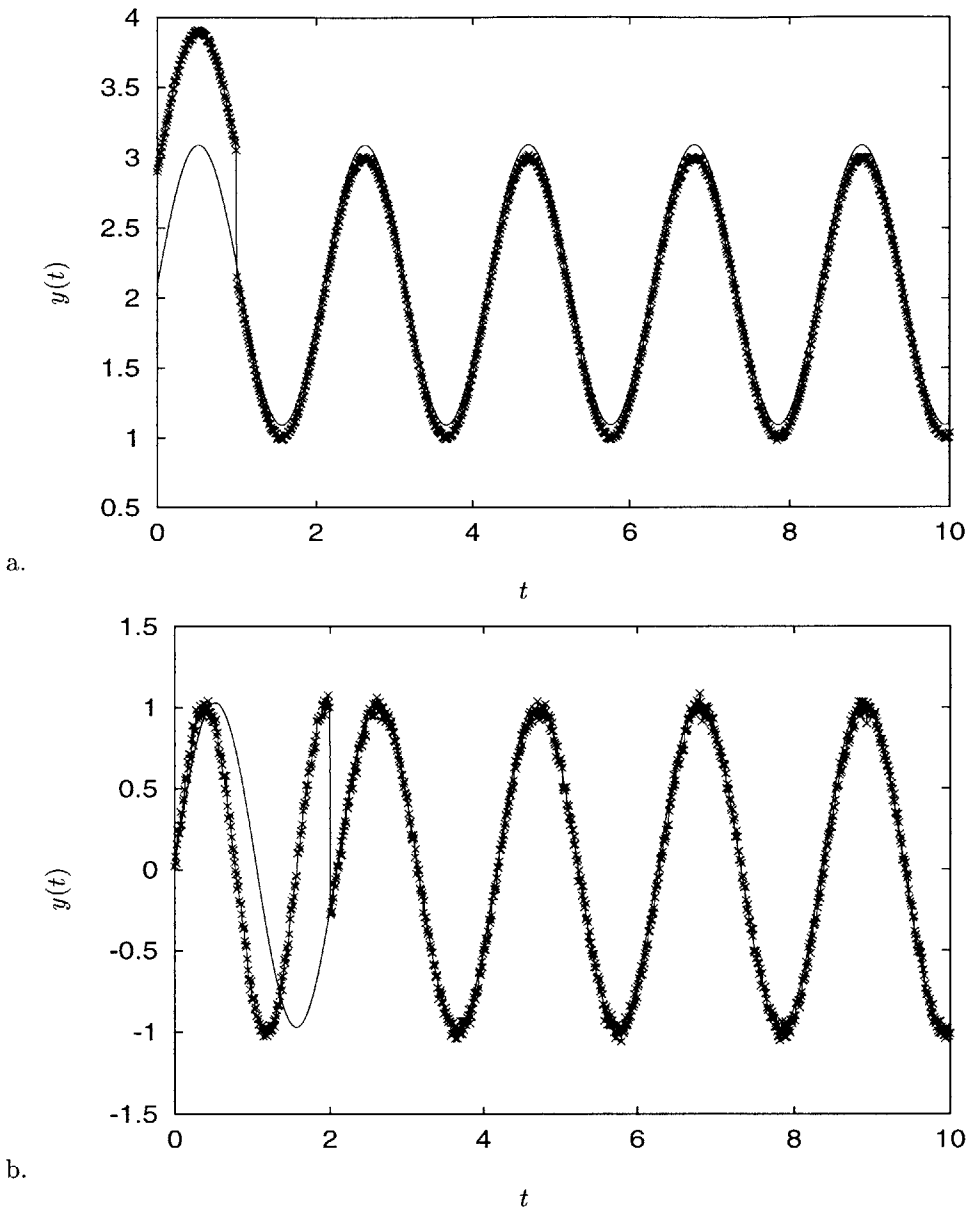


Figure 3-9: Simulated sinewave "observations" with low-time errors fit with (3.34) using Subroutine 3.2. In both graphs, the observations are data points connected by lines and the fit is the solid line. These examples illustrate expected behavior when fitting a model that does not describe the data accurately. Subroutine 3.2 avoids low-time errors, while Subroutine 3.1 does not.

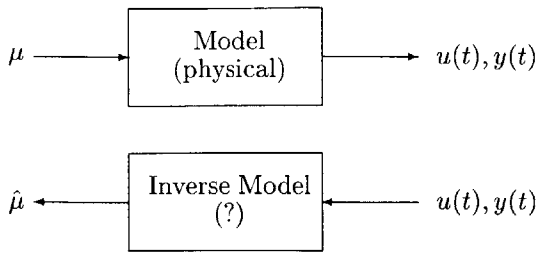


Figure 3-10: Pre-estimation of μ viewed as a modeling problem.

3.4 Fast pre-estimation

One view of iterative minimization of a loss function is that the difficulty is the lack of a suitable initial guess for minimization rather than the minimization procedures themselves. Since most methods perform best when close to a minimum, a fixed-time, direct procedure for formulating a good initial guess is highly desirable. Finding initial parameters without iteration is advanced by Caudill in [31] in terms of "directly inverting" a physical system to estimate its parameters. Caudill asserts that "the existence of the inverse for the available values of u is tantamount to assuming that the data is complete." Stated differently, the presumed identifiability of parameters μ from some system S with inputs u and outputs y is equivalent to asserting the existence of some function

$$\mu = F_S(u, y) \tag{3.35}$$

which conveniently provides the desired parameters. The situation is shown in Fig. 3-10. It is clear that F_S can be investigated off-line, given knowledge of realistic inputs, the model, and reasonable ranges for the parameters. An approximation to F_S might be constructed, at least for some range of physical systems, and employed on-line for fast estimation of initial guess parameters μ_0. The initial guess would ordinarily be further refined using ordinary methods. To emphasize the preliminary nature of an estimate obtained with this approach, the use of F_S will be called "pre-estimation."

In [31], a particular system was considered and an F_S found explicitly for that system. In the generic framework of (3.1), the explicit approach is not feasible, and determination of a pre-estimator becomes a generic modeling problem. This outlook is emphasized in Fig. 3-10. The forward model constraining inputs and outputs as a function of parameters is based on physical reasoning, and the inverse model requires nonlinear black-box flexibility.

One way to proceed with the general idea suggested by Fig. 3-10 is to make the inverse model two parts, as suggested in Fig. 3-11. In Fig. 3-11, the input/output information is decomposed into a meta-parameter estimate $\hat{\gamma}$, and then mapped to the desired parameters. Division of the inverse

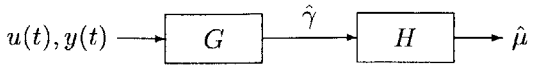


Figure 3-11: Decomposition of inverse model into two steps. The first step G produces meta-parameters $\hat{\gamma}$ which should have good noise properties. The performance of G can be validated independently from the design of H, which effects the final mapping to the desired parameters.

model into two parts is arbitrary and theoretically unnecessary, but splitting the model simplifies the requirements of each half.

Meta-parameters γ should reduce the observational data and be unbiased with respect to the disturbances in the measurements. This statistical requirement means that the model underlying G will probably be derived from familiar system identification models with known properties, e.g. a Box-Jenkins structure, output error model, an auto-regression, or simply a projection on a suitable basis [30]. Independent of the final mapping to parameters, the disturbance characteristics of G and the extent to which G correctly represents the data can be verified. Perhaps most important, specification of the rough form of G allows the engineer to add physically relevant structure without a direct physical model – a process Ljung calls "semi-physical modeling" [46].

The second stage in Fig. 3-11 maps meta-parameters to the parameters of interest. An advantage of the two-step inverse model approach is that H maps a relatively small space (the meta-parameters) to the desired parameters, in contrast to the one-step approach where the entire inverse model is a black box and must map the observational data directly to parameters. However, the two-step approach does not provide any hints as to what structures might achieve the mapping required of H. In principle, if the transformation to meta-parameters is substantially reversible and the parameters are identifiable from the observational data, the mapping is possible – one could just reconstruct the observational data from the meta-parameters and invoke the identifiability argument. This is not a constructive argument, i.e. it does not suggest a procedure for finding H. In practice, a reasonable approach is to try a generic function approximation model and validate its performance. Common choices that generalize cleanly to the multi-dimensional function approximation required of H include, for example, the two-layer perceptron network [71, 61, 5, 48] and radial basis functions [22, 46]. Other structures, e.g. splines, might also be used. Global polynomials, historically a popular approximation tool, are probably a poor choice because their local approximation ability is obtained at the expense of rapidly diverging higher-order terms. These higher-order terms often lead to excessive interpolation errors with arbitrary data. See [22] or the preface to [36] for more details.

The two step approach of Fig. 3-11 is motivated by, yet distinct from, two general strategies in neural networks. One approach is to embed a function thought to be relevant to the purpose of the network, with the goal of reducing the number of unknowns. Approximations of this kind are generically called functional link networks [48]. An interesting example in which the desired functionality is manifest in the structure of the network is given in [69], which combines aspects of the Runge-Kutta integration procedure with a neural network to accurately reproduce the states of a target system for identification. Similarly, in [9] the authors mix auto-regressive techniques with a neural network for the purpose of identifying a simple system. Figure 3-11 splits a complicated task into two specialized subsystems. This approach parallels the popular practice in neural networks of forming a complicated "hybrid network" from multiple sub-networks [48]. In [48], the author cautions that hybrid networks discard information from the input. This is the goal in Fig. 3-11, where G simplifies H by discarding disturbances and even parts of the transient that are less useful in finding the parameters.

A general objection to black-box and neural network techniques is that an unforeseen combination of inputs might produce a wildly incorrect output. This could be disastrous if the black box were modeling a system as part of an active control, for example, as in [72]. As the generality and complexity of any model increases, so does the potential that the model has unanticipated and unfavorable behavior in special circumstances. In the pre-estimation context, however, a small proportion of unanticipated outputs is not a problem. Even if one or two parameter pre-estimates are unreasonable and must be discarded, the remaining pre-estimates may still be tremendously effective in terms of reducing estimation time if they are close to their true values.

Nonlinear black-box modeling is a large and active area, and pre-estimation is a friendly environment for black-box function approximation techniques of all kinds. Combining the general properties outlined in this section with the techniques of any reasonable black-box reference, e.g. [61, 5], would probably yield several possible approaches. Rather than exhaustively investigating these options, this section motivates and demonstrates two particularly compelling choices.

3.4.1 State-space reconstruction and TAR

One possibility for implementing G in Fig. 3-11 is suggested by the idea of state-space reconstruction. State-space reconstruction (the "embedding theorem") holds that the state of a system (at least, the state that matters to the output) can be reconstructed from the lags of the inputs and outputs, up to a homeomorphic distortion [22]. The embedding theorem is especially interesting in its connection to the identifiability of a system. If a system is identifiable given data and a parameterization, by definition the parameters (and initial state if so parameterized) can be found from the input/output data. Given the initial state and parameters, the states can be reconstructed by simulation. In effect,

the difference between plotting the lags and identifying the parameters is finding the distortion that maps the lags to the states!

To illustrate the embedding theorem graphically, consider the Duffing oscillator [70],

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= x - x^3 - \delta y. \end{aligned} \tag{3.36}$$

Figure 3-12a shows a state-space trajectory $x(t)$, $y(t)$ for this system, with t ranging from 0 to 100 and $\delta = 0.2$. Fig. 3-12b shows the lagged pair $x(t)$, $x(t+0.01)$, which is related by a mild distortion, or embedding, to the original states shown in Fig. 3-12a.

In the context of system identification, Fig. 3-12 suggests that lagged data might provide state information that could greatly simplify load parameter estimation problems. This is especially true in situations like nonintrusive diagnostics, where the observations are unlikely to include the entire state of the target load. Unfortunately, the unknown distortion linking the lagged data to the state space is a problem for identification of a physical model. In fast pre-estimation, however, G in Fig. 3-11 uses an *arbitrary* model that should not be perturbed by a mild distortion. If G uses lagged data, the behavior of the system will be captured as if the states were available, but with some distortion in the meta-parameters corresponding to the embedding. Distortion in the meta-parameters is not a concern, as they are mapped by H to the desired pre-estimate. Structures for G that include lagged data seem particularly promising.

One familiar structure incorporating time-lag information is the auto-regressive (AR) model

$$P(z^{-1})y = e, \tag{3.37}$$

where $P(z^{-1})$ is a polynomial in the delay operator and e is a disturbance [46]. The AR model immediately results in a linear least-squares problem with rows,

$$\begin{pmatrix} y_{k-1} & y_{k-2} & \cdots & y_{k-N} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} = y_k + e_k. \tag{3.38}$$

Unfortunately, the disturbance in (3.37) enters in an unlikely way. If y is obtained from measurement, a more appropriate model is

$$P(z^{-1})(y + e) = 0. \tag{3.39}$$

Finding the parameters in this case is a nonlinear problem [30]. If the noise is white, the Yule-Walker

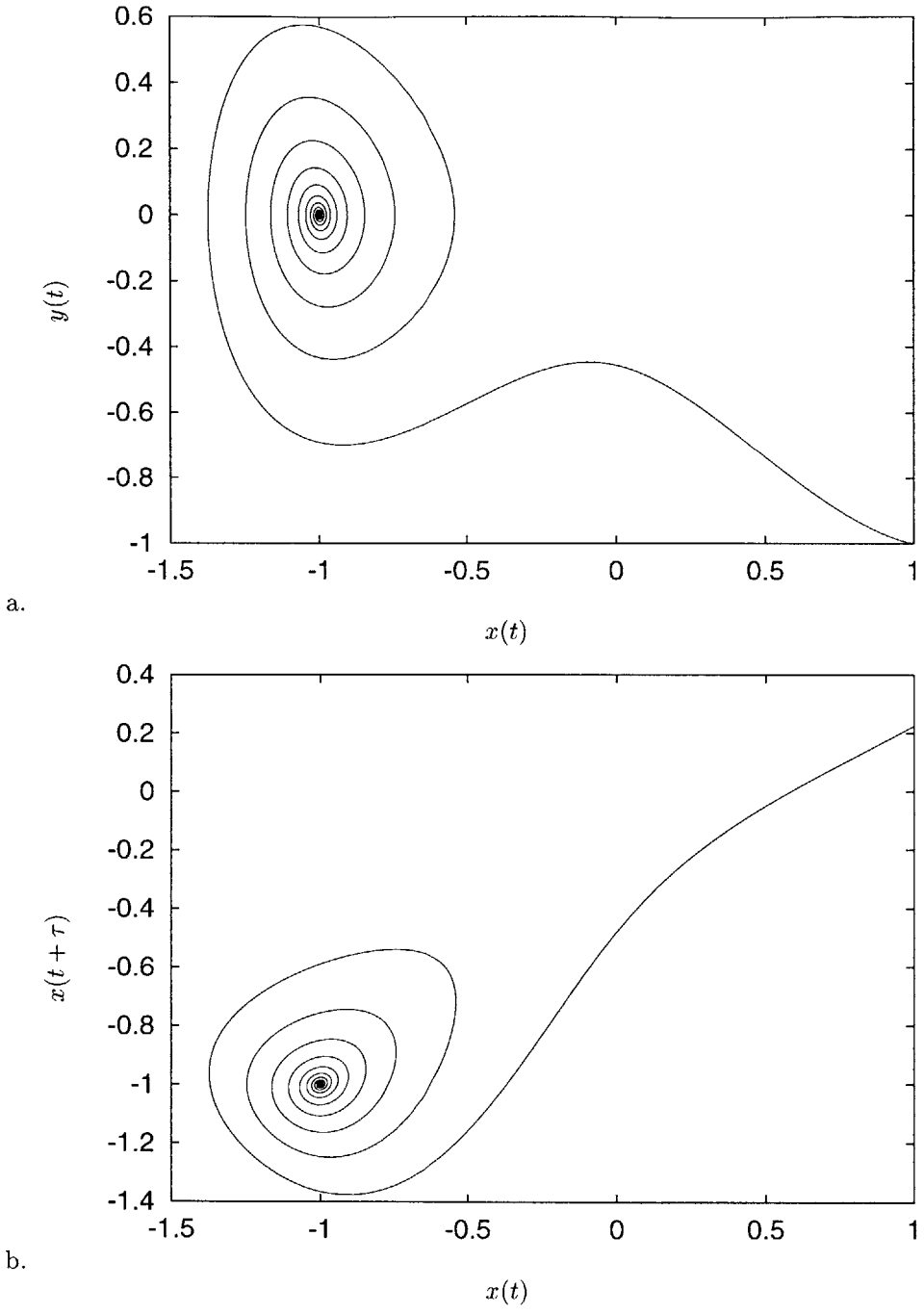


Figure 3-12: a. State space trajectory of a Duffing oscillator. b. Lags of the Duffing oscillator. These spaces are related by a smooth, invertible map.

equations provide an approximate approach. In particular, if (3.38) is written $Ap = b$, the error in $A'A$ due to the presence of noise in the regressors is the sample correlation matrix

$$E'E = \begin{pmatrix} \hat{C}_{yy}(0) & \hat{C}_{yy}(1) & \cdots & \hat{C}_{yy}(n) \\ \hat{C}_{yy}(1) & \hat{C}_{yy}(0) & & \\ \vdots & & \ddots & \vdots \\ \hat{C}_{yy} & & \cdots & \hat{C}_{yy}(0) \end{pmatrix}. \tag{3.40}$$

The Yule-Walker approach is to solve the normal equations with the aid of instrumental variables $Z = z^{-1}A$, i.e. solve

$$Z'Ap = Z'b. \tag{3.41}$$

In this case, the error due to disturbances in the regressors is

$$(z^{-1}E)'E = \begin{pmatrix} \hat{C}_{yy}(1) & \hat{C}_{yy}(2) & \cdots & \hat{C}_{yy}(n) \\ \hat{C}_{yy}(2) & \hat{C}_{yy}(1) & & \\ \vdots & & \ddots & \vdots \\ \hat{C}_{yy}(n) & & \cdots & \hat{C}_{yy}(1) \end{pmatrix}, \tag{3.42}$$

which should be small if the disturbance is uncorrelated. Octave subroutine 3.3 demonstrates how to compute coefficients of an AR model using the Yule-Walker approach.

Subroutine 3.3 *Solution of AR model using Yule-Walker approach.*

```
% -*-Octave-*-                                                    0
% make autoregressive model, Yule-Walker
function [p,e] = arf(data,delays)
  N = rows(data) - delays(length(delays)) - 1;
  A = zeros(N, length(delays));
  Z = A;
  b = -data(1:N);   % undelayed.

  for i = 1:length(delays)
    A(:,i) = data(delays(i)+1:delays(i)+N);
    Z(:,i) = data(delays(i)+2:delays(i)+N+1);                     10
  end;

  A = [A ones(rows(A),1)];
  Z = [Z ones(rows(Z),1)];

  % the identify matrix regularizes
  p = (Z'*A) \ (Z'*b);
  e = A*p-b;
end;
```

If the system behavior is not well approximated by an AR model, several AR models with limited regions of support may provide better characteristics. This is the "threshold auto-regressive" or

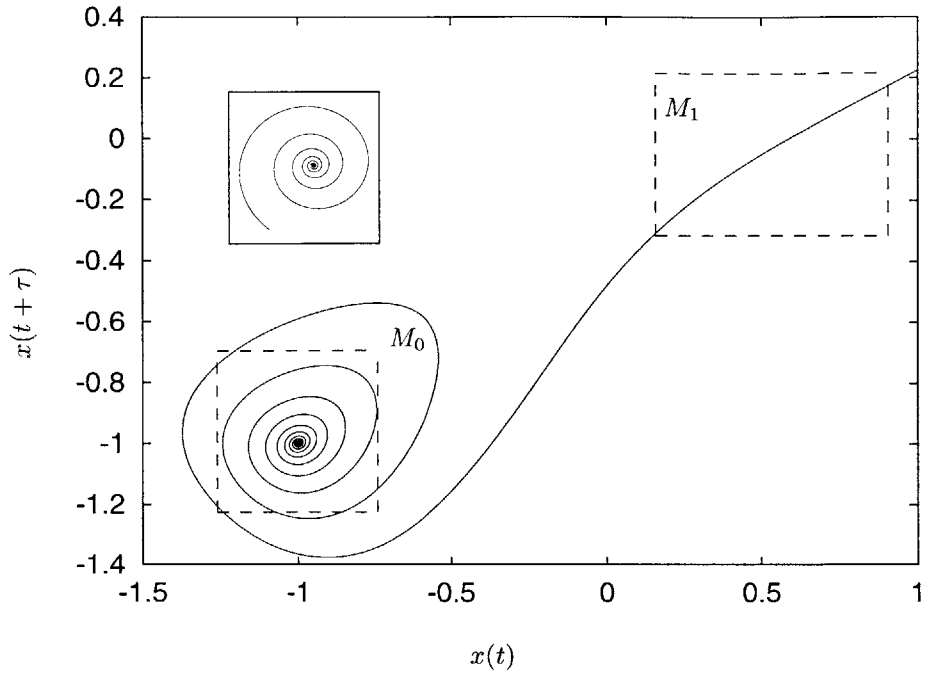


Figure 3-13: Lags of the Duffing oscillator, showing regions where linear models might accurately describe the system. The inset shows the damped sinusoidal response of a linear system for comparison to the region labeled M_0.

TAR model [22]. The idea is shown schematically in Fig. 3-13, where a plot of the lags of a Duffing oscillator response is partitioned into portions of the trajectory that "look linear" and might be approximated by an AR model. The region labeled M_0, for example, is similar to the damped sinusoidal response of a linear system shown in the inset of Fig. 3-13. A series of plots like Fig. 3-13 over a range of parameters for a target system might suggest useful thresholds for a pre-estimation function G based on the TAR model.

The TAR model can be written as a series of AR fits, as in the following Octave example.

Subroutine 3.4 *Solution of threshold autoregressive problem using AR subroutine.*

```
% -*-Octave-*-                                                    0
% threshold auto-regressive model (TAR)
%
function [tp,te] = tarfu(data, thres, delays)
  [tp,te] = arf(data(1:thres(1)),delays);

  for i = 2:length(thres)
    [p,e] = arf(data((thres(i-1)+1):thres(i)), delays);
    tp = [tp; p];
    te = [te; e];
  end;                                                            10
```

```
end;
```

3.4.2 Radial basis functions

Radial basis functions are approximations based on a superposition of functions whose values depend on the distance between the argument of the approximation and a set of control points called centers. In fast pre-estimation, for example, a radial basis function might approximate H to map meta-parameters to pre-estimates. For a user function $F(x)$, $F(R^M) \rightarrow R$, an RBF approximation is

$$y = \sum_{i=1}^{N} a_i \phi(\|x - c_i\|_2) \tag{3.43}$$

for a set of centers c_i [22]. Assuming that the same centers are appropriate for all components of a multidimensional problem, or that the collection of centers is augmented until sufficient, a multidimensional RBF approximation is

$$\begin{aligned} Y &= A \begin{pmatrix} \phi(\|x - c_1\|_2) \\ \phi(\|x - c_2\|_2) \\ \vdots \\ \phi(\|x - c_N\|_2) \end{pmatrix} \\ &\approx \begin{pmatrix} F_1(1) & F_2(x) & \cdots & F_M(x) \end{pmatrix}'. \end{aligned} \tag{3.44}$$

Finding the coefficients A of a radial basis function approximation, given the centers and function $\phi(r)$ can be as simple as solving the problem

$$\begin{pmatrix} \phi(\|x_1 - c_1\|_2) & \phi(\|x_1 - c_2\|_2) & \cdots & \phi(\|x_1 - c_N\|_2) \\ \phi(\|x_2 - c_1\|_2) & \phi(\|x_2 - c_2\|_2) & \cdots & \phi(\|x_2 - c_N\|_2) \\ \vdots & & \ddots & \vdots \\ \phi(\|x_M - c_1\|_2) & \phi(\|x_M - c_2\|_2) & \cdots & \phi(\|x_M - c_N\|_2) \end{pmatrix} A' = \begin{pmatrix} u'_1 \\ u'_2 \\ \vdots \\ u'_M \end{pmatrix}, \tag{3.45}$$

for a set of M input/output pairs $(\, x \quad u \,)$.

Typical choices for the function $\phi(r)$ around which the RBF approximation is built include r, r^3, and e^{-r^2} [22, 5]. Particularly interesting are choices of $\phi(r)$ for which (3.45) is a linear problem; this is the case for r, r^3 but not $\phi(r) = e^{-r^2/\gamma}$, for example. There are several strategies for picking the centers of a radial basis function [22, 5]. These include picking the centers by hand and relying on user intuition, parameterizing the centers to minimize some loss function over the training set (a nonlinear problem), picking arbitrary centers from the training data, picking uniform or random centers on or off the support of the data, or picking centers that are "representative" by clustering the training data.

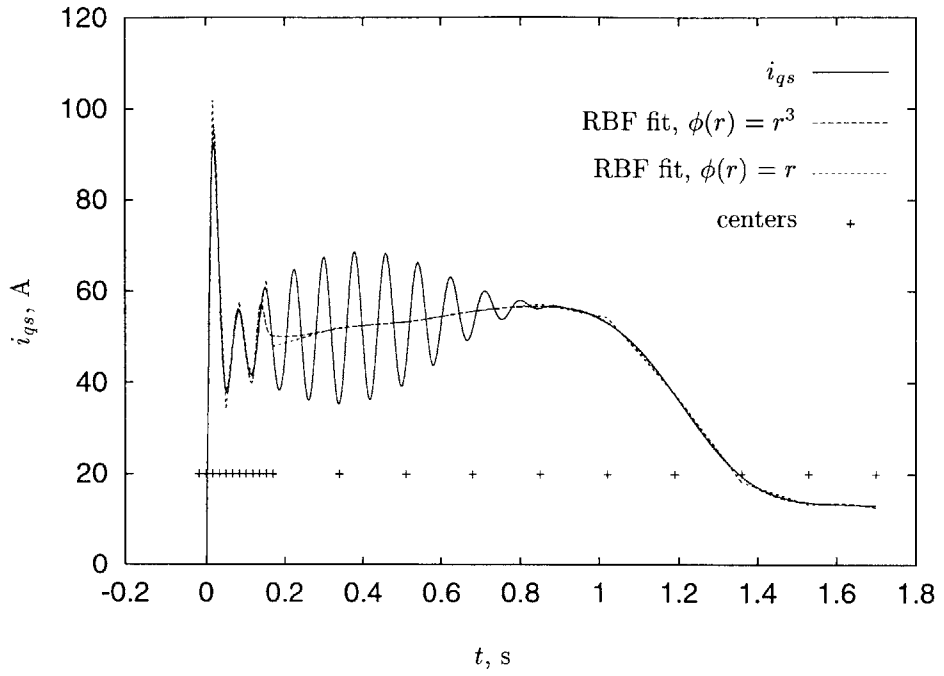


Figure 3-14: Radial basis function approximation of induction motor transient response with linear and cubic $\Phi(r)$. Centers for the radial basis function are indicated.

Figure 3-14 illustrates several aspects of radial basis function approximation by example. In Fig. 3-14, the radial basis function is approximating the scalar-valued time series $i_{qs}(t)$. The argument of the approximation is t, and the centers are the values of t indicated by the points. From Fig. 3-14, it is clear that the approximation with $\Phi(r) = r$ is a piecewise linear fit, with the "joints" determined by the positions of the centers. With $\Phi(r) = r^3$ and the same centers, the approximation is similar but smoother. A final aspect of Fig. 3-14 is that the centers are uniformly spaced in blocks selected for the qualitative goal of "good fit" in the detailed low-time portion of the transient, a "smoothed-over fit" in the middle section, and a close fit for the final part of the transient. The quality of fit and use of the radial basis function approximation in Fig. 3-14 suggests its application as a model for G in Fig. 3-11. The relative ease with which an operator can cause a radial basis function approximation to emphasize "important" parts of a transient is another valuable property in this context. Also, the simple procedure for finding the coefficients and straightforward generalization to higher dimensions make the radial basis function an attractive candidate for H in Fig. 3-11.

3.4.3 AR/RBF pre-estimation of a sinusoidal model

As an example, consider the model

$$y(t) = \mu_1 \sin(\mu_0 t) + \mu_2 \cos(\mu_0 t). \tag{3.46}$$

From the point of view of pre-estimation, the interesting parameter is μ_0, since estimating the other parameters is a linear problem given knowledge of μ_0. Also, the coefficients of an AR model will not depend on μ_1 and μ_2, which together may be thought of as phase and amplitude information. In real problems, it may be that a particular choice for G excludes the possibility of pre-estimating some parameters – this should not cause concern as long as the "hard" parameters can be pre-estimated. This example is one of an interesting set of problems with a linear and a nonlinear part for which decomposed algorithms may be useful [66]. Generally, however, there is little penalty for solving the linear part of the system iteratively. Similarly, there is no point in pre-estimating parameters that enter a problem linearly; essentially perfect values for the linear parameters will be found on the first iteration of the subsequent minimization.

A pre-estimator was designed for the sinusoidal model (3.46) with μ_0 in the range $[6, 60]$, with t uniformly spaced for 1000 samples in the interval $[0, 10]$ and $\| \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \|_2$ in the range $[6, 16]$. A second-order AR model, fit with Subroutine 3.3, was used to obtain meta-parameters. This AR modeling step corresponds to G in Fig. 3-11. The meta-parameters were mapped to $\hat{\mu}_0$ by a RBF with $\phi(r) = r^3$ and four centers selected uniformly on the support of the meta-parameters. The RBF coefficients were obtained from 200 parameter/response pairs obtained for μ selected randomly from a uniform distribution.

The performance of the pre-estimator is shown in Fig. 3-15. The fixed cost pre-estimates of Fig. 3-15 are especially interesting in view of the local minima, e.g. Fig. 3-3, that a direct iterative approach would have to avoid.

Figure 3-16 provides a comparison between the function evaluations required by Subroutine 3.2 with and without a pre-estimated initial guess. A direct comparison to a standard routine like Matlab's `leastsq` would be interesting, but `leastsq` did not successfully converge in all cases. Notice that the performance with the pre-estimator is uniformly as good as the performance without the pre-estimator in the special cases when the initial guess is serendipitously close to the target. Note that a certain fixed number of function evaluations are required to assess termination criteria and evaluate Jacobians. Subroutine 3.2 was invoked with identical initial step and interval parameters for purposes of comparison. If these had been adjusted to reflect the "head start" afforded by the pre-estimated parameters, the difference in function evaluations would be more striking.

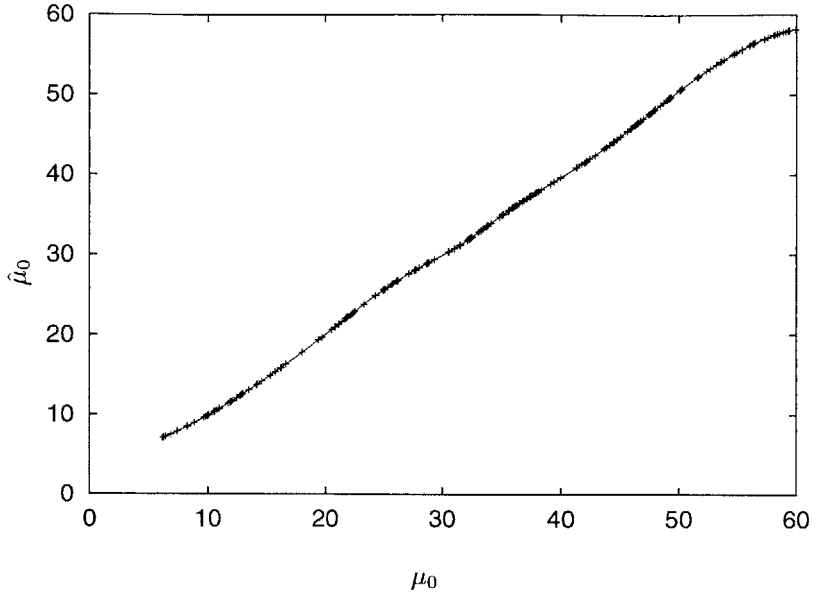


Figure 3-15: Performance of AR/RBF pre-estimator for estimating frequency of a sinusoid. Line indicates performance with parameter set used to obtain the RBF fit, data points show performance with a cross-validation set.

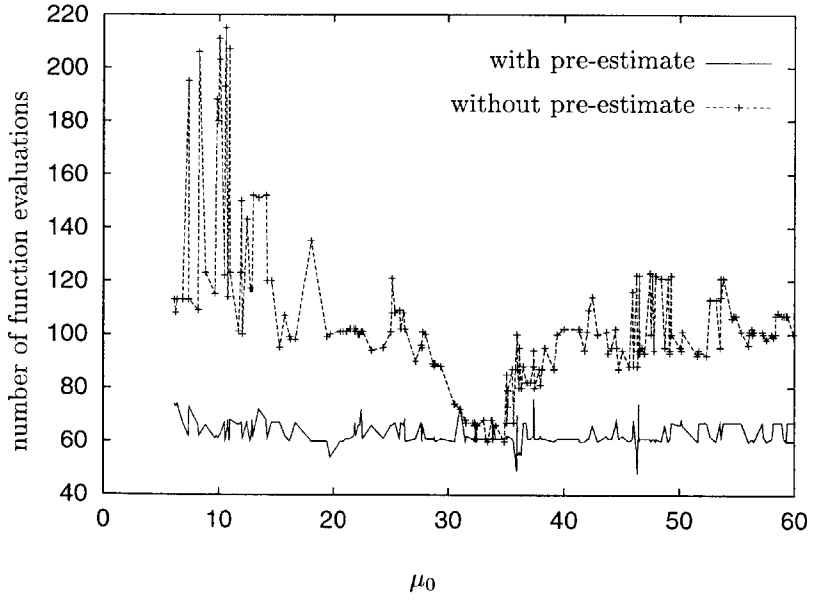


Figure 3-16: Number of function evaluations required for Subroutine 3.2 using a nominal guess centered in the parameter space and using an AR/RBF pre-estimated initial guess, §3.4.3. Parameters were taken from the cross-validation set.

3.4.4 AR/RBF pre-estimation of DC motor with fan load

A DC motor and fan with series resistance R, inertia J, motor constant K, input voltage V and drag coefficient β can be modeled with the differential equation

$$\frac{dx}{dt} = V - \alpha_1 x - \alpha_2 x^2, \tag{3.47}$$

and output equation

$$I = \alpha_3(V - \alpha_1 x), \tag{3.48}$$

where $x = \frac{JR}{K}\omega$ and

$$\alpha_1 = \frac{K^2}{RJ} \tag{3.49}$$

$$\alpha_2 = \frac{\beta K}{J^2 R} \tag{3.50}$$

$$\alpha_3 = \frac{1}{R}. \tag{3.51}$$

This model was developed for diagnostic testing in a vehicle, discussed in §4.3.1. The model is interesting because information about the mechanical configuration can be extracted from the electrical transient.

For pre-estimation, an inverse model was constructed using a first-order AR model with an offset term to obtain meta-parameters (G in Fig. 3-11). The meta-parameters were mapped to a pre-estimate using a radial basis function with eight uniformly spaced centers per dimension in the meta-parameter space (H in Fig. 3-11). The sixteen radial basis function coefficients were found using data from simulations of two hundred random, uniformly distributed parameter vectors ($\alpha_1\alpha_2\alpha_3$). Performance of the AR/RBF pre-estimator with five hundred randomly selected cross-validation parameter sets is show in Fig. 3-17. As in §3.4.3, the linear parameter α_3 was not pre-estimated. Both construction and cross-validation data had Gaussian, white noise added.

3.4.5 RBF/RBF pre-estimation of induction motor parameters

As a final example, consider the induction motor model of §4.3.2. This system has a damping parameter β, an inertia parameter K, and four electrical parameters r_r, r_s, L_l, and L_m.

A pre-estimation system was constructed using operations G and H as suggested in Fig. 3-11, both based on RBF approximations. A time-series RBF approximation with $\phi(r) = r^3$ was used for G, using the same centers as in Fig. 3-14, and applied to both i_{qs} and i_{ds}. Two centers per meta-parameter, selected to cover the range of meta-parameters generated by G for 1500 randomly selected parameter vectors, were used in the RBF approximation to H. The coefficients of H were

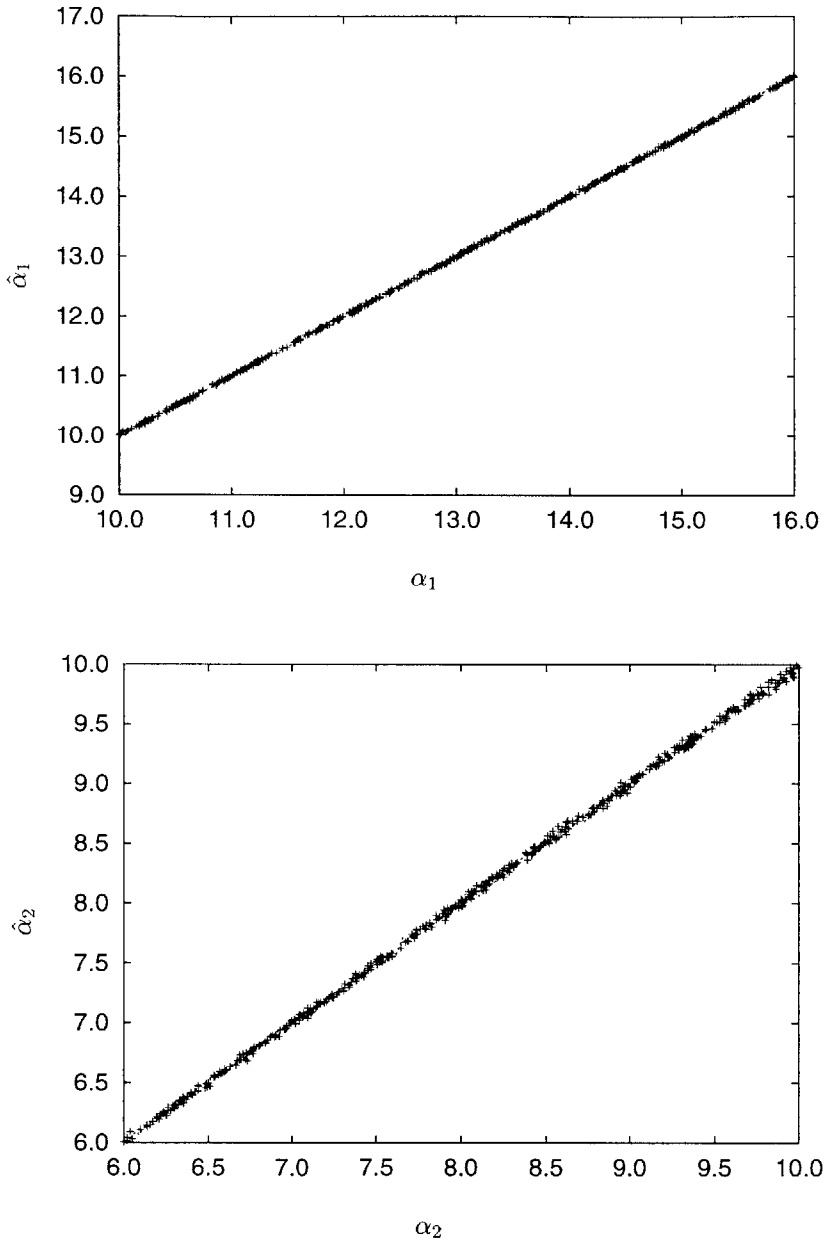


Figure 3-17: Performance of AR/RBF pre-estimator for DC-motor and fan model. Line indicates performance with parameter set used to obtain RBF fit (200 points), data points show performance with a cross-validation set (500 points).

obtained from 4000 randomly selected training parameter vectors and validated using another set of 1500 parameter vectors. The results are shown in Fig. 3-18.

3.4.6 Dataset selection and reduced order modeling

An important empirical observation made in [40] is that electrical transients for similar loads tend to scale in time and amplitude. For example, realistically loaded induction machines make qualitatively similar transients over a broad range of power ratings, although the parameters of the individual machines may be widely different. Larger machines, for example, often have longer, larger transients that are otherwise similar to the transients from small machines. This observation is of great importance to the NILM, and suggests that many transients may be described by just two parameters, gain and time scale, with respect to a reference transient. In a gross sense, a typical five to seven parameter physics-based model may be over-parameterized for an initial search. One way to reduce the number of parameters is to incorporate prior knowledge, e.g. scaling or design rules.

The pre-estimation approach can incorporate prior knowledge when creating the function H mapping meta-parameters to pre-estimate. In practice, this would be done by creating training data reflecting the design or scaling rules. If a system fails, of course, the design rules built into the pre-estimation mapping function might be useless. In principle, however, a diagnostic system could provide an alert before failure. Formally, design or scaling information might be exploited in the iterative stages of parameter estimation, as well as in the pre-estimation stage. The assumption made here is that prior information is most useful for reducing the complexity of the *initial* search, and in a final search might obfuscate diagnostic information.

As an example, consider a collection of similar induction machines parameterized by rated power, P. To first order, the lumped model parameters of the machines should scale so that the per-unit parameters are constant. For example, if the induction machines are intended for the same voltage, the real parameters should scale so that

$$\mu_e(P) \approx \mu_e(P_0)\frac{P_0}{P}, \tag{3.52}$$

where μ_e is the vector of electrical parameters $(\,r_r \quad r_s \quad L_l \quad L_m\,)'$ and $\mu_e(P_0)$ is the parameter vector of the canonical machine with power rating P_0. Assume that the damping

$$\beta(P) \approx \beta(P_0)\frac{P}{P_0}, \tag{3.53}$$

and moment of inertia

$$J(P) \approx J(P_0)\frac{P}{P_0}. \tag{3.54}$$

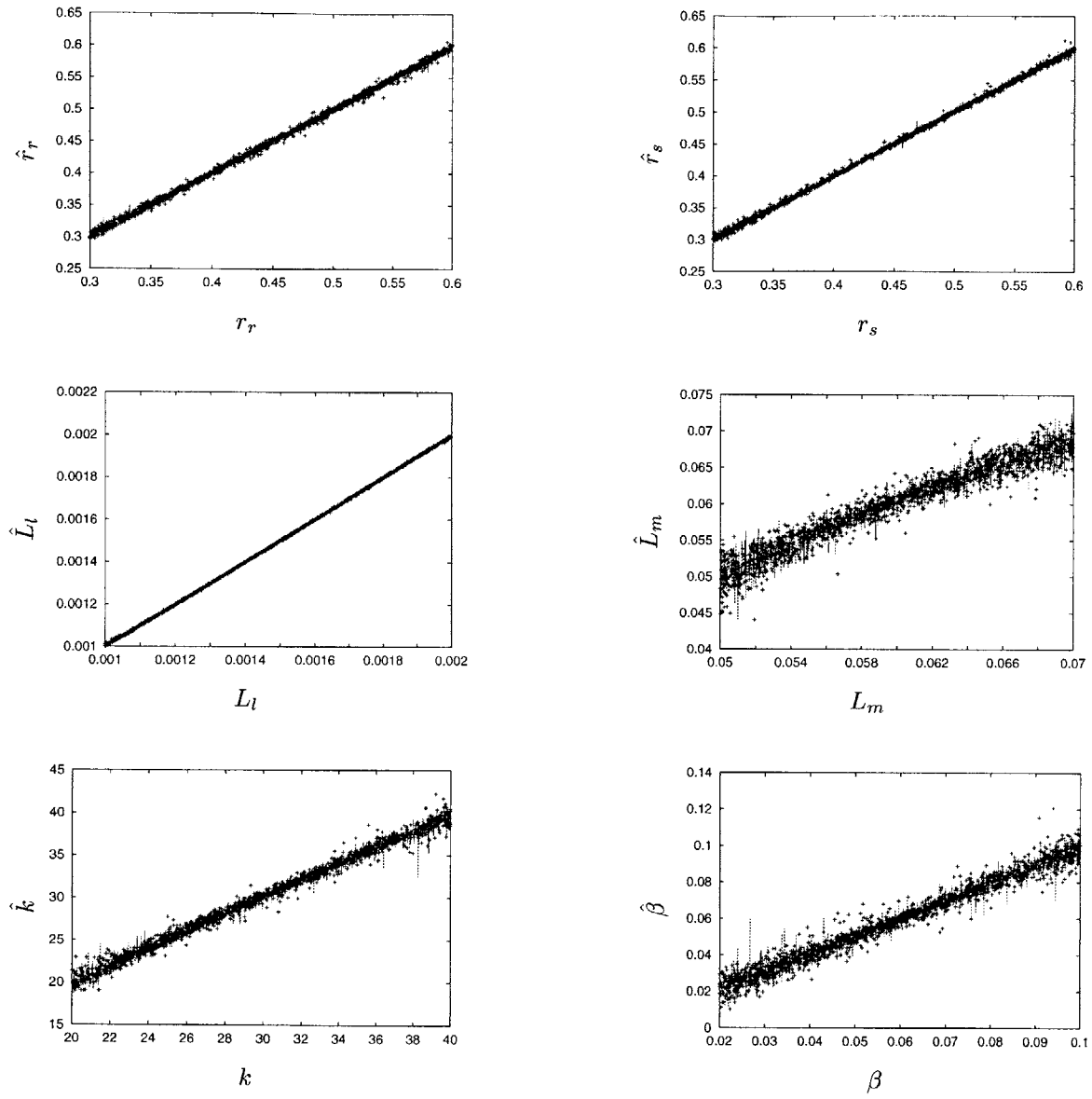


Figure 3-18: Performance of RBF/RBF pre-estimator for induction machine models. Lines indicate performance with parameter set used to obtain RBF fit, points show performance with a cross-validation set.

The expression for the moment of inertia does not follow strictly from a scaling argument, but is illustrative.

The pre-estimator was designed as in §3.4.5, but trained with 5000 randomly selected parameter vectors synthesized according the scaling rules. In addition to scaling the nominal parameters over a factor of four using the base power, the data set contained random deviations of about twenty percent of the nominal parameters. Distinct training and validation sets were constructed using the same procedure. The performance of the pre-estimator using scaling information is shown in Fig. 3-19, which may be compared to Fig. 3-18.

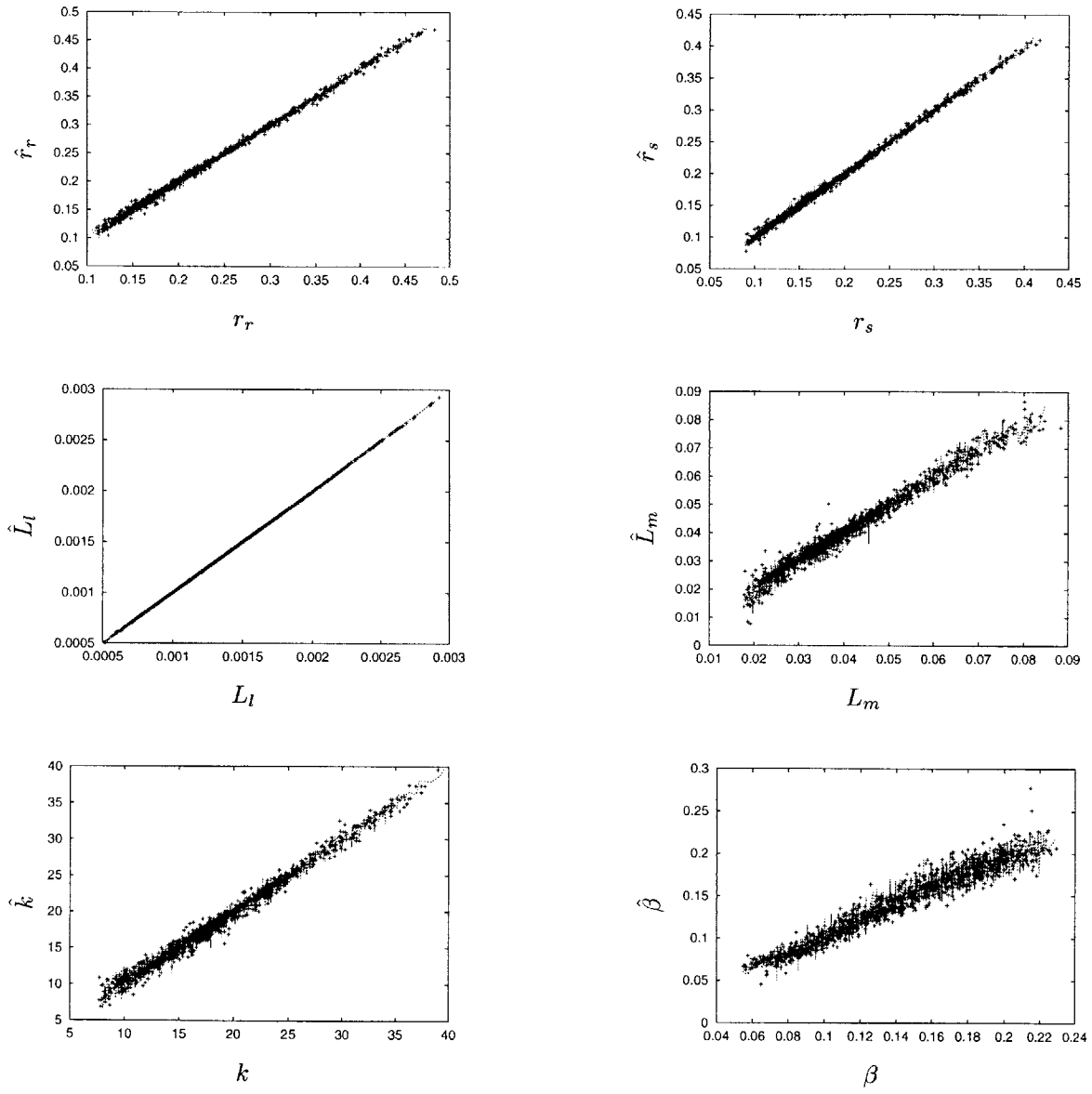


Figure 3-19: Performance of RBF/RBF pre-estimator for an induction machine system, using scaling information. Estimated parameters appear on the vertical axis, true parameters on the horizontal axis. Lines indicate estimates with the parameter set used to fit the output RBF. Points show estimates for a cross-validation set.

3.5 Estimating parameter distributions

Once estimated parameters have been found, it is essential to confirm that the model and its parameters are useful in the intended application. This process is generally called validation. Depending on the intended purpose of the model, validation may include several activities. Model order determination seeks to balance the complexity of the model against the descriptive requirements of the data. Residual analysis is typically used to confirm the assumptions under which the estimation method has provable properties with respect to the estimates. Cross-validation is a check that the predictive capabilities of a model generalize to other data sets not used to fit the model. These activities are interrelated. For example, model order determination requires some metric to assess how well the data fit. These metrics are frequently drawn from the general category of residual analysis. Complete discussion of these issues can be found in any of several estimation [30, 4, 55, 57, 3] or statistics oriented texts [44, 4, 17].

The emphasis in validation depends on the purpose of the model. The nonintrusive diagnostic scenario proposed in this thesis is essentially statistical hypothesis testing, i.e. establish acceptable parameter thresholds and signal a potential problem when thresholds are exceeded. If the thresholds are established based on design or specification information, parameter estimates must be both accurate and precise to minimize the probability of error. Alternatively, if bounds are established relative to estimated parameters, i.e. parameters are allowed to vary by some tolerance from the nominal estimated values of a "known good" system, the parameter estimates need only be precise. The distinction between accuracy and precision is illustrated in Fig. 3-20. In either scenario, the usefulness of a model proposed for diagnostics depends on estimates of the parameter distributions. The following two sections are an overview of procedures to estimate parameter distributions and sensitivity in linear and nonlinear estimation problems. The linear case is important because it is sometimes applied to the nonlinear problems that are more likely to be found in nonintrusive load diagnostic scenarios.

3.5.1 Linear Problems

In a linear problem, finding the distribution of the parameters is straightforward, particularly with the usual white, stationary, Gaussian disturbance assumption. Consider the linear least squares problem, $Ax = b + \epsilon$, where ϵ is a $\mathcal{N}(0, \sigma^2)$, stationary, white-noise process. If $A'A$ has sufficient rank, i.e. $\hat{x}$ is identifiable, this problem has a unique solution

$$\hat{x} = (A'A)^{-1}A'(b + \epsilon) \tag{3.55}$$

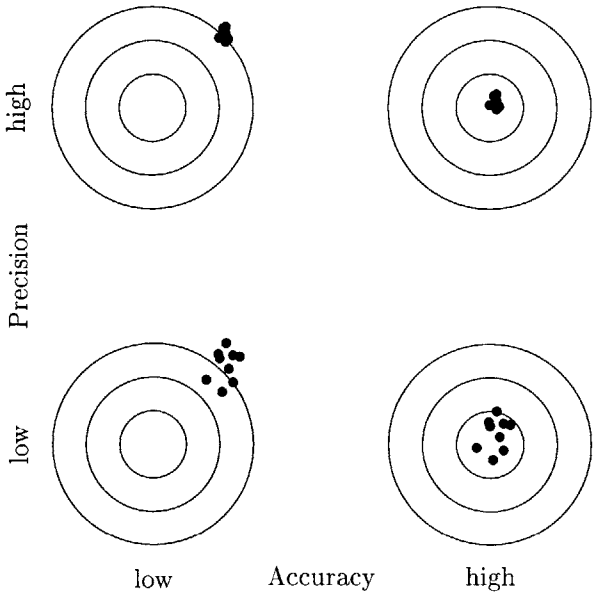


Figure 3-20: Illustration of accuracy and precision. Accuracy and precision requirements depend on the nature of the inquiry.

with an A'-orthogonal residual $r = b+\epsilon-A\hat{x}$. The matrix $C = (A'A)^{-1}$ contains the *standard errors* of the estimates $\hat{x}$. In particular, a variance estimate for the parameters estimate $\hat{x_k}$ is C_{kk}. The standard covariance estimate of parameters $\hat{x}_j$, $\hat{x}_k$ is C_{jk}. Note that for C to have this interpretation, the rows of A and b must be scaled according to the σ of the observations.

Some care is required at this point. Note that if σ is obtained *a posteriori* from analysis of the residual, i.e. $\sigma = \hat{\sigma}$, then the distribution of the parameters is the Student's T and *not* Gaussian; $\hat{\sigma}$ is itself an estimate with a Gaussian distribution. If there are few parameters relative to the number of observations and the disturbance is stationary, the difference between the Student's T and the Gaussian distribution is small [44]. These considerations do not change the scaling properties of the matrices involved, but do impact their interpretation in terms of expected errors in the parameters.

Parameter sensitivity is closely related to identifiability. The relation is most clearly seen by singular value decomposition of (3.55), i.e.

$$\begin{aligned} A\hat{x} &= b \\ U\Sigma V\hat{x} &= b \\ \hat{x} &= V'\Sigma^{-1}U'b. \end{aligned}$$

An identifiability problem, without disturbance, is manifested in exactly zero singular values. Elimi-

nating a zero singular value corresponds to reducing the number of parameters, although in practice, the singular values do not correspond exactly to system parameters $\hat{x}$ but map to the parameter space non-trivially through U and V. If there is an identifiability problem and disturbance in the regressors, the smallest singular values will be non-zero. Correspondingly, there will be large standard errors on the diagonal of C.

The singular value approach quickly leads to a parameter sensitivity estimate in terms of the norm of a perturbation in the observations. Specifically, the norm of the parameter bias u introduced by a perturbation in the observations v, i.e. satisfying $A(x+u) = b+v$, is bounded

$$||u||_2 \leq \frac{1}{\sigma_{min}}||v||_2. \tag{3.56}$$

Observations about the norm of the parameter vector are less useful if the individual parameters vary widely in magnitude. However, (3.56) emphasizes that the sensitivity of the parameters can be approached deterministically, useful if the prerequisites of the stochastic argument cannot be satisfied.

3.5.2 Nonlinear problems

The analysis for the linear estimation problem is often applied directly to the nonlinear case for the linearization at the equilibrium $\hat{x}$ [10, 29]. The analog of (3.55) is

$$\delta = (J'J)^{-1}J'r \tag{3.57}$$

where J is the Jacobian with appropriately scaled rows and r is the residual at $\hat{x}$. Of course, the magnitude of the residuals and the validity of the Jacobian for perturbations around the equilibrium determines the success of this approach. Some work has been done to investigate when the linear approximation is useful; in [10], for example, the authors are particularly concerned about the accuracy of confidence interval estimates based on the linear approximation.

Parameter distribution estimates may be used in nonintrusive diagnostics to decide if diagnostics are feasible in a system, and what the thresholds should be if this is the case. Assuming that the linear approximation is valid for predicting parameter distributions is a very unsatisfying compromise in this context. An alternative technique that requires no assumptions about the linearization or specific properties of the noise involves the simulation of repeated experiments. If an experiment could be performed repeatedly, a parameter distribution estimate could be inferred from the histogram of the parameter over many experiments. This process can be simulated introducing an artificial disturbance with appropriate characteristics and re-estimating parameter values. Repeated

estimation is inherently parallelizable, and the process can be efficient if the parameter distributions are tight enough that a good guess is always available. In [55] this approach is termed "Monte-Carlo simulation of synthetic data sets." A related approach is the bootstrap, which involves estimation using a resampling of the data set. A basic discussion of the bootstrap appears in [55], a more thorough exposition can be found in [77]. The synthetic dataset approach is used in the field study in §4.2.1.

3.6 Interface to NITC

Some "glue" is required to connect a program implementing the system identification methods of this chapter to the nonintrusive classification techniques of Chapter 2. This glue is a small program that extracts diagnostic program names from the text tags and executes them using the "raw data" part of the stream as input. Program 3.1 gives a Perl implementation.

Program 3.1 *Diagnostic dispatch program in Perl.*

```
#!/usr/bin/perl                                                    0
# diagnostic dispatch program
# s. r. shaw, 1999

while(<>) {
    ($time,$name,$idprog,$length,$therest) = split /:/,$_,5;
    read STDIN,$buf,$length;
    open IDP, "|$idprog" or die "Can't spawn $idprog: $\n";
    $newtag = join ':',$time,$idprog,$length;
    print IDP $newtag,"\n";
    print IDP $buf;                                                10
    close IDP;
}
```

Diagnostic programs are run one at a time so that the system remains "lightly loaded" and time sensitive tasks are not pre-empted. On an multiprocessor system where net performance would be increased by increasing the number of tasks, it would be relatively simple to modify Program 3.1 to run several diagnostic tasks at one time.

An advantage of dispatching individual diagnostic programs instead of feeding the diagnostic stream into one large program containing all the models is that the programs can be self-contained scripts in a language convenient for handling data. One practical possibility is to write wrapper programs that prepare the data for more generic identification programs by filtering, subtracting the steady state, and locating the start of the transient. Program C.10 is an example of a wrapper that performs these tasks for the DC fan motor system covered in more detail in §4.3.1. Program C.10 also reads the output of a generic identification program and prepares a graphics stream for display using `xnilm` or `w3nilm`.

In addition to patching together diagnostic streams, graphics and identification programs, wrappers are a convenient place to do pre-estimation or other pre-processing activities.

3.7 Summary

The diagnostic role of the NILM can be approached as a problem of identifying physical models. With parameters available any of several well-established methods, e.g. threshold testing, might be used to detect faults. The difficult step, and the emphasis of the chapter, is automated system identification of the physical models.

A major problem in unsupervised system identification is lack of suitable starting values for minimization of the loss function. This problem is particularly serious if one model and initial guess is used to fit several different loads, as in the context of nonintrusive load monitoring. Two approaches are given for handling the problem.

The techniques of §3.3 can be compared to closing a zipper. The minimization begins with low-time data, and attempts to match low-time data before moving on to high-time data. The methods of §3.3 can be related to the extended Kalman filter, and also have some parallels in VLSI placement techniques that alternate partioning steps with optimization steps [34].

Section 3.4 comprises a collection of suggestions for forming an "inverse model" that maps system responses to parameters. The first step of the procedure is to model the system response with an arbitrary or semi-physically motivated structure to obtain meta-parameters. These meta-parameters should have good noise properties, and the model used to obtain them should be easy to solve and represent the data well. These requirements may be met by familiar linear system identification techniques – many linear models fit the responses of nonlinear problems acceptably. The meta-parameters are then mapped with a black box to the desired parameters.

The tools of this chapter help solve system identification problems in the context of nonintrusive diagnostics. That context includes several challenging attributes that make the tools valuable. The methods must work unsupervised, with no intervention from a trained operator. Performance with a poor initial guess is essential, since one initial guess may be required to fit all transients matching one or more exemplars. Finally, ease of model specification is a key feature. The methods require no effort in excess of the work required to write a simulation model. The methods of this chapter, as implemented in §C.2, are used for the identification and diagnostic case studies of Chapter 4.

Chapter 4

Results

The experimental results in this chapter appear in three sections reflecting the main contributions of the thesis; the nonintrusive transient classifier (NITC), system identification, and NITC combined with system identification. Demonstrations of transient classification are mostly screens captured from the `xnilm` program with scenarios from automobile and building level environments. The system identification methods of Chapter 3 are demonstrated on data collected from loads in the Iowa Energy Center's Energy Resource Station (IEC/ERS). Finally, examples of system identification combined with load classification are drawn from AC and DC systems.

4.1 Nonintrusive transient classifier

The nonintrusive transient classifier was tested in both AC and DC environments. An automobile with a conventional twelve-volt power distribution system was used for DC testing. Building level performance was validated in the "mock-building," a representative collection of loads also used in [38], and in a circuit servicing a laundry facility in a dormitory at MIT.

4.1.1 NITC in an automobile

A 1986 Chevrolet Nova was used for testing the NITC in an automotive environment. A block diagram of the system is shown in Fig. 4-1. Principle components are the transducers, instrumentation amplifier, analog-to-digital converter, and portable computer. In addition, as shown in the diagram, all instrumentation is powered by an independent battery-based power supply.

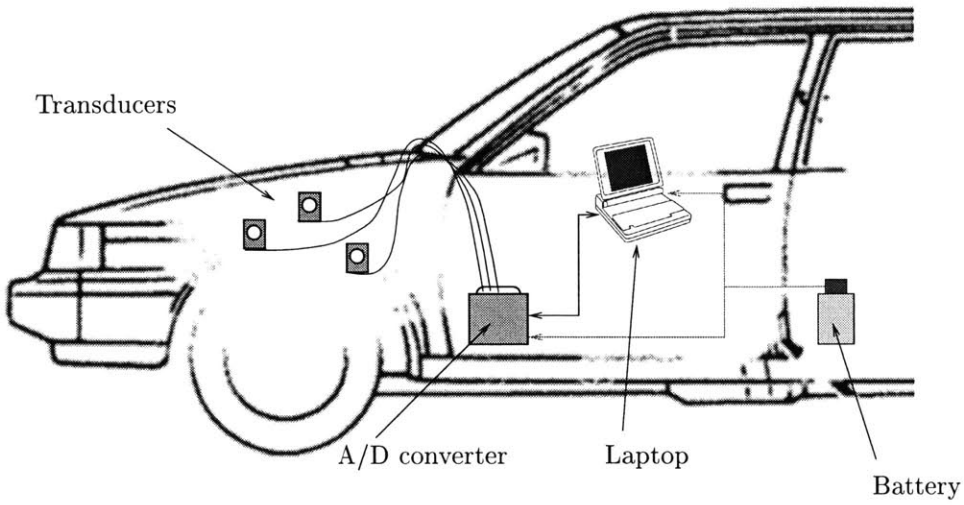


Figure 4-1: Block diagram of NITC system installed in a 1986 Chevrolet Nova for DC field tests.

Instrumentation

A working model for the electrical system in the test vehicle is shown in Fig. 4-2, which is drawn to reflect the spatial orientation of the components. Relatively high currents make consideration of parasitic resistances important. For example, the voltage on the battery is not generally the voltage on the alternator, due to the relatively large currents carried by the economically-sized wire connecting the two sources. The loads shown in Fig. 4-2 are dispersed spatially between alternator and battery; most but not all connections originate at the positive terminal of the battery.

All currents returned from the loads through the chassis are measured as shown in Fig. 4-2. This choice of measurements eliminates consideration of the relatively high frequency ignition coil current and the ripple between alternator and battery. Although these currents might contain interesting information, they would require a sampling rate considerably higher than needed for transients due to other loads in the vehicle. Fortunately, the ignition coil and other undesirable signals are easily separated from other load currents by the choice of measurements shown in Fig. 4-2. Measurement choices other than that shown in Fig. 4-2 might be attractive. To reduce cost, a single current sensor on just one return wire might be feasible. A single sensor would be especially interesting in conjunction with software techniques to eliminate the ignition coil and battery ripple currents.

The current outputs of the LEM LA55-P transducers shown in Fig. 4-2 are summed together by connecting all three outputs at the node of the data acquisition unit. This summing operation has the effect of canceling the measurement of any currents from loads bonded to the engine block. For example, return current from the starting motor flows primarily from the engine block to the negative terminal of the battery, but some current will flow through the two chassis-to-block connections (i_1

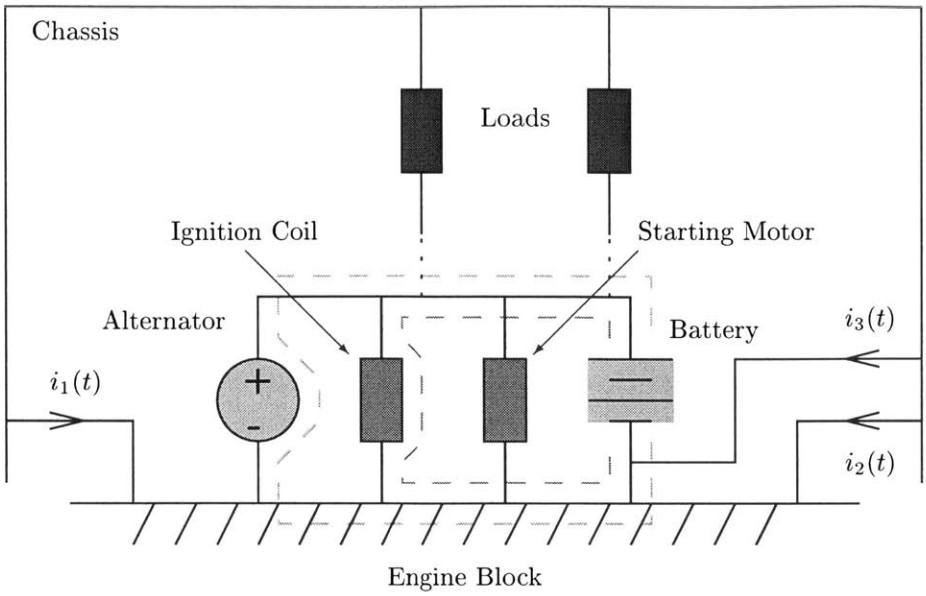


Figure 4-2: Working model of experimental car electrical system. Dashed lines indicate nominal current paths for the alternator and ignition coil. Measuring currents returned from the chassis, as shown, helps avoid the signals associated with the alternator and coil. Any fraction of the current from these loads that does flow through the chassis is canceled in the quantity $i_1 + i_2 + i_3$ used for monitoring.

and i_2) and return to the battery through the battery-to-chassis connection i_3. These currents cancel in the sum $i_1 + i_2 + i_3$, which is the measured quantity. Other undesirable signals include the charging current between the alternator and battery, and the periodic high-frequency signal due to the ignition system. Although these currents primarily return to the battery through the engine block, some fraction does flow through the engine block and the chassis to the battery. The engine block is electrically bonded to the chassis only by the three grounding connections shown in Fig. 4-2.

The location of the Hall-effect current sensors in the vehicle is shown in Fig. 4-3. The transducers were installed on each of the three wires connecting the battery and engine block to the chassis. Signal and power wires from the transducers are routed through an unused rubber gasket and hole in the firewall. The analog-to-digital converter, laptop computer, and independent power source are stowed in a plastic enclosure in the passenger compartment.

The combined signal from the current sensors is sampled at 150 Hz, using a 12 bit LT1294 A/D converter controlled by a 16C84 PIC micro-controller running at 10 Mhz. In addition to controlling the A/D converter, the PIC microcontroller also transmits measured data over a 32 kbps RS-232 connection to the laptop computer. Commercial equivalents of this custom data acquisition system exist and could also be used.

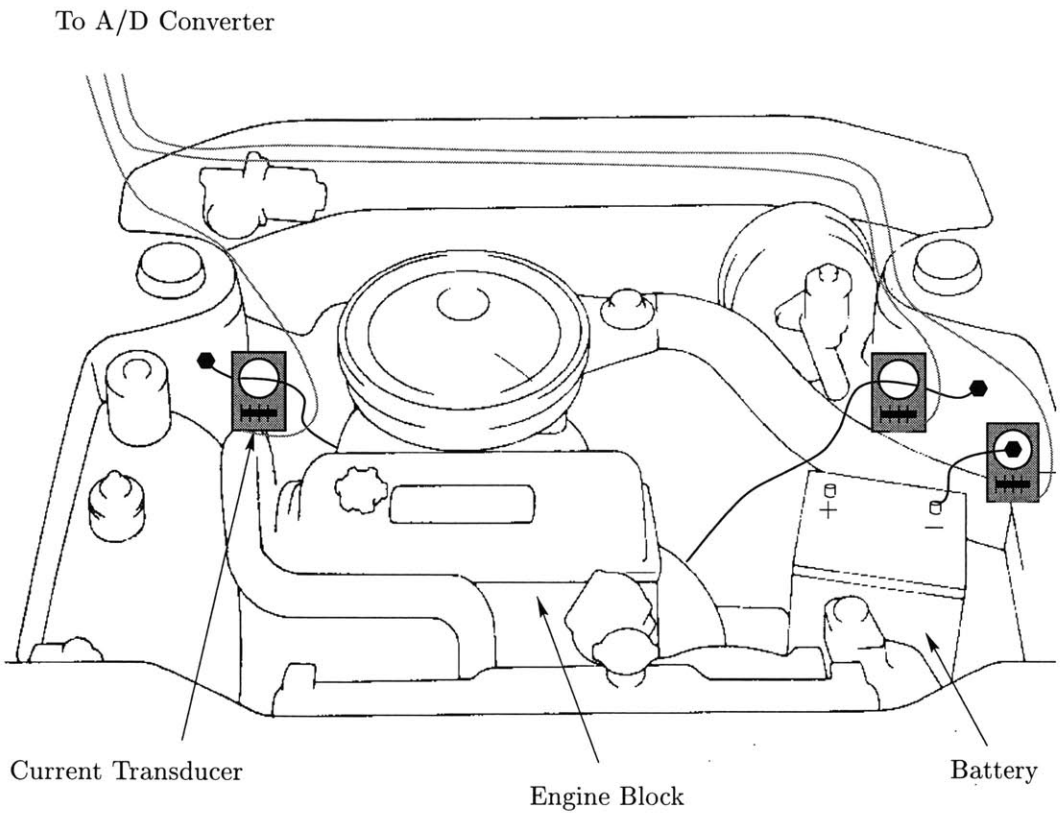


Figure 4-3: Location of sensors in vehicle.

Results

Loads in the car are mostly lights and motors. For testing, the NITC was supplied with nine exemplars for loads including the hazard lights, brake lights, fan, headlights, and cigarette lighter. Of these loads, the various lights were frequently confused – not an error, but a consequence of scaling the exemplars for the best possible fit. In particular, scaled incandescent lightbulb transients are very similar, because a large incandescent lightbulb performs the same physical task as a small lightbulb. In an application requiring that the headlight be distinguished from the dome light, for example, limits could be established to prevent the large headlight exemplar from scaling to fit the relatively small dome light transient. Alternatively, the scaling properties of the exemplars could be retained, and different sized loads could be discriminated based on their scale factors. Some examples of NITC in the automotive environment are given in Fig. 4-4 and Fig. 4-5. An example of fan load detection can be found in §4.3.1.

Figure 4-4 shows a headlight transient followed immediately by another similar transient. Figure 4-4 was created by the rotation of the headlight control by the driver. The first transient is the parking lights, the second is the headlights. The spacing of the two transients depends on the speed of rotation of the control. Other user controls in the automobile also have the property that the transient associated with a desired setting may be preceded by several transients as the control is moved past the intervening settings. In cases like this where transients occur in succession by design of the control, exemplars will need to accommodate both widely separated sequences of individual transients corresponding to slow control actuation and cases where transients are stacked together due to fast actuation.

Figure 4-5 shows a dome light transient classified as a headlight transient with a scaling factor of 0.21. The exemplar matches the data remarkably well, and demonstrates the similarity between large and small incandescent lightbulb transients. Figure 4-5 is also interesting because the small scale of the dome light makes it easy to see a boxcar-type signal immediately before and in the steady-state following the lightbulb transient. This signal, which is difficult to see in Fig. 4-4, may be due to an unmodeled load.

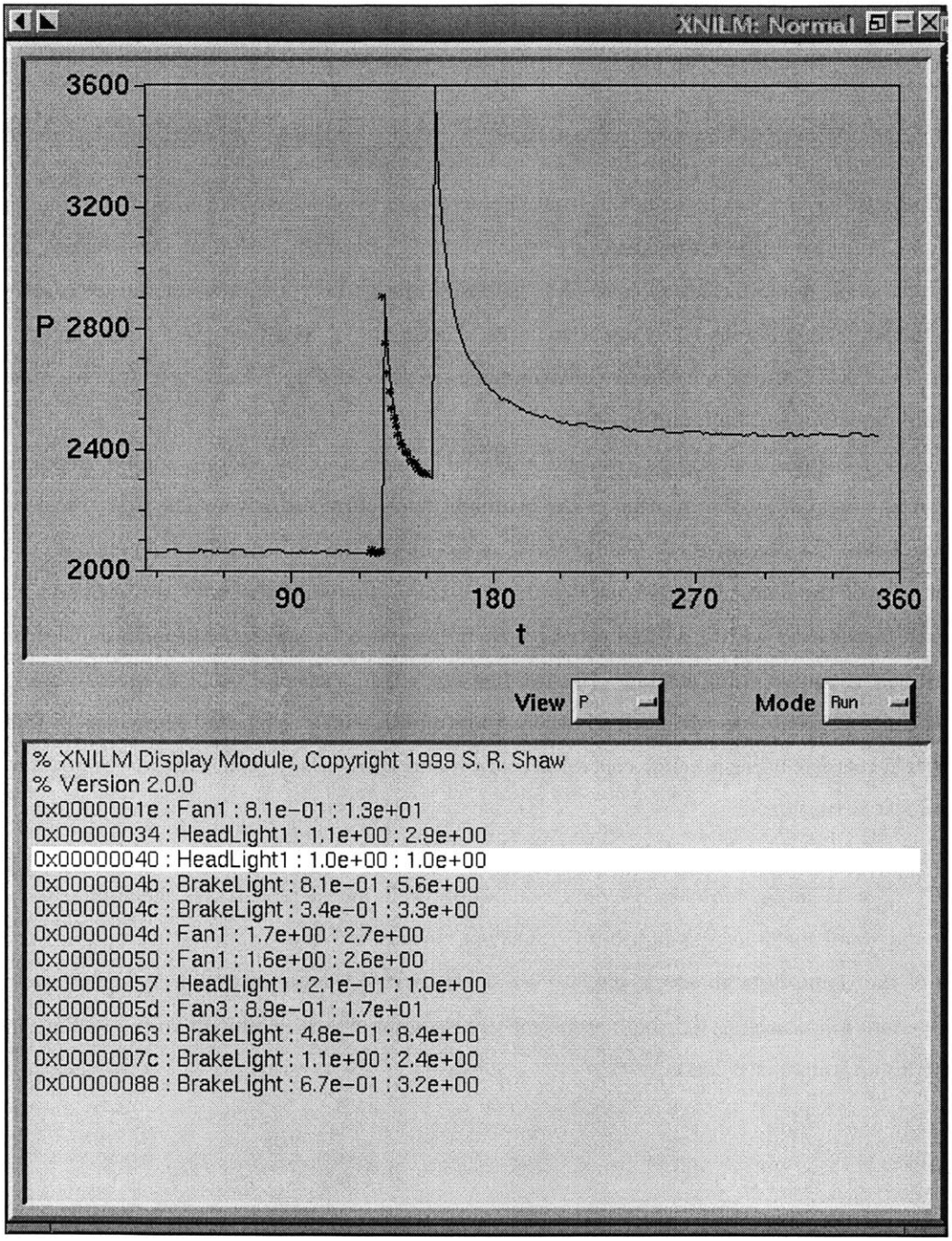


Figure 4-4: Screen shot of `xnilm` showing detection of headlight transient. The gain (first number following text part of tag) of nearly one indicates that no scaling was done. Solid lines in the graph are measured data; the overlayed points show the fit of exemplar to transient.

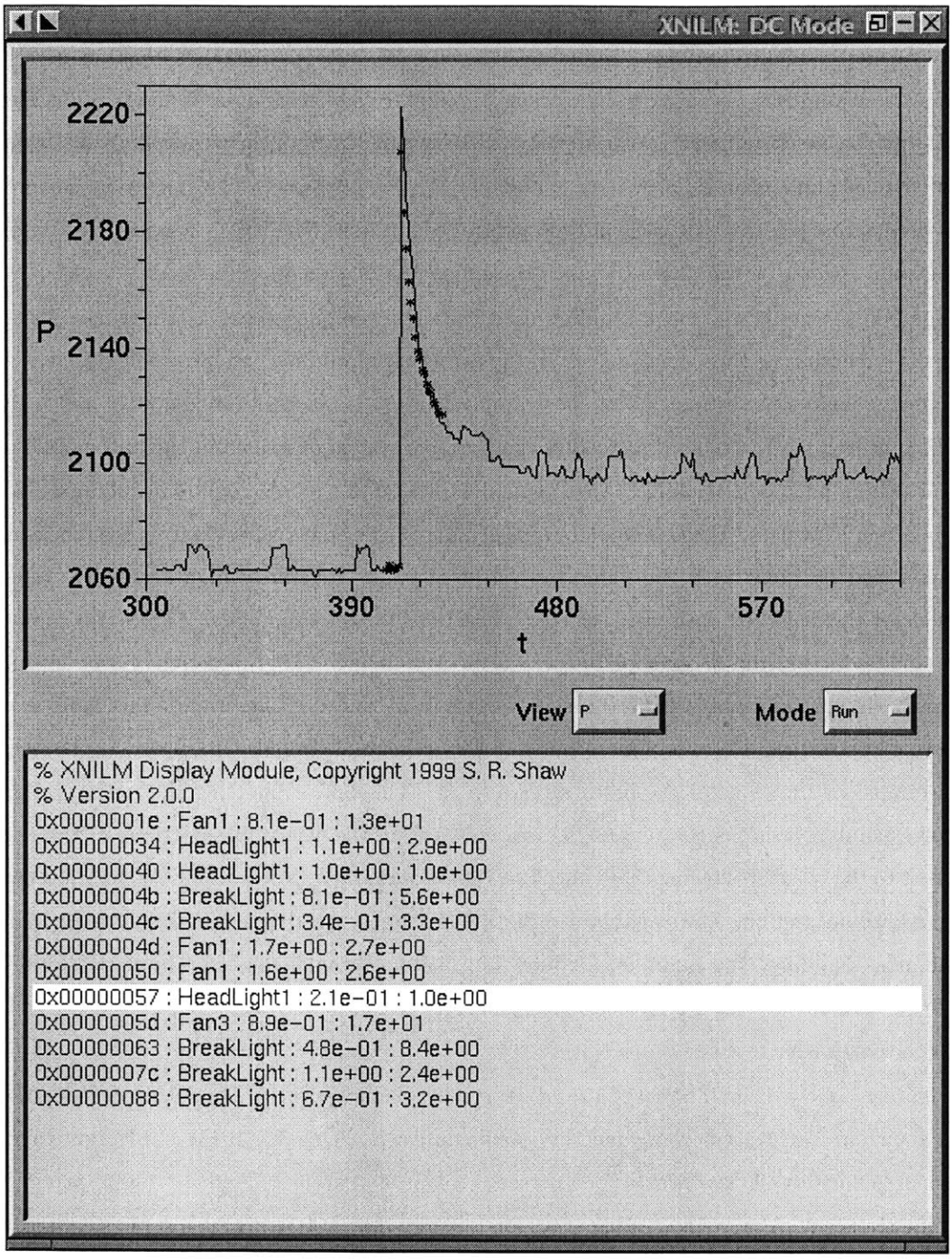


Figure 4-5: Screen shot of `xnilm` showing detection of a transient from the dome light. Solid lines in the graph are measured data; the overlayed points show the fit of exemplar to transient. Notice the level and character of the disturbance.

4.1.2 NITC in the mock building

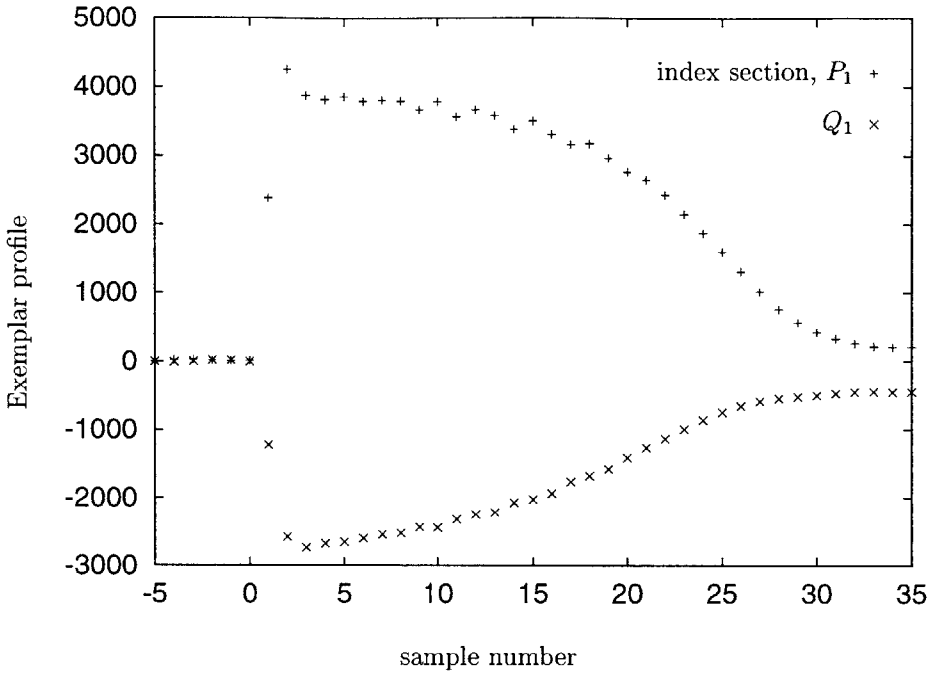
The mock building consists of a circuit breaker panel, wiring, loads and relays representing a scaled-down version of a medium sized commercial facility [38]. Loads include induction machines, a computer, instant start, compact fluorescent, rapid start and incandescent lights. The electrical service in the mock building is three-phase with neutral. All single phase loads are connected between phase and neutral, providing a nominal 120 VAC service. The induction machines are connected across the three phases, with no neutral connection. The NITC is configured to measure voltage and current on the phase to which the single phase loads are connected.

The NITC tests in the mock building used the same compiled codes as those used in the car, with the exception of the device driver and preprocessing routines. Preprocessing for an AC environment requires the estimation of spectral envelopes. Also, a desktop computer and a PCL818 data acquisition card were used instead of the portable system described in the automobile tests.

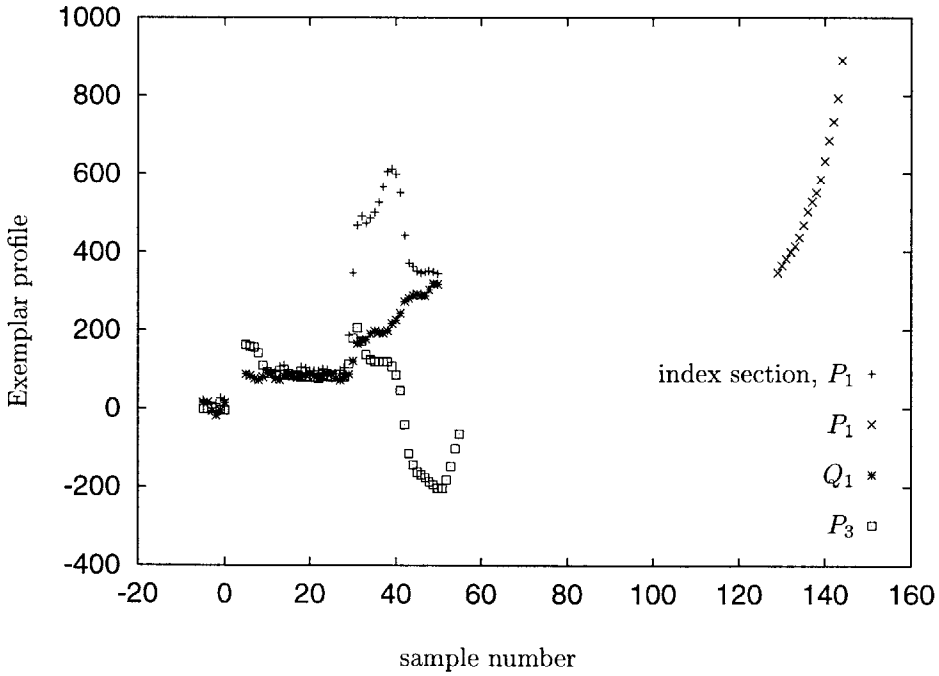
For testing purposes, the NITC was supplied with a set of eight exemplars including at least one exemplar per load class. These exemplars were generated using data collected from the mock-building and modified by hand with the `vsection` script of §C.3. Relatively high-derivative, repeatable portions of the observed data were selected as sections for the exemplars. Figure 4-6 shows rapid start and induction motor exemplars for the mock building. The induction motor exemplar is relatively simple, and most of the transient occurs in P_1 and Q_1. The rapid-start lamp exemplar is more complicated. There is a large but not very repeatable impulse-like spike at the beginning of the transient that locates the index section in P_1. The data supporting the index section (and its corresponding sections in Q_1 and P_3) are drawn from the repeatable portions of the transient adjacent to the spike. Following these initial sections is a characteristic jump in P_1, which is captured by an additional section. The exemplars in Fig. 4-6 are simply starting points that have proved useful in the mock building. For example, the induction motor exemplar might be improved by breaking the single sections in Fig. 4-6 into two sections. This would be particularly useful if the induction machine took longer to come up to speed.

Figure 4-7 shows classification of a rapid-start lamp bank transient. The rather detailed behavior in P_1 shown in Fig. 4-7 is accompanied by useful content in P_3, shown in Fig. 4-8. Figure 4-9 shows transient behavior and classification of a personal computer transient. Figure 4-10 shows a induction machine transient. A more interesting situation is shown in Fig. 4-11, where a rapid start lamp bank is correctly classified from sections that bracket an incandescent lamp bank transient. Figure 4-12 shows that the incandescent transient is also correctly classified.

From the performance point of view, it is interesting to note that on a 450 Mhz Pentium II, the NITC processed the data used for the figures in this section in nineteen seconds of CPU time. This corresponds to a processing rate of about 480 kB/s, which compares favorably with the sampling



a.



b.

Figure 4-6: Plots of exemplars for induction motor (a) and rapid-start lamp (b). The sampling rate is 120 Hz.

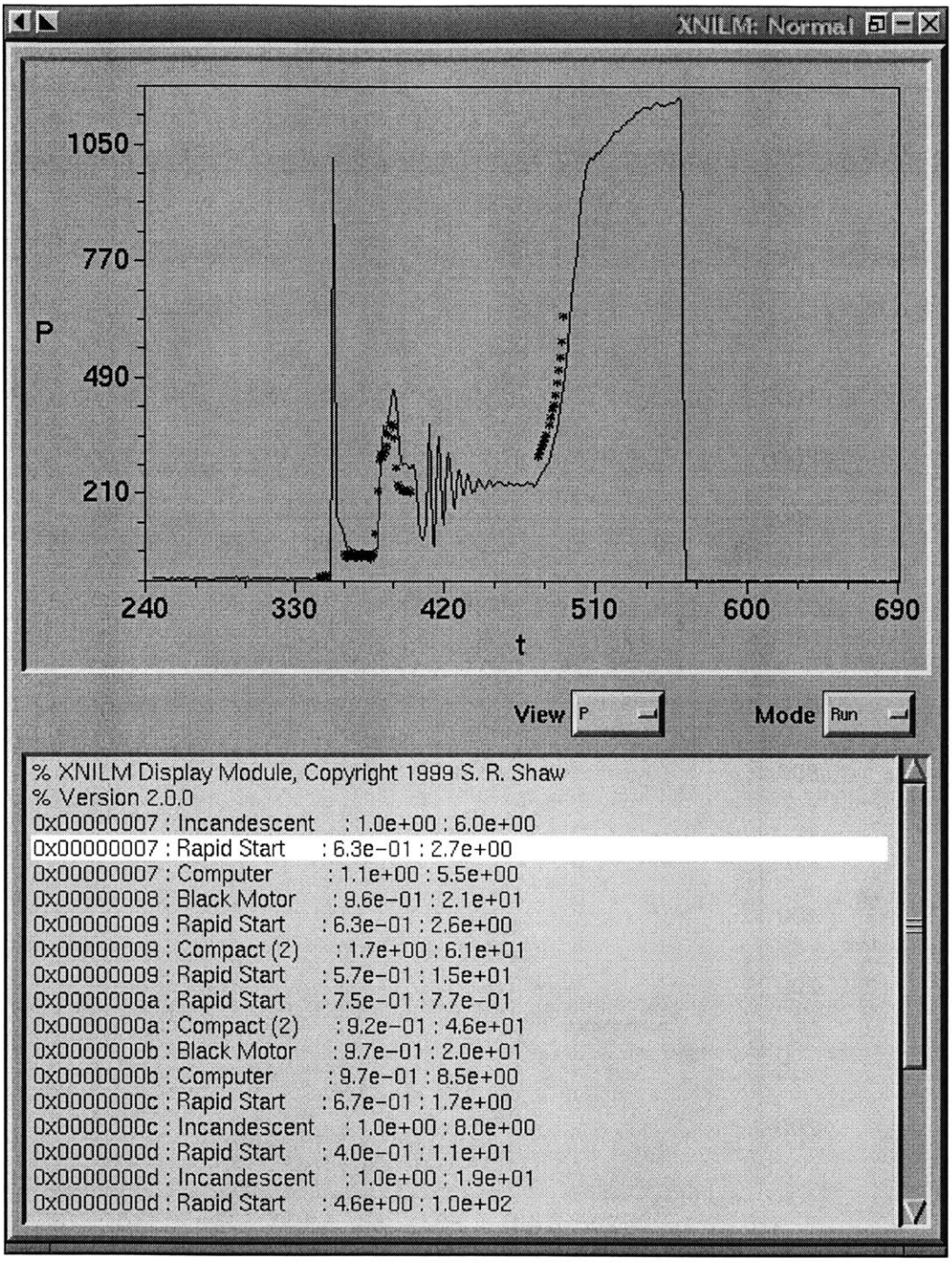


Figure 4-7: Screen shot of `xnilm` showing detection of a rapid-start lamp bank transient. Based on repeated observation, the decaying oscillation between the two sections of the exemplar is a portion of the transient which is not reproducible. Solid lines in the graph are spectral envelope data; the overlayed points show the fit of exemplar to transient.

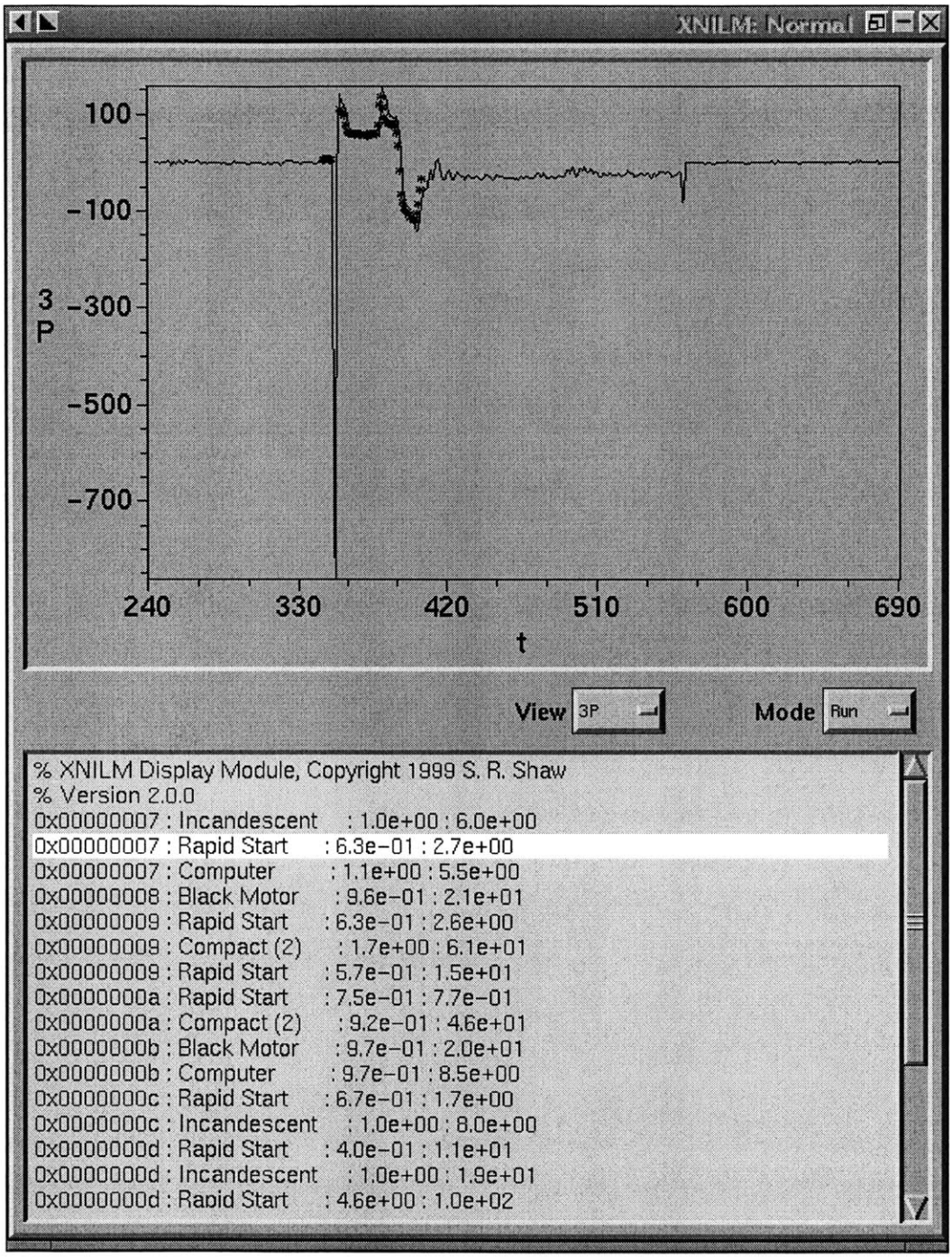


Figure 4-8: Match of rapid-start lamp bank transient in P_3.

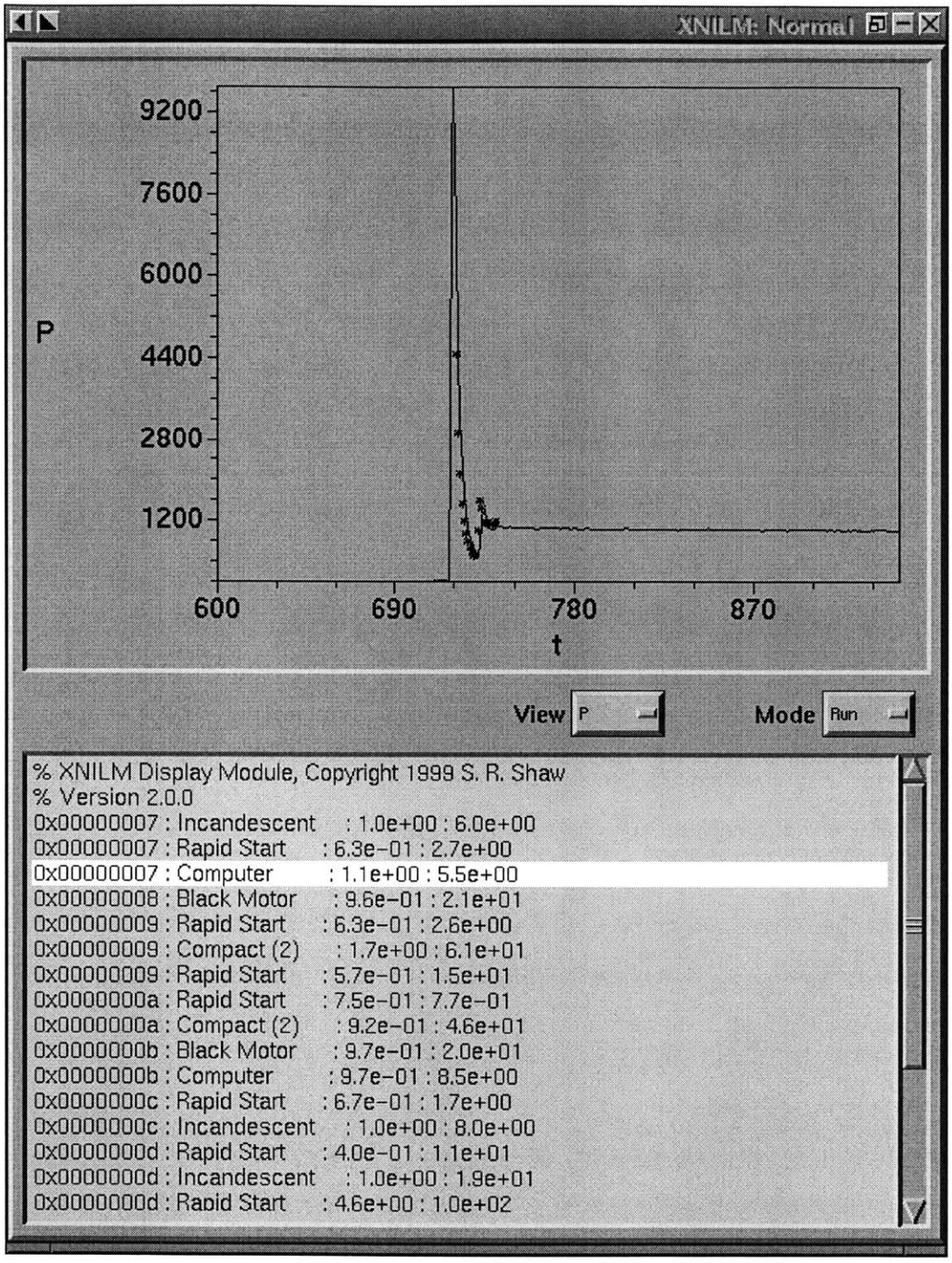


Figure 4-9: Screen shot of `xnilm` showing detection of a computer transient. Solid lines in the graph are spectral envelope data; the overlayed points show the fit of exemplar to transient.

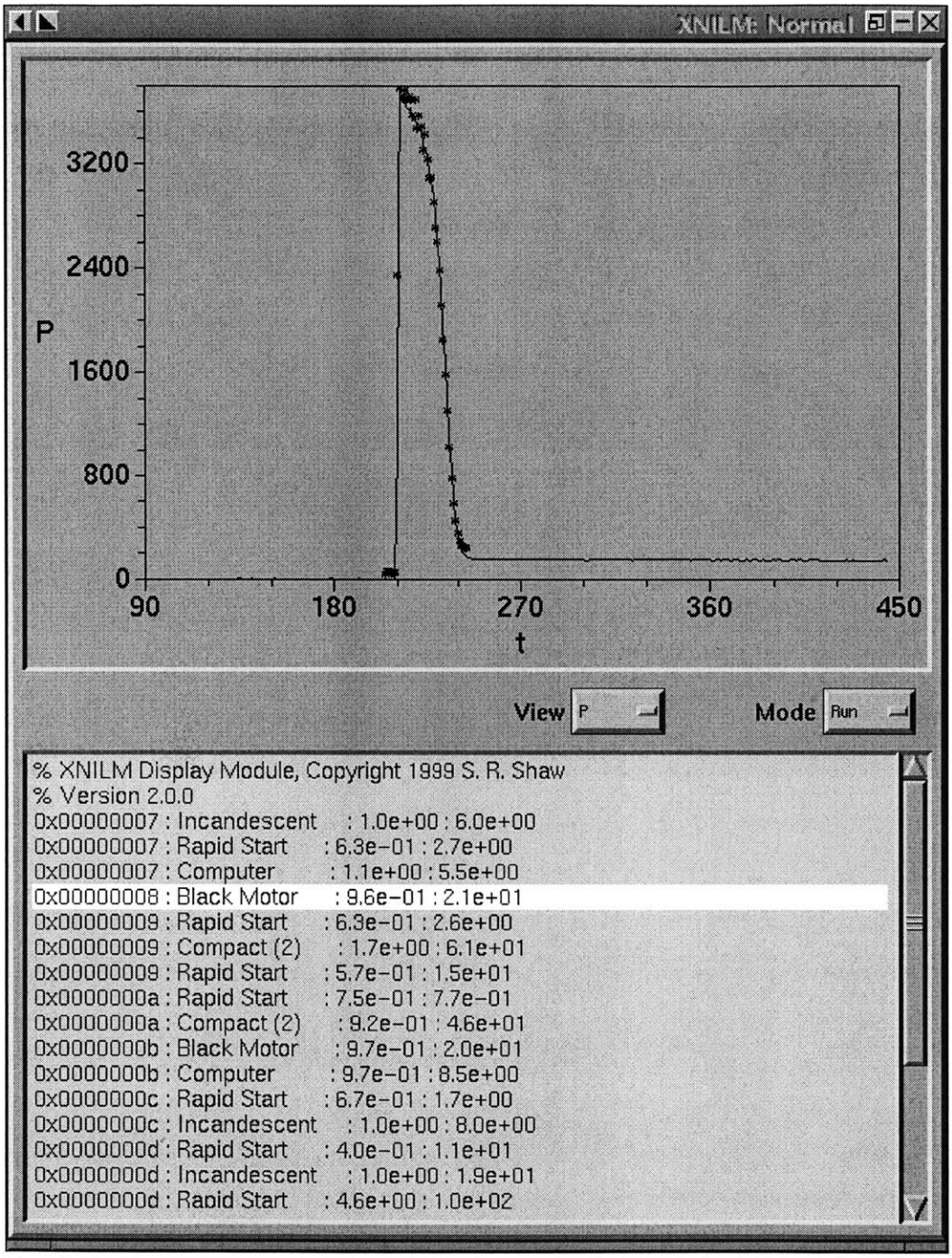


Figure 4-10: Screen shot of `xnilm` showing detection of an induction motor transient. Solid lines in the graph are spectral envelope data; the overlayed points show the fit of exemplar to transient.

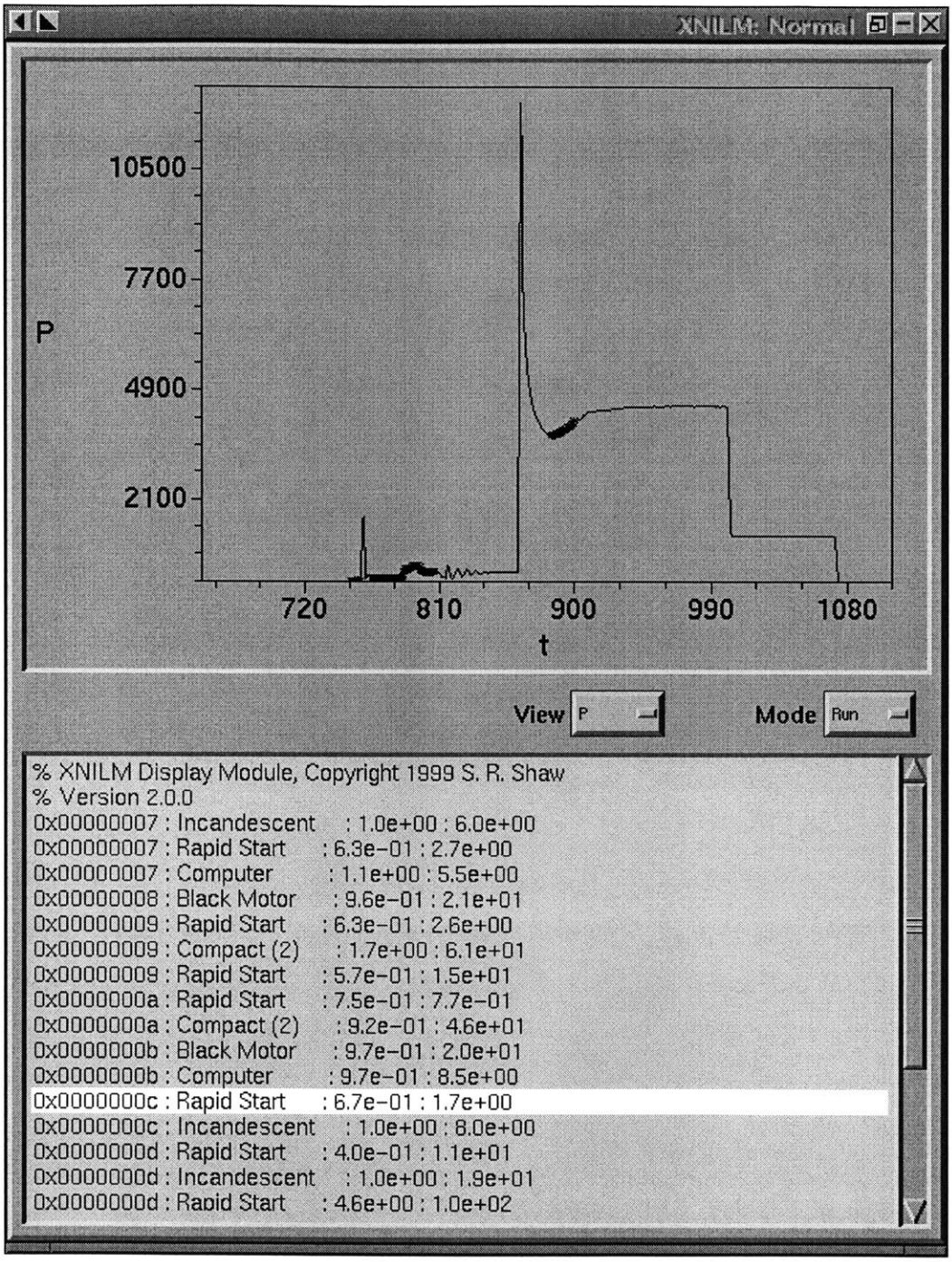


Figure 4-11: Screen shot of `xnilm` showing detection of a rapid-start lamp bank, with an incandescent light bulb transient between the two matched sections. Solid lines in the graph are spectral envelope data; the overlayed points show the fit of exemplar to transient.

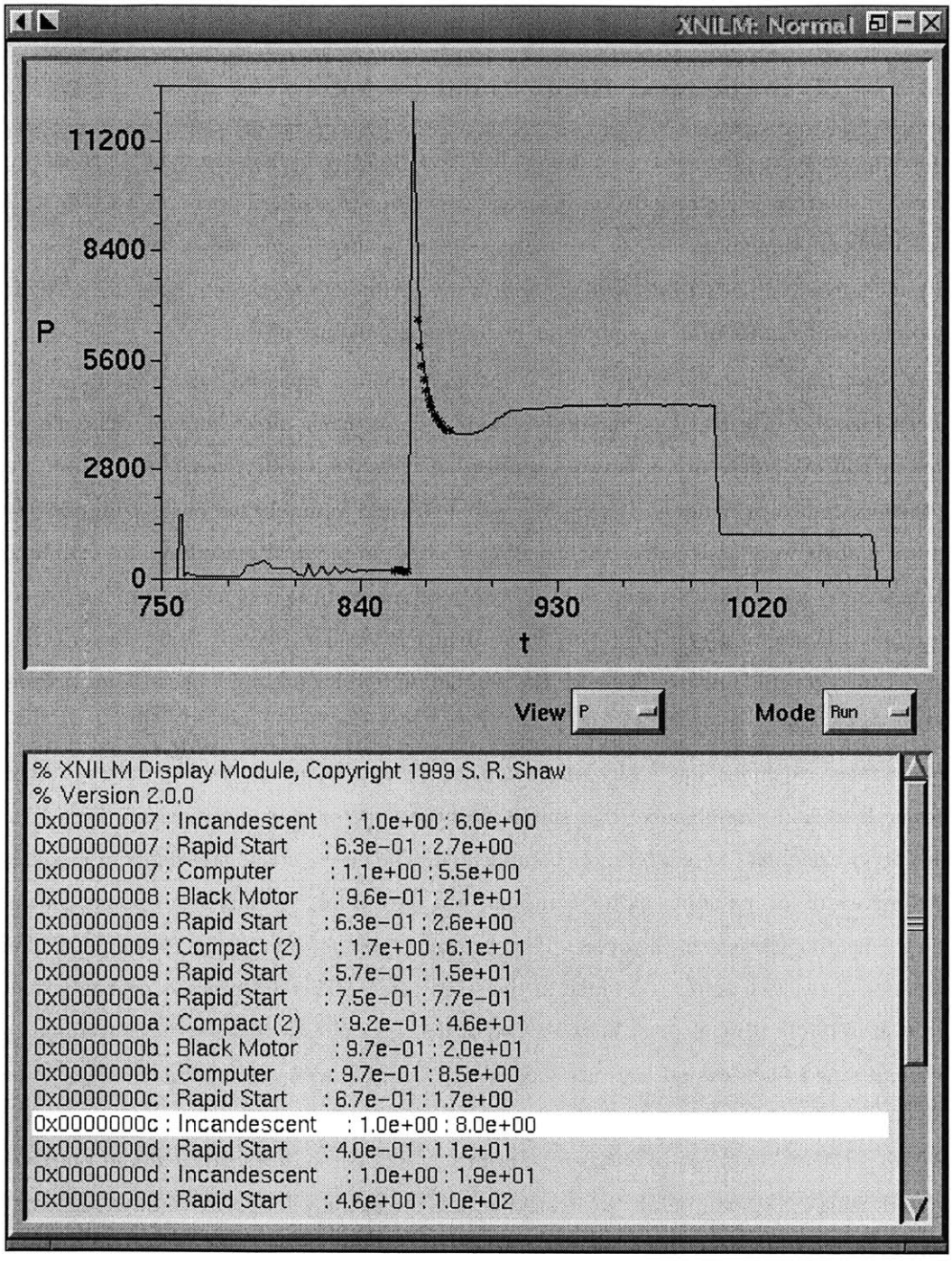


Figure 4-12: The same sequence as shown in Fig. 4-12, but showing detection of the incandescent bulb transient.

rate of about 32 kB/s. Of course, these numbers depend on the number of exemplars, and the processing rate figure is conservative because it includes the fixed cost of starting up the program.

4.1.3 NITC in the Next House laundry room

The main power room of an MIT dormitory called Next House was selected as a final test environment for nonintrusive transient classification. Current and voltage transducers were installed on a 120 VAC phase-to-neutral, three-phase feeder serving the laundry facility. Loads known to be connected to this service include a ventilation fan, washers and dryers. Although all three phases are instrumented, the monitor uses signals from one phase as in the mock building.

The Next House installation is the first instance where a transient-type nonintrusive monitor has been installed in a real building where loads are not under direct control. The monitor was trained by collecting data over a network connection while the laundry was in actual use. The data were then examined by hand, and exemplars were extracted from selected parts using the `vsection` program. Labels were assigned to the exemplars based on physical reasoning and examination of the transients – an approach that would probably be impractical in a more complicated scenario. The exemplar labels are Dryer Heater A, Dryer Heater B Washer, Dryer A Start, Dryer B Start and Dryer C Start. There is no exemplar for the ventilation fan, for example, because its transient was not observed in the training data.

The washer exemplar was easily selected from the exemplars in the test data set because of its power level and resemblance to a simple induction motor startup transient, Fig. 4-13. The various dryer exemplars were a bit more complicated. The dryer "start" transients in Fig. 4-14 show characteristics of an induction motor transient, but have a large steady-state power consumption due to the heating element in the dryer. The two exemplars shown in Fig. 4-14 are quite different in the amount of reactive power consumed in the steady state. This difference is probably due to the connection of the heating element between two of the three phases of the service. Assume that the three phase line-to-neutral voltages are labeled $v_{a,b,c}$, with corresponding currents $i_{a,b,c}$, and that the NITC measures currents and voltages i_a, v_a. A load connected across phases and sinking current from i_a experiences a voltage of either $v_a - v_c$ or $v_a - v_b$. For such a load, real and reactive power computed using v_a, i_a will be off by a rotation, i.e. a resistor will appear to have reactive power. For example, with balanced three-phase voltages

$$\begin{aligned} v_a &= V\sin(\omega t) \\ v_b &= V\sin(\omega t + \frac{2\pi}{3}) \\ v_c &= V\sin(\omega t - \frac{2\pi}{3}), \end{aligned} \tag{4.1}$$

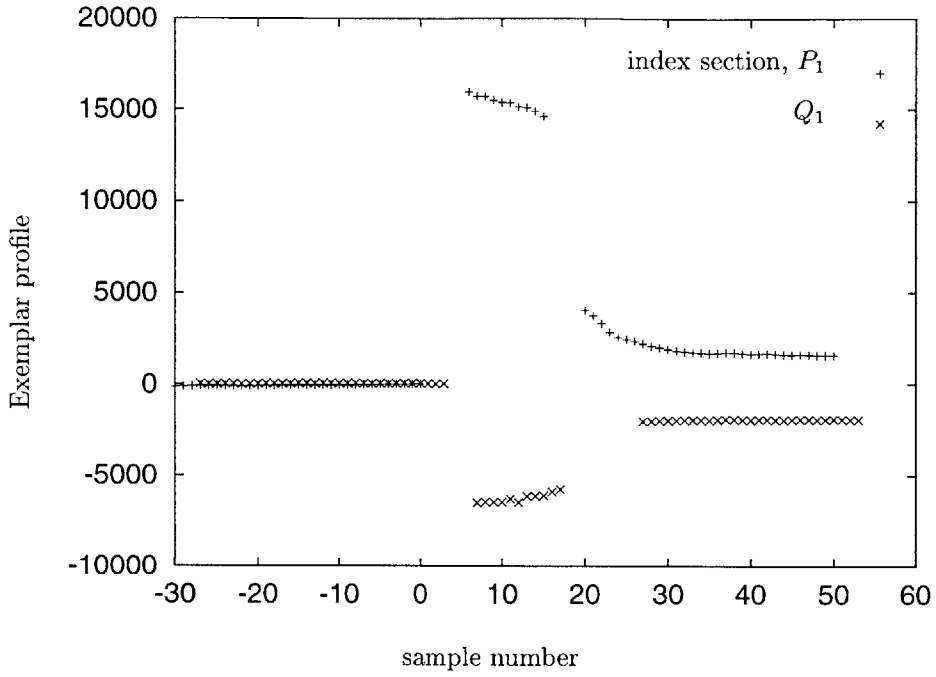


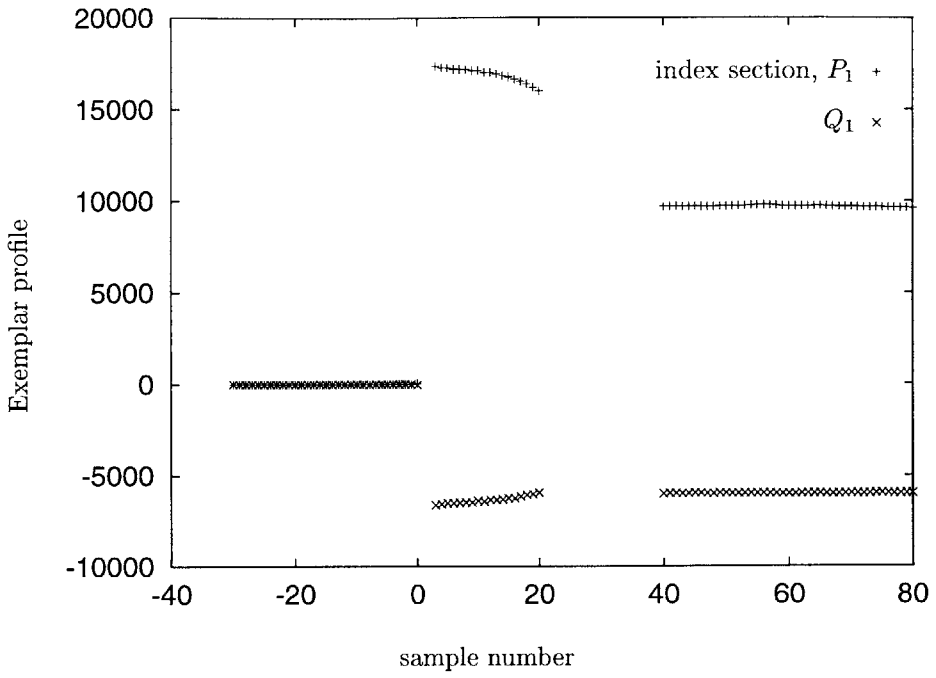
Figure 4-13: Plot of washer start exemplar – essentially an induction motor. The sampling rate is 120 Hz.

the current for a resistive load connected between phases a and c will be

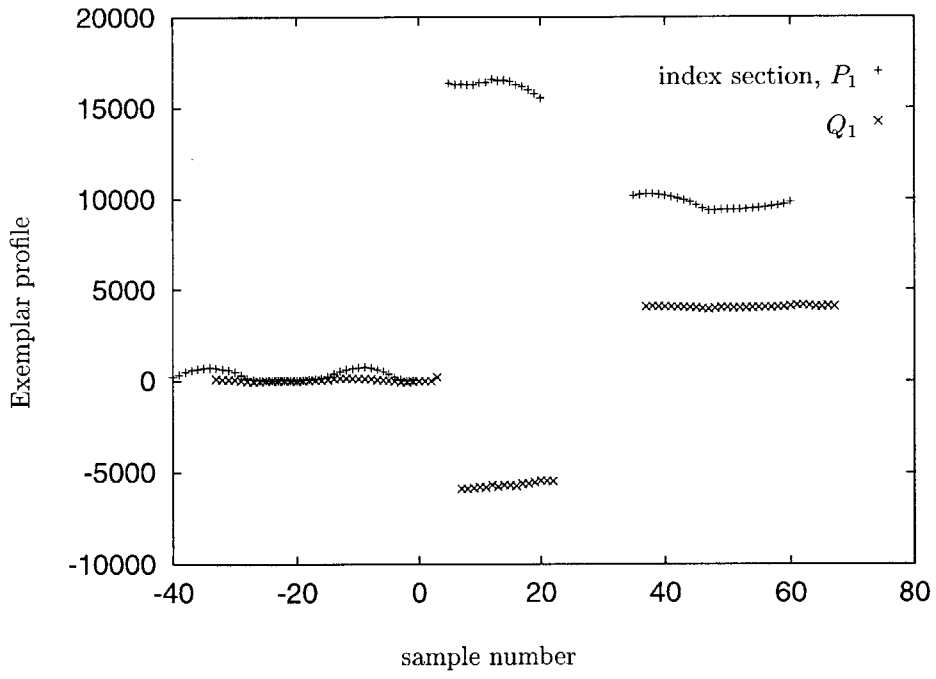
$$\begin{aligned} i_a &= \frac{V}{R}(\sin(\omega t) - \sin(\omega t - \frac{2\pi}{3})) \\ &= \frac{V}{R}(\frac{3}{2}\sin(\omega t) + \frac{\sqrt{3}}{2}\cos(\omega t)). \end{aligned} \tag{4.2}$$

In effect, the cosine term will appear as reactive power. This analysis is consistent with the differences between the "A" and "B" dryer start exemplars in Fig. 4-14.

The exemplars for the Next House installation are supported by data in P_1 and Q_1. In this particular situation, higher-order spectral envelopes did not provide repeatable or distinctive patterns for classification. Initial results from the Next House installation are encouraging. Reliability in classifying transients seems comparable with the performance in the mock building, and the software and hardware have demonstrated the ability to work continuously, unattended. Typical classification results are shown in the web browser screen shots in Fig. 4-15 and Fig. 4-16.



a.



b.

Figure 4-14: Plots of "Dryer A Start" exemplar (a) and "Dryer B Start" exemplar (b). The sampling rate is 120 Hz.

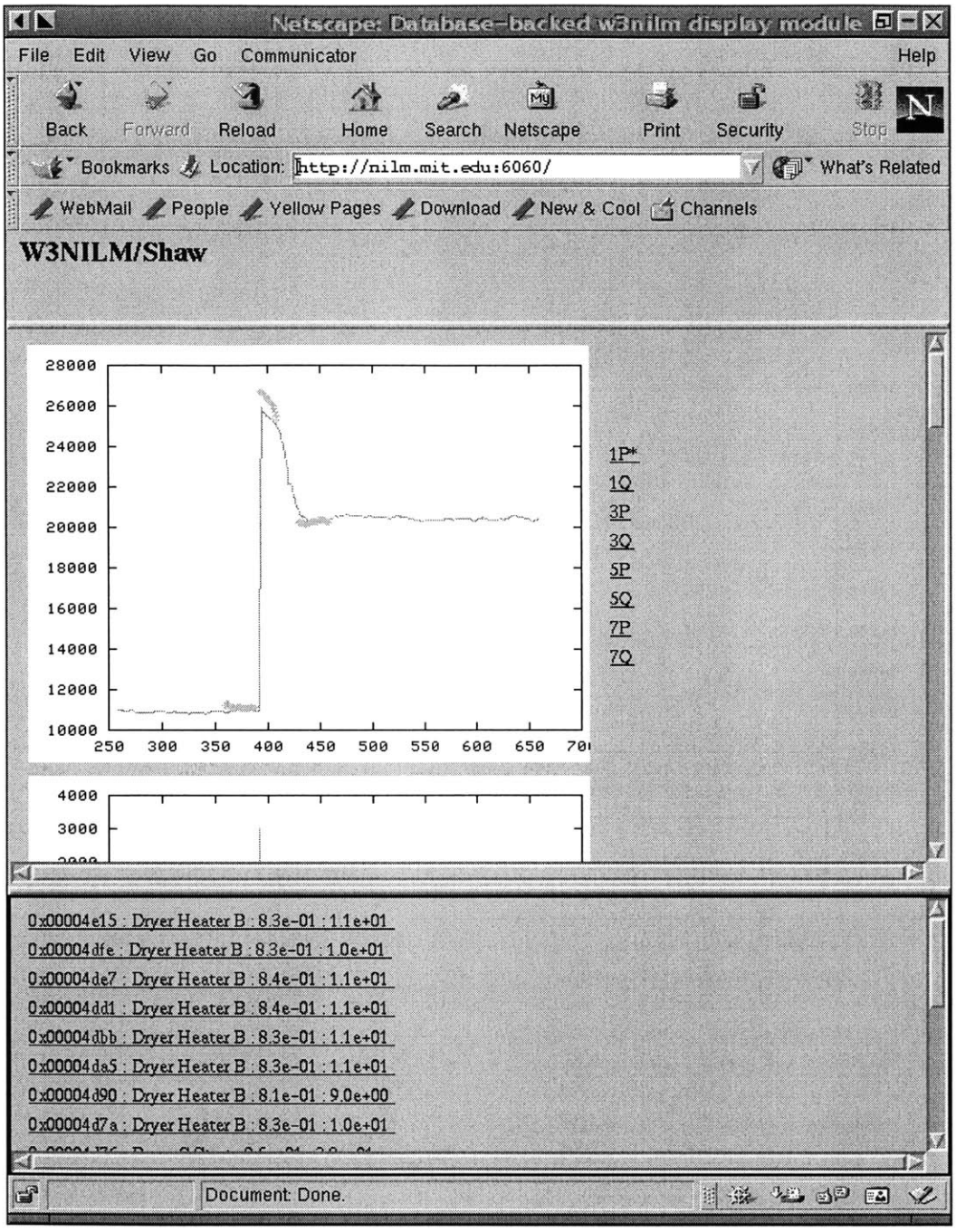


Figure 4-15: Screen served by `w3nilm` showing detection of a dryer. The initial jump is a motor transient combined with the step-like heating element transient. After the dryer spins up, the steady state power is mostly due to the heating element. Solid lines in the graph are spectral envelope data; the overlayed points show the fit of exemplar to transient.

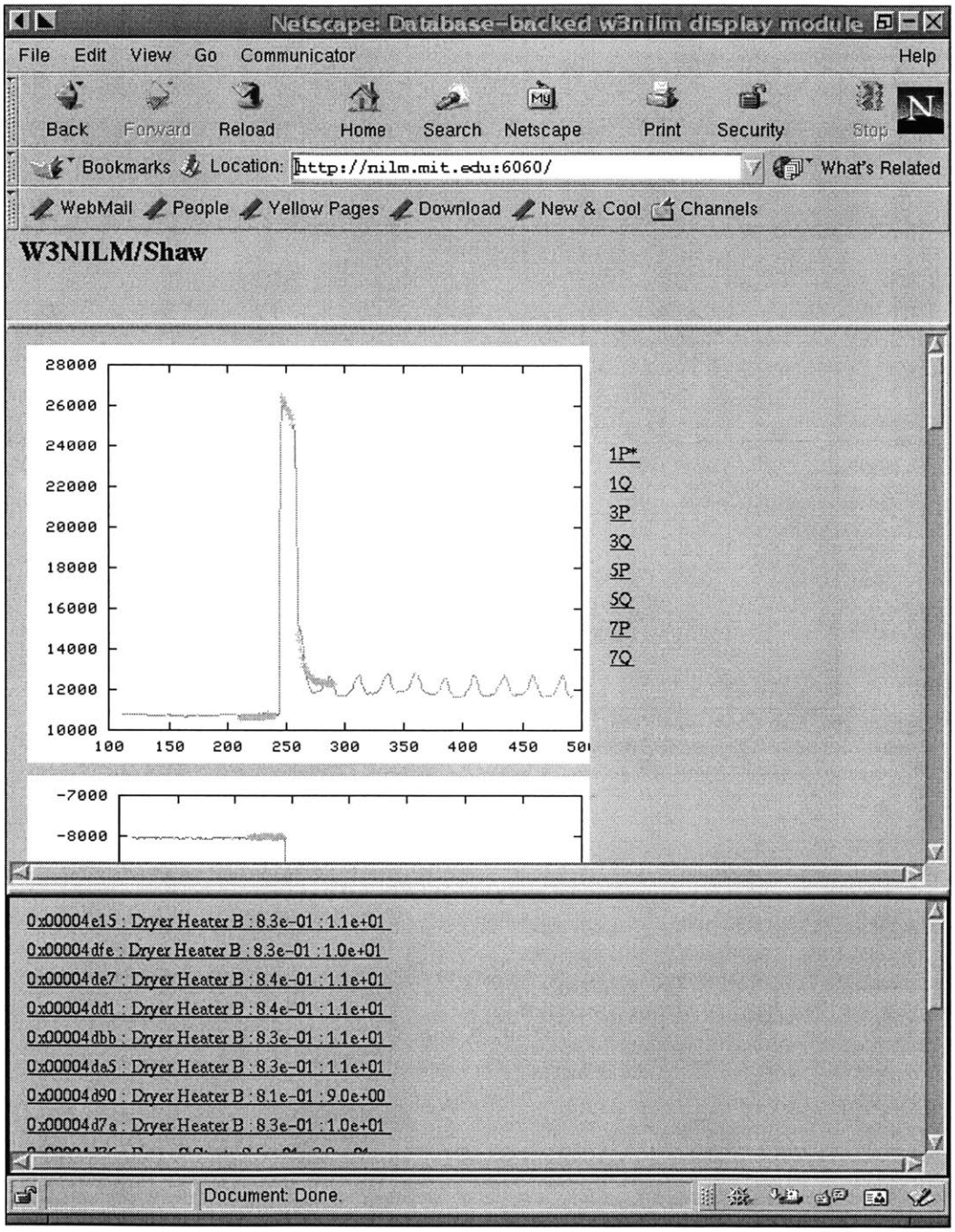


Figure 4-16: A washer transient as detected by NITC and displayed by `w3nilm`. The spectral envelope shows a typical motor transient followed by a cusp-like steady state presumably due to the action of the agitator.

4.2 System identification

Field tests of the system identification methods presented in Chapter 3 were performed using data from the Energy Resource Station (ERS) at the Iowa Energy Center (IEC). Two machines were installed and used to collect data for this thesis and as part of a continuing testing program.

A block diagram of the installation at the IEC/ERS appears in Fig. 4-17. Two machines, Remote 1 and Remote 2, were installed on site. In addition to measuring the total current at the main panel (for Remote 1) and the motor control center (for Remote 2), selected loads associated with air handling unit B and the chiller were individually measured. With the connections shown in Fig. 4-17, Remote 1 measures spectral envelopes for the entire building and Remote 2 measures spectral envelopes for the motor control center. The machines are 200 Mhz Pentium-class machines with 32 Mb memory and roughly 1 Gb hard drives. Both machines are equipped with a network interface card, a Advantech PCL818L data acquisition card, and a preprocessor DSP. The preprocessor DSP, shown in Fig. 4-18, is no longer used – it has been replaced by the preprocessor described in Chapter 2 running on the host machine. The computers run RedHat Linux 4.2 (`http://www.redhat.com`) and are remotely maintained by secure shell over the Internet.

For purposes of this thesis, the machines at IEC/ERS were used to collect submetered data from induction machines associated with "air handling unit B" (AHU-B). With updated software and some hardware reconfiguration, these machines could in principle run NITC.

4.2.1 Iowa Energy Resource Center Pump Diagnostics

Deposits in the pipes of a HVAC heater exchanger are difficult to detect non-invasively and can contribute to decreased heating and cooling efficiency. Using the Iowa Energy Center's Energy Resource Station, electrical transients for a pump in the HVAC system were collected for both normal and obstructed flow condition. Detection of the obstructed flow is demonstrated via parameter estimation using the collected data.

Experimental Setup and data collection

A simplified diagram of the chilled water circulation system is shown in Figure 4-19. For the test, the three-way valve shunting the heat exchanger was positioned so that all liquid flowed through the heat exchanger. The return flow pump, equipped with a variable frequency drive, was operated to control pressure as indicated. The response of this control loop was presumed to be slow enough to ignore in comparison to the start-up transients of the supply pump. To simulate the obstructed flow fault, operators at the IEC/ERS installed a valve in series with the heat exchanger, as indicted. In the no-fault condition, with both return and supply pumps running, flow in the loop was approximately

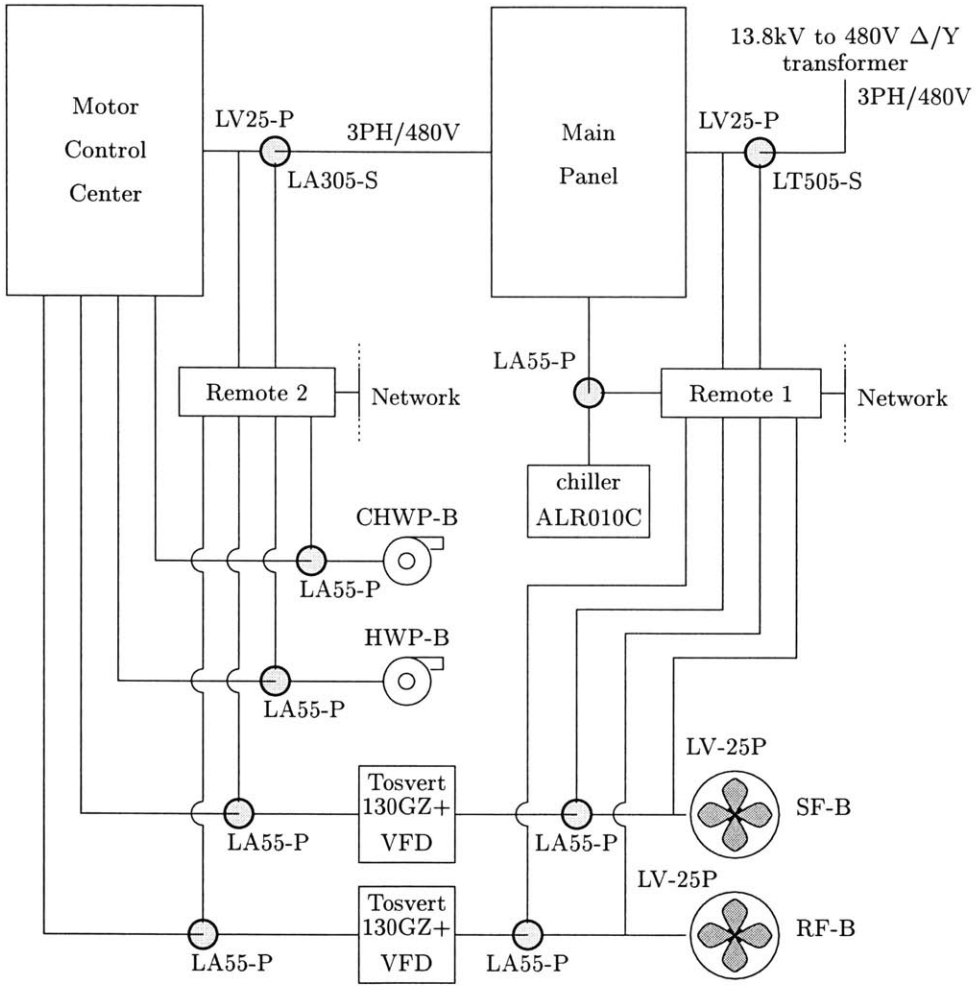


Figure 4-17: Block diagram of installation at the Iowa Energy Research Station. All transducers are labeled with their LEM part number. Remote 1 and Remote 2 are the machines responsible for collecting data.

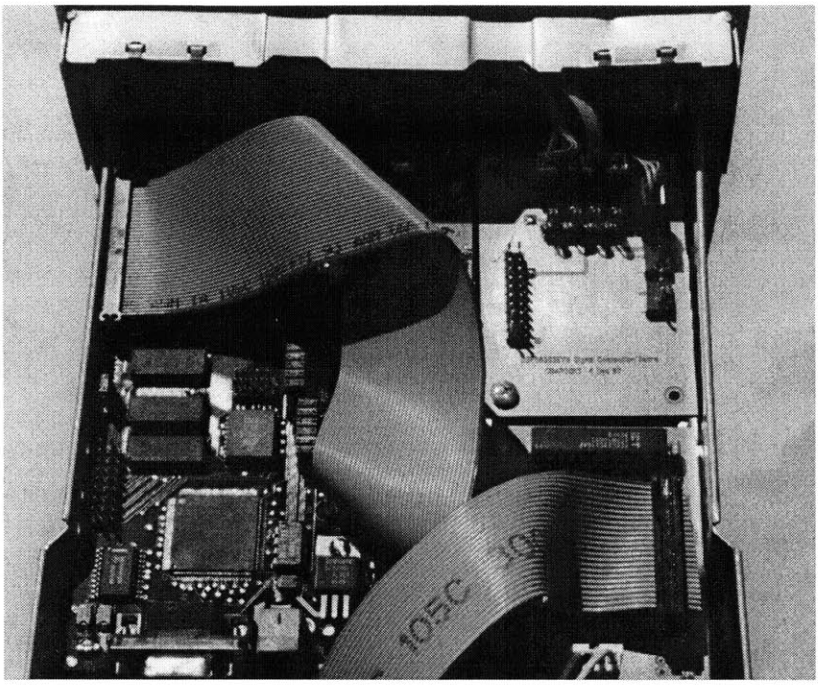


Figure 4-18: DSP preprocessor board as installed in the NILM computers in IEC/ERS. This preprocessor pre-dates the software preprocessor of §2.2.1 and is described in detail in [59].

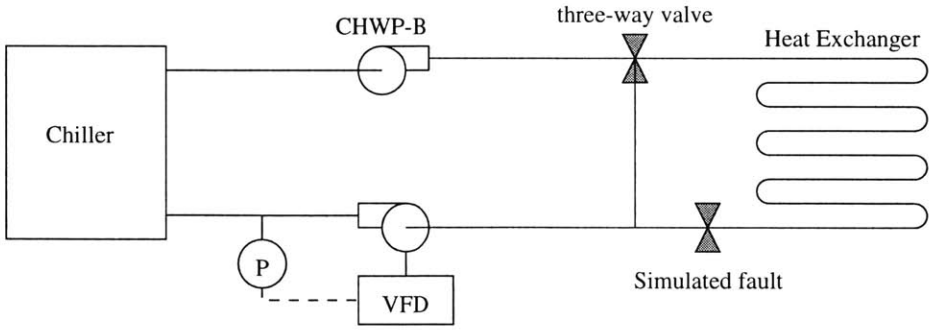


Figure 4-19: Simplified diagram of IEC cooling loop, for AHU-B. Obstructed cooling coil was simulated with a valve. Current to CHWP-B was measured.

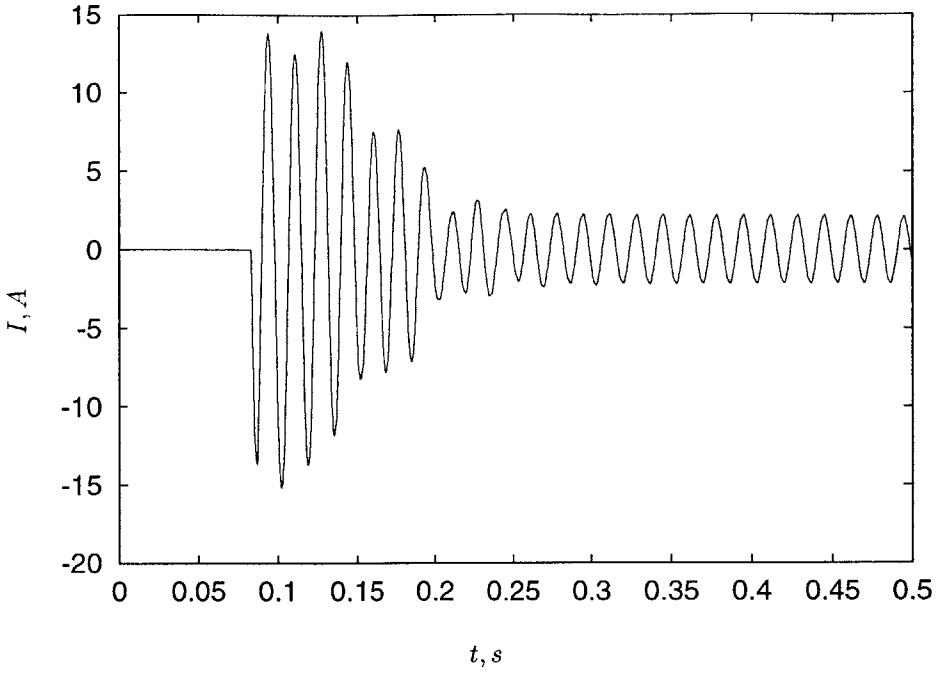


Figure 4-20: Typical electrical transient for CHWP-B motor.

27.1 gpm. With the fault in place, flow was reduced to 11.2 gpm – approximately 40% of nominal flow.

The experimental procedure was to introduce the simulated fault, turn CHWP-B on and off a few times, remove the fault, and again cycle CHWP-B. The resulting start-up transients were recorded and transmitted to the lab for analysis.

Transducers were installed to measure current on two of the phases feeding the balanced, three-phase 480 V pump motor. For unknown reasons one of the transducers failed between installation and the test period. Data from the remaining transducer was sampled with 12 bit resolution at 4000 Hz. A typical transient is shown in Fig. 4-20.

While inspecting the data, it was clear that an unanticipated fault was present. These "glitches," as shown in Fig. 4-21, are probably the result of a bad contactor. Since contactor failure was not part of the model, transients with contactor problems were discarded. The remaining transients were labeled e1, e3, e7 and e8. Transients e1 and e3 were collected with the simulated fault in place.

Model

A simple model for the cooling system pump is an induction machine connected to an inertia with damping. Unfortunately, in addition to the six parameters implied by this simplistic model, it is

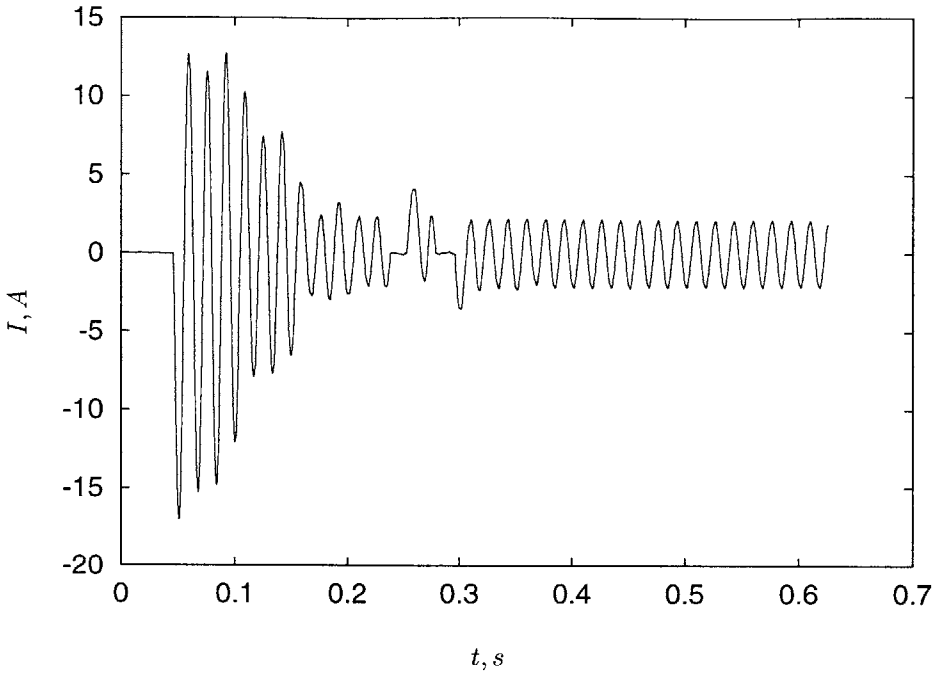


Figure 4-21: Transient demonstrating intermittent contactor failure on CHWP-B.

necessary to estimate the electrical angle when the motor was turned on and the initial speed of the motor. Even when off, the pump motor may be driven by the flow from the other pump in the system. Extracting eight parameters from a simple transient like Fig 4-20 is a questionable proposition.

One solution is to assume that the fault lies in the mechanical system, i.e. that the motor is the same when comparing two transients. A "joint", two-transient model can be formed where identical electrical motor parameters are used for both transients, while the mechanical parameters are allowed to differ for each transient. Mathematically, the model consists of two copies of the usual induction motor electrical equations [58, 37, 20, 73] with the same parameters. Specifically, the model consists of two independent induction motor state vectors, the electrical states for each evolving according to

$$\frac{d}{dt}\begin{pmatrix}\lambda_{qs}\\ \lambda_{ds}\\ \lambda_{qr}\\ \lambda_{dr}\end{pmatrix} = \begin{pmatrix}v_{qs}\\ v_{ds}\\ 0\\ 0\end{pmatrix} - \begin{pmatrix}r_s i_{qs} + \omega_0 \lambda_{ds}\\ r_s i_{ds} - \omega_0 \lambda_{qs}\\ r_r i_{qr} + (\omega_0 - \omega)\lambda_{dr}\\ r_r i_{dr} - (\omega_0 - \omega)\lambda_{qr}\end{pmatrix}, \tag{4.3}$$

where ω_0 is the electrical frequency, ω is the rotor speed, and

$$\begin{aligned}
\lambda_{qs} &= L_l i_{qs} + (L_l + L_m)(i_{qs} + i_{qr}) \\
\lambda_{ds} &= L_l i_{ds} + (L_l + L_m)(i_{ds} + i_{dr}) \\
\lambda_{qr} &= L_l i_{qr} + (L_l + L_m)(i_{qs} + i_{qr}) \\
\lambda_{dr} &= L_l i_{dr} + (L_l + L_m)(i_{dd} + i_{dr}).
\end{aligned}$$

Although the two state vectors are governed by the same electrical parameters, parameters of the mechanical interactions differ. The mechanical interactions are

$$\frac{d}{dt}\omega = 3K_{1,2}(\tau - \beta_{1,2}\omega), \tag{4.4}$$

where τ is proportional to the torque of electrical origin for the appropriate system, i.e. $\tau = \lambda_{qr} i_{dr} - \lambda_{dr} i_{qr}$. The model is compared to the measurements in the lab frame, using the output equation

$$i = i_{ds}\cos(\omega_0 t + \phi) - i_{qs}\sin(\omega_0 t + \phi) \tag{4.5}$$

for each system. The parameter ϕ in the output equation is the phase in the line cycle when the induction machine was switched on. The parameter vector of the joint model is then

$$\mu = (\, r_r \quad r_s \quad L_m \quad L_l \quad K_1 \quad \beta_1 \quad \omega(0)_1 \quad \phi_1 \quad K_2 \quad \beta_2 \quad \omega(0)_2 \quad \phi_2 \,)', \tag{4.6}$$

and the state vector x is

$$x = (\,(\,\lambda_{qs} \quad \lambda_{ds} \quad \lambda_{qr} \quad \lambda_{dr} \quad \omega\,)_1\,(\,\lambda_{qs} \quad \lambda_{ds} \quad \lambda_{qr} \quad \lambda_{dr} \quad \omega\,)_2\,). \tag{4.7}$$

The joint model may be identified by applying the techniques of Chapter 3 to the residual

$$r = \begin{pmatrix}
i_{1as}[1] - \hat{i}_{1as}[1] \\
i_{2as}[1] - \hat{i}_{2as}[1] \\
i_{1as}[2] - \hat{i}_{1as}[2] \\
i_{2as}[2] - \hat{i}_{2as}[2] \\
\vdots \\
i_{1as}[N] - \hat{i}_{1as}[N] \\
i_{2as}[N] - \hat{i}_{2as}[N]
\end{pmatrix} \tag{4.8}$$

where $i_{1as}[k]$ is the k'th sample of the measured current from the first dataset, $i_{2as}[k]$ is the k'th sample from the second dataset, and the $\hat{i}_{1as}$ is the modeled current corresponding to the first

Dataset	Transient 1	Transient 2	Type
ds0	e1	e7	fault / no-fault
ds1	e3	e8	fault / no-fault
ds2	e3	e7	fault / no-fault
ds3	e1	e8	fault / no-fault
ds4	e1	e3	fault / fault
ds5	e7	e8	no-fault / no-fault

Table 4.1: Dataset organization for IEC/ERS pump tests.

	Electrical Parameters			
Dataset	r_r	r_s	L_m	L_l
ds0	1.548e+01	1.424e+01	7.266e-01	3.046e-02
ds1	1.487e+01	1.418e+01	7.050e-01	3.096e-02
ds2	1.529e+01	1.421e+01	7.205e-01	3.061e-02
ds3	1.511e+01	1.418e+01	7.132e-01	3.077e-02
ds4	1.545e+01	1.380e+01	7.324e-01	3.072e-02
ds5	1.499e+01	1.462e+01	7.048e-01	3.046e-02

Table 4.2: Electrical parameter estimates by dataset for IEC/ERS pump tests.

dataset. The residual entries are interleaved by dataset so that the elements are sorted by time, as assumed by the techniques in Chapter 3. As implemented in the `id` code in §C.2 used in this problem, a regularization term is also effectively added to r.

Datasets for the joint model were organized from the available transients as shown in Table 4.1. Identifying different combinations of transients provides a useful cross check of the consistency of the estimates.

Results

The model fit the data well, as show in Fig. 4-22. The residuals are small, but not structureless. The model is simple and does not include complicated modeling of the interaction between water and pump – the structure in the residuals *might* support a more complicated model with the data available. On the other hand, a more complicated model might cause problems with other data sets.

Parameters for the six datasets are shown in Table 4.2 and Table 4.3. The datasets are separated by type, as given in Table 4.1. The parameters in Tables 4.3 and 4.2 agree very nicely. In particular, the individual transients produce remarkably consistent estimates independent of the transients that they are paired with when fitting a joint model.

The failure of the residuals to confirm the statistical prerequisites for maximum-likelihood estimation is not necessarily important. For example, it may be that the parameters are invariant under

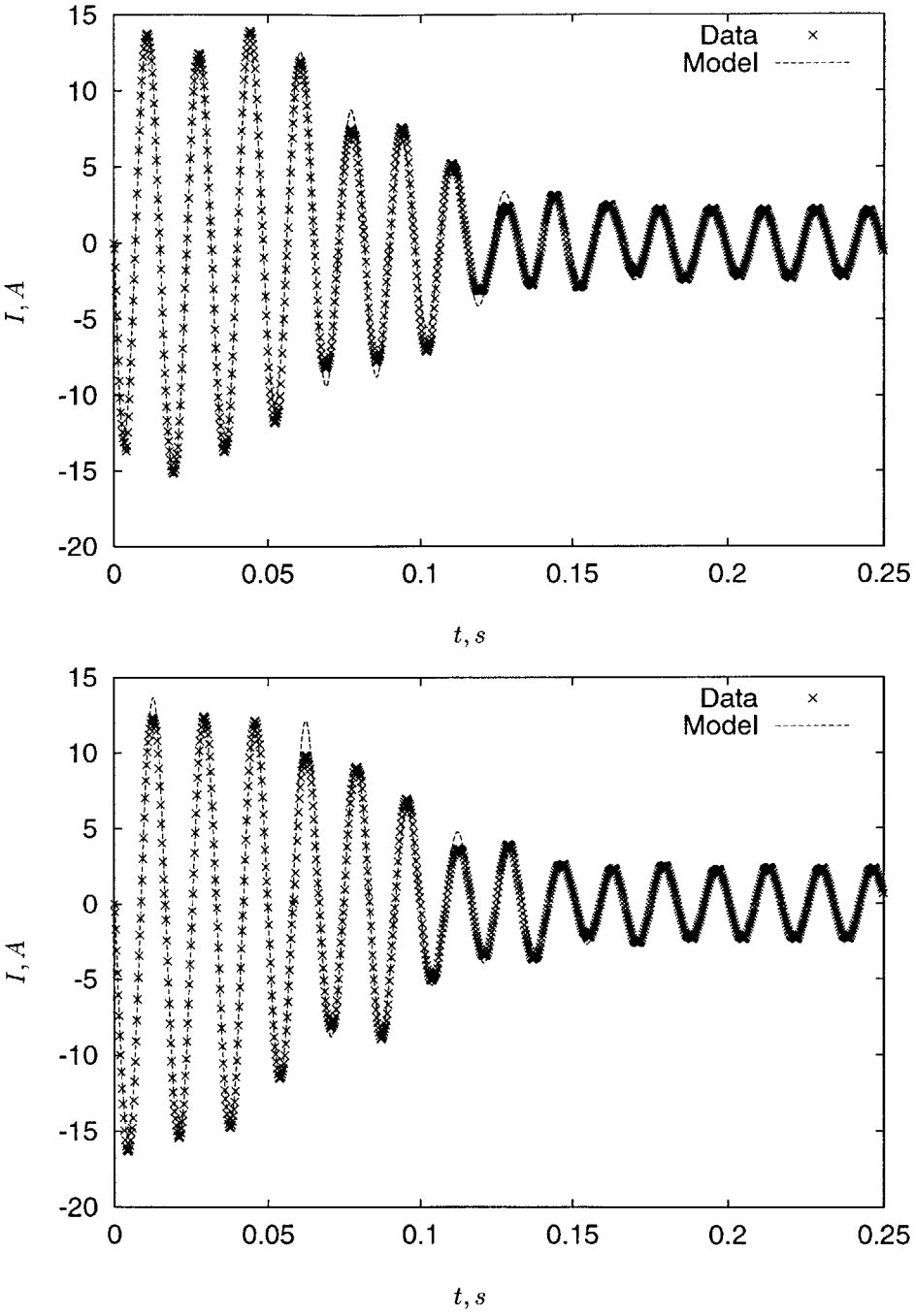


Figure 4-22: Comparison of model to experimental data for data set ds0. Top is with fault, bottom is without fault. The quality of fit is typical of the other datasets.

Dataset	Mechanical Parameters			
	K_1	β_1	K_2	β_2
ds0	1.435e+02	1.544e-02	1.531e+02	1.830e-02
ds1	1.415e+02	1.498e-02	1.516e+02	1.771e-02
ds2	1.427e+02	1.515e-02	1.531e+02	1.828e-02
ds3	1.422e+02	1.534e-02	1.524e+02	1.783e-02
ds4	1.451e+02	1.566e-02	1.454e+02	1.540e-02
ds5	1.496e+02	1.803e-02	1.487e+02	1.763e-02

Table 4.3: Mechanical parameter estimates by dataset for IEC/ERS pump tests.

a perturbation of the observations that would make the residual satisfy maximum-likelihood requirements. The key issue, both from the point of view of parameter quality and diagnostic concerns, is the robustness of the parameters under perturbations of the input. This issue can be addressed by making parameter distribution estimates using synthetic datasets, as discussed in §3.5. Figure 4-23 shows parameter distribution estimates for 4000 synthetic datasets combining dataset ds0 with a $\mathcal{N}(0, .1)$ white disturbance. The non-Gaussian distribution estimates confirm that classical analysis at the equilibrium would be misleading, and the estimates show encouraging robustness. More interesting, however, is the robustness of the mechanical parameters presumed to indicate the fault. Any diagnostic decision would be highly error prone if the distributions of the parameters under a realistic disturbance model overlap significantly. Estimates of the distributions of the mechanical parameters under fault and no-fault conditions with dataset ds0 appear in Fig. 4-24. Figure 4-24 suggests that detection of the fault would be successful with the disturbance used to create the synthetic datasets.

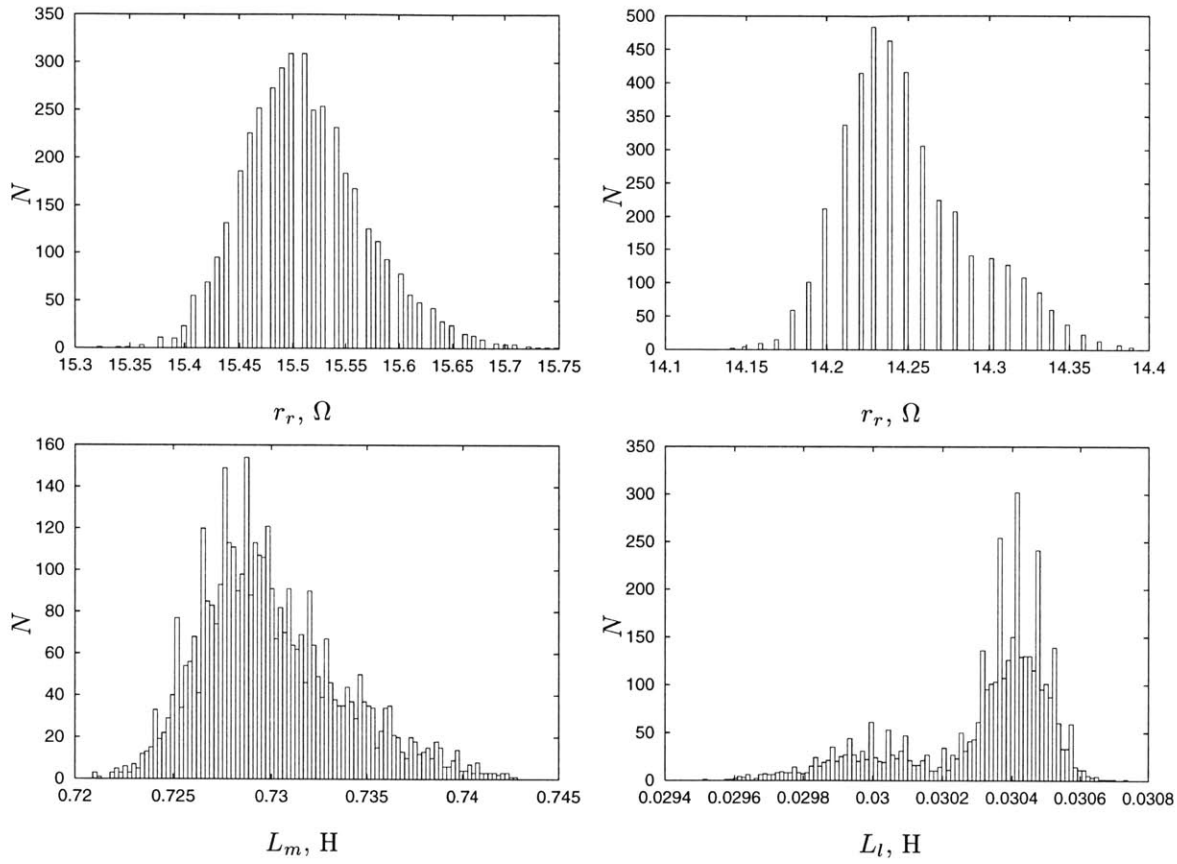


Figure 4-23: Histograms for electrical parameters r_r, r_s, L_m, and L_l under $\mathcal{N}(0, .1)$ white disturbances. Tight distributions for these parameters are not essential for fault detection, but suggest that the model is reasonable.

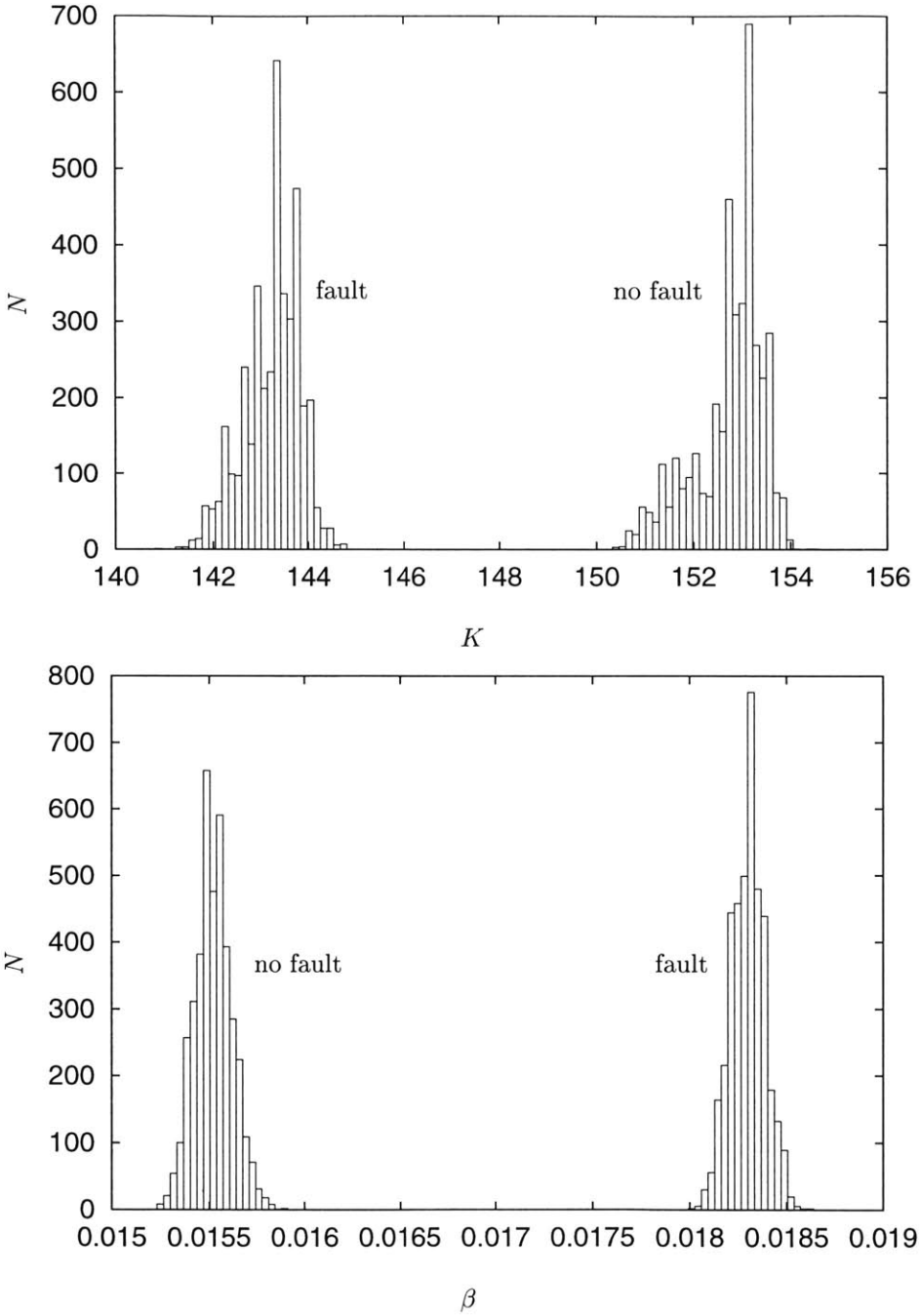


Figure 4-24: Distribution estimates for mechanical parameters β and K. The distributions of these parameter estimates strongly suggest that the fault can be detected with small probability of error under the presumed disturbance.

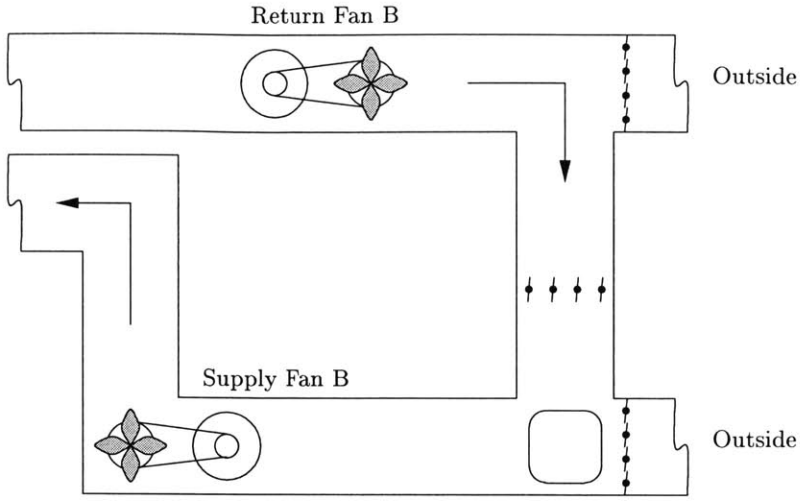


Figure 4-25: Schematic diagram of IEC/ERS air handling unit B. Arrows indicate direction of air flow. Valves on the right allow building air to be recirculated or mixed to varying degrees with outside air. Supply Fan B was used in the tests.

4.2.2 Iowa Energy Resource Center Fan Diagnostics

Fans and ducts that move air through a building are interesting targets for nonintrusive diagnostics. Modern ductwork includes features such as dampers that mix inside and outside air, depending on control inputs. Failure of these systems to behave as expected could lead to inefficiency or failure to regulate conditions inside the building.

To investigate diagnostic possibilities in ventilation systems, data were collected from one of the air-handling units at the IEC/ERS. A diagram of the air handling unit appears in Fig. 4-25. Transients from the motor connected to supply fan B were measured, and two kinds of faults were considered. Unlike the pump situation, two functioning current transducers were available for measurement.

First, the mixing box door (indicated on lower right of Fig. 4-25) was opened to create a gross change in the flow characteristics of the system. The original intent of the test was to change the configuration of the dampers controlling the extent of recirculation. For example, it would be especially interesting to detect if commands from a building energy management system to the recirculation dampers were foiled due to mechanical problems with the damper. Unfortunately, in this particular installation, arbitrarily changing the position of the dampers to simulate the fault was not possible without serious modifications to the damper control. In particular, the dampers controlling outside air are interlocked with the condition of the supply fan motor. The dampers start closed and open to the control position some time after the fan starts. As a substitute, the mixing box door was opened to simulate the magnitude of air-flow change that might result from a

stuck damper.

The second diagnostic concern in the air handling unit was the effect of a slipping fan belt. This fault was introduced by loosening the belt connecting the fan to the motor.

Model

Deciding what aspects of a complicated situation like Fig. 4-25 to include in an identification model is a challenging task. Many aspects of the system are under active control, or are subject to human intervention. For example, the load seen by the air handling unit changes when someone opens a door in a room served by the unit.

As a first assumption, controls (e.g. for the return fan) were assumed to operate on a time scale much slower than the induction motor startup transient. Also, other aspects of the system (for example, state of doors and windows in rooms served by the air handling unit) were assumed to remain constant during the test. The joint modeling technique used in the pump diagnostics was also used, i.e. the induction motor was assumed to be a constant for purposes of comparing two situations. As a starting point, the mechanical situation was modeled as in §4.2.1, i.e.

$$\frac{d}{dt}\omega = 3K_{1,2}(\tau_{1,2} - \beta_{1,2}\omega). \tag{4.9}$$

Unlike the pump diagnostics, two phases of current measurements were available for fan transients. These measurements were transformed to the synchronous dq frame [37] for identification.

Results

Parameter estimates for the IEC/ERS fan tests are given in Tables 4.4 and 4.5. Dataset A refers to the no-fault situation, B corresponds to the open mixing box door, C is the slipping belt, and D is both slipping belt and open door.

The electrical parameters of Table 4.4 should be essentially the same for all combinations of datasets, since the same motor was used throughout. There is quite close agreement for parameters r_s and L_l and not such good agreement for the magnetizing inductance L_m and rotor resistance r_r. It is possible that the rotor resistance r_r might vary due to thermal effects, but the variation seen in the magnetizing inductance L_m is extreme. It seems likely that r_r and L_m are being influenced by the changes introduced in the mechanical part of the system. The most suspect combination in Table 4.4 is the first row.

The mechanical parameters for the six combinations of datasets are given in Table 4.5. In this table, parameters with subscript i correspond to the dataset appearing in the i'th column under the heading "Datasets." For example, β_2 in row three corresponds to an estimate for dataset D

Datasets		Electrical Parameters			
1	2	r_r	r_s	L_m	L_l
A	B	2.35e-01	8.07e-01	5.84e-02	4.62e-03
A	C	3.76e-01	8.15e-01	9.72e-02	5.13e-03
A	D	3.99e-01	7.95e-01	9.14e-02	4.99e-03
B	D	2.89e-01	7.98e-01	1.17e-01	4.97e-03
C	B	3.02e-01	8.06e-01	9.79e-02	4.58e-03
C	D	3.26e-01	8.03e-01	1.52e-01	4.78e-03

Table 4.4: Electrical parameter estimates for IEC/ERS fan tests. A is no-fault, B is open mixing box door, C is slipping belt, D is slipping belt and open mixing box door.

(open door, loose belt) in conjunction with dataset A (no-fault) and β_1 in row three corresponds to an estimate for dataset A in conjunction with dataset D. Because the datasets appear in different combinations in Table 4.5, a pseudo cross-validation check can be performed. Mechanical parameters corresponding to an individual dataset (A,B,C or D) should be roughly the same. Figure 4-26 aids in this comparison by plotting the parameters corresponding to different mechanical situations. Note that the "non-fault" points in Fig. 4-26 are quite distinct from the "fault" points. Also, the outlier in Fig. 4-26 corresponds to the suspect row one of Table 4.4.

Parameters in Fig. 4-26 and Table 4.5 make good physical sense. The situations where the fan belt is slipping show the lowest friction coefficient. As shown in Fig. 4-26, the slipping fan belt situation is relatively close to the slipping fan belt and open door scenario. This makes sense – if the motor is not coupled to the fan, the duct configuration is irrelevant. Physical interpretation of K in Fig. 4-26 is a bit more involved. Since K is proportional to $\frac{1}{J}$, larger values of K mean *less* inertia. The open door scenario has the most coupled inertia, followed by both slipping fan belt cases, and the no-fault situation has the least inertia. Since the slipping fan belt essentially uncouples the motor (at least for the startup transient), the motor should see only the inertia of the rotor and pulley. This hypothesis agrees well with the close inertia values obtained for both slipping belt situations. In the no-fault scenario, the inertia values are *lower* that the uncoupled case. This makes sense because the no-fault case is the only case where the return fan is supplying a force that tends to accelerate the rotor. The back pressure should not translate exactly into a change in inertia, but its gross effect may, particularly in the absence of a more sophisticated model. When the mixing box door is open, the coupling of the supply fan to the return fan is greatly lessened, and the effective inertia includes the air column coupled to the fan.

The fit of the model to the observations, shown in Fig. 4-27 through Fig. 4-32, is fairly good. Note that these graphs show data which have been dq transformed. See [37] for details. Other mechanical models were attempted, including adding a $\gamma\omega^2$ to (4.9). These models did not offer much improvement in the fit, and in some cases it was clear that the data would not support a more

Datasets		Mechanical Parameters			
1	2	K_1	β_1	K_2	β_2
A	B	3.94e+02	8.72e-02	7.51e+01	7.16e-02
A	C	2.46e+02	1.41e-01	1.37e+02	3.40e-02
A	D	1.92e+02	1.27e-01	1.28e+02	2.93e-02
B	D	5.25e+02	7.38e-02	1.62e+02	3.73e-02
C	B	1.53e+02	6.26e-02	4.89e+01	6.76e-02
C	D	1.43e+02	5.46e-02	1.43e+02	3.67e-02

Table 4.5: Mechanical parameter estimates for IEC/ERS fan tests. A is no-fault, B is open mixing box door, C is slipping belt, D is slipping belt and open mixing box door.

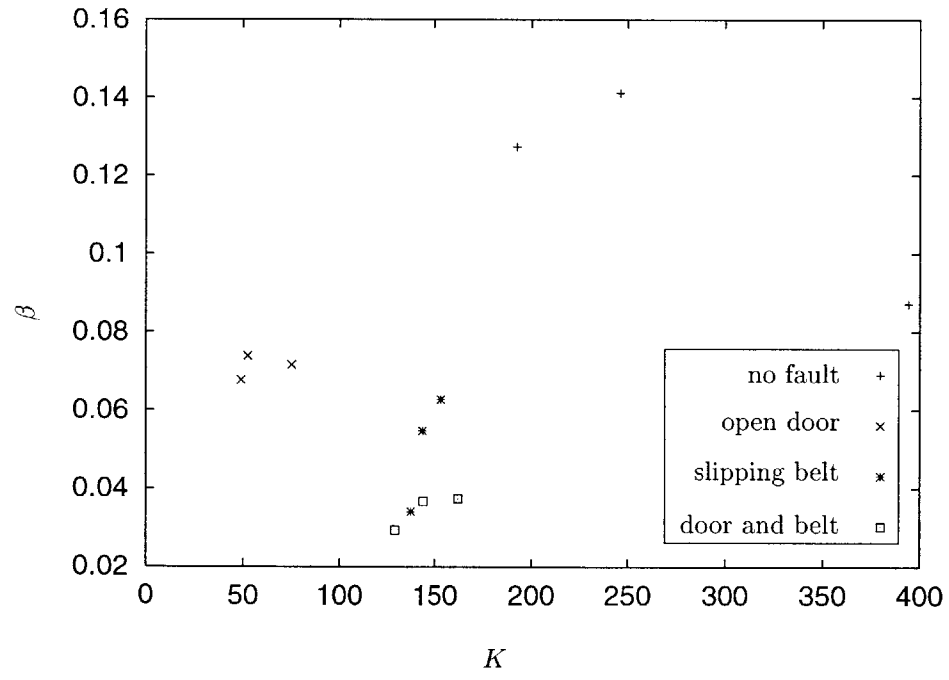
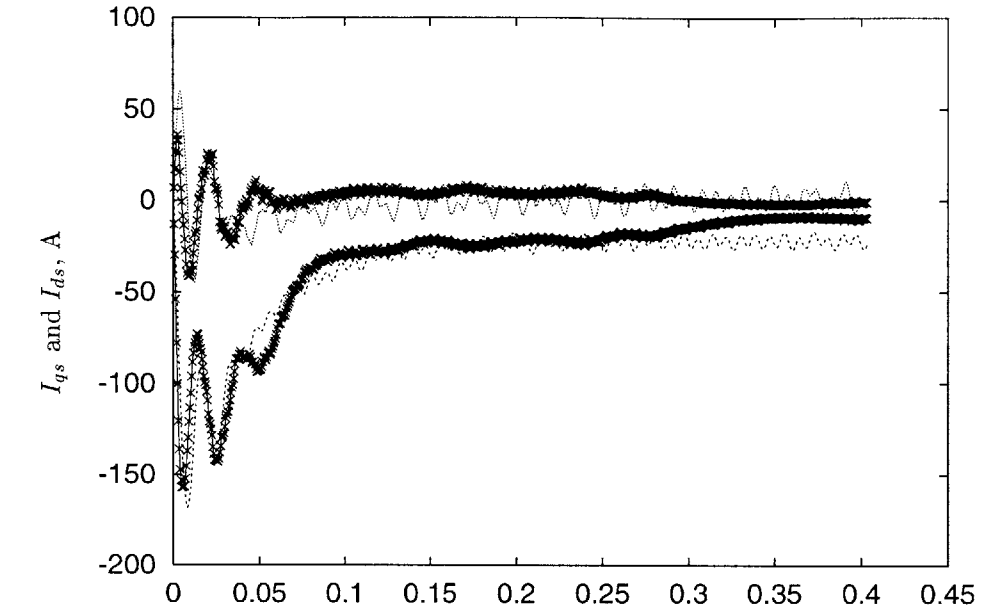
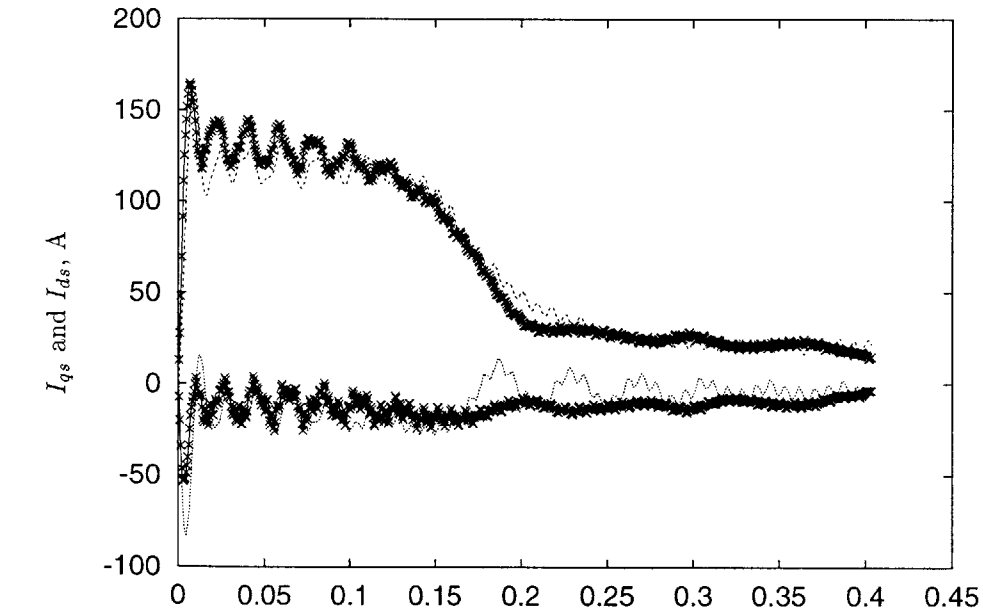


Figure 4-26: Scatter plot showing parameters for each dataset when estimated in conjunction with the other datasets. The nine points in the lower left corner are fault points. Note that the rightmost point corresponds to the combination in the first row of Table 4.4, which is suspect.

complicated model.

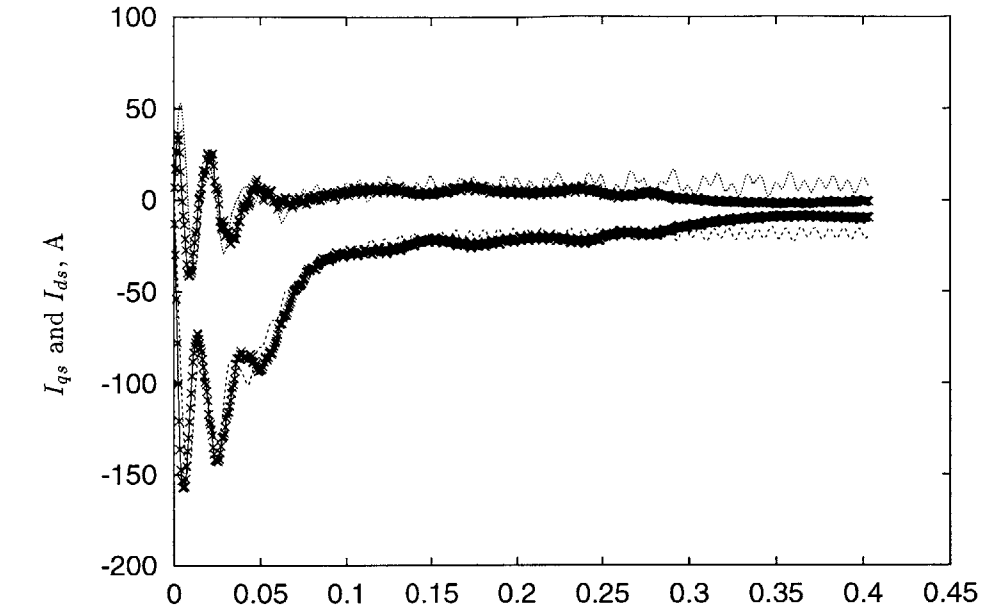


a.

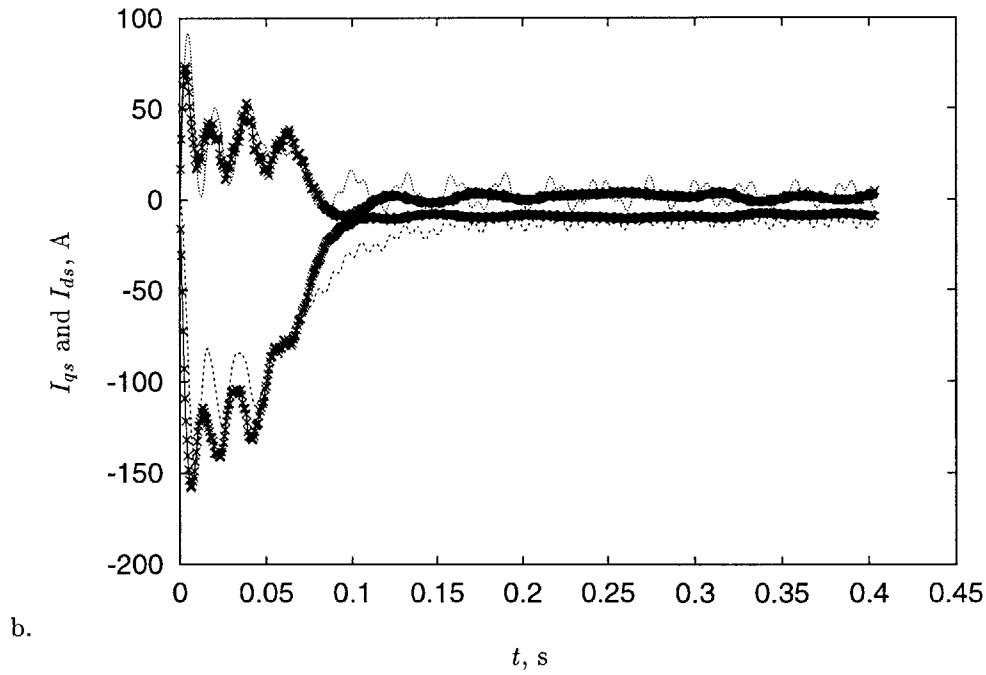


b.

Figure 4-27: Fit of joint model to experimental data sets for tight belt and 100% recirculation (a) and tight belt and open mixing box door (b).

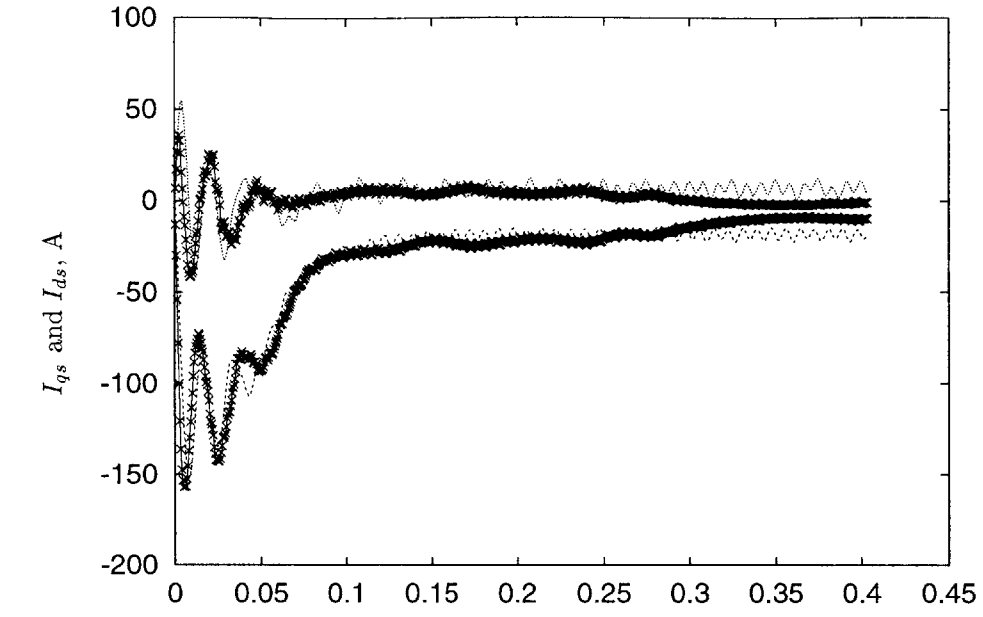


a.

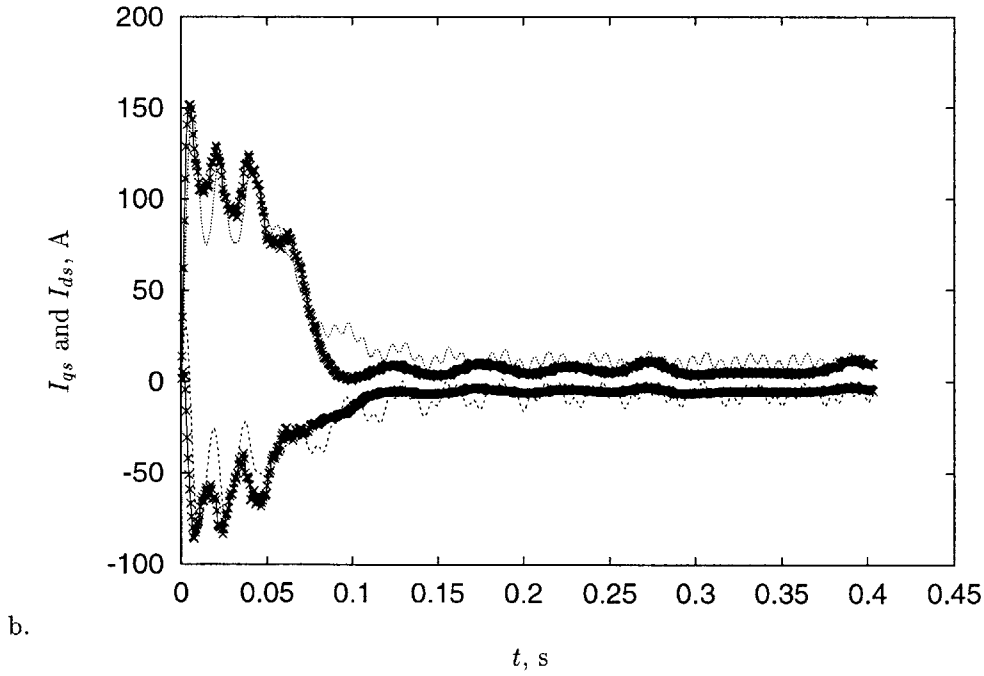


b.

Figure 4-28: Fit of joint model to experimental data sets for tight belt and 100% recirculation (a) and loose belt and 100% recirculation (b).

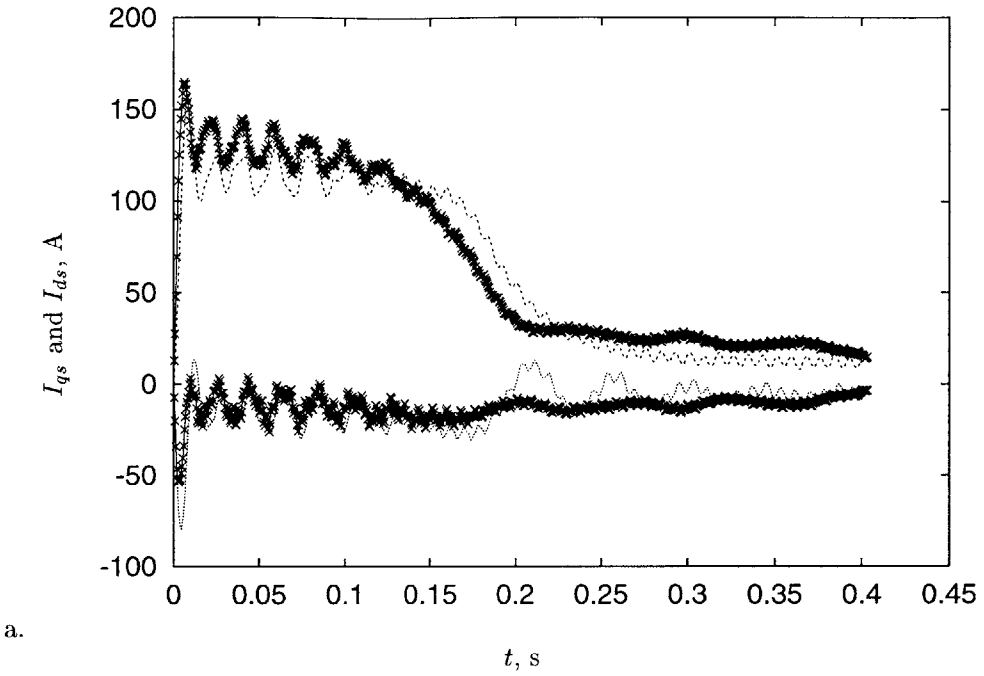


a.

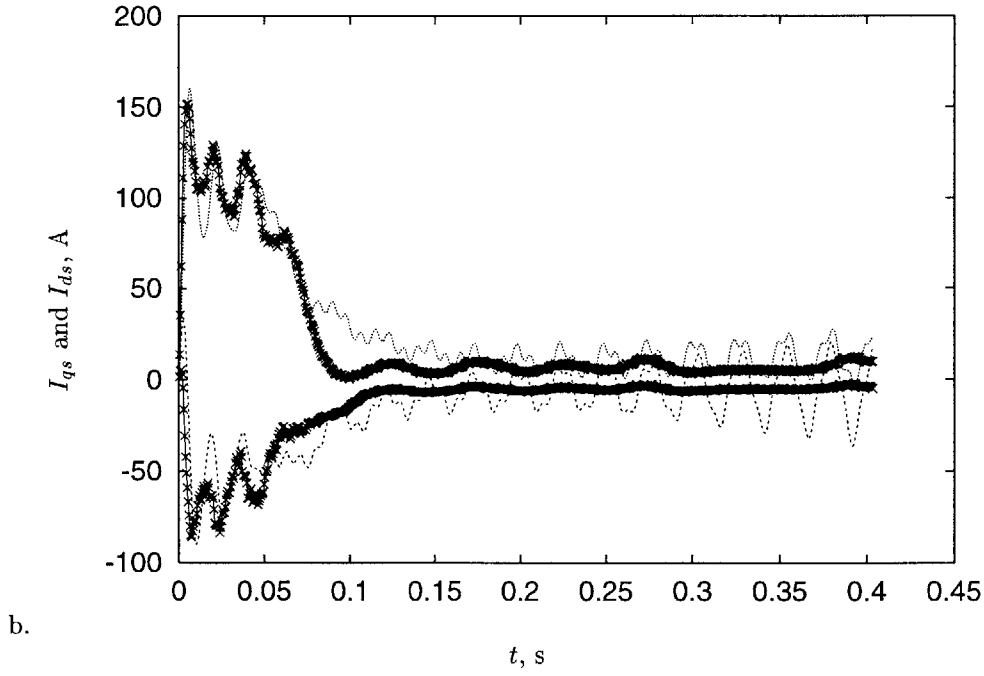


b.

Figure 4-29: Fit of joint model to experimental data sets for tight belt and 100% recirculation (a) and loose belt and open mixing box door (b).

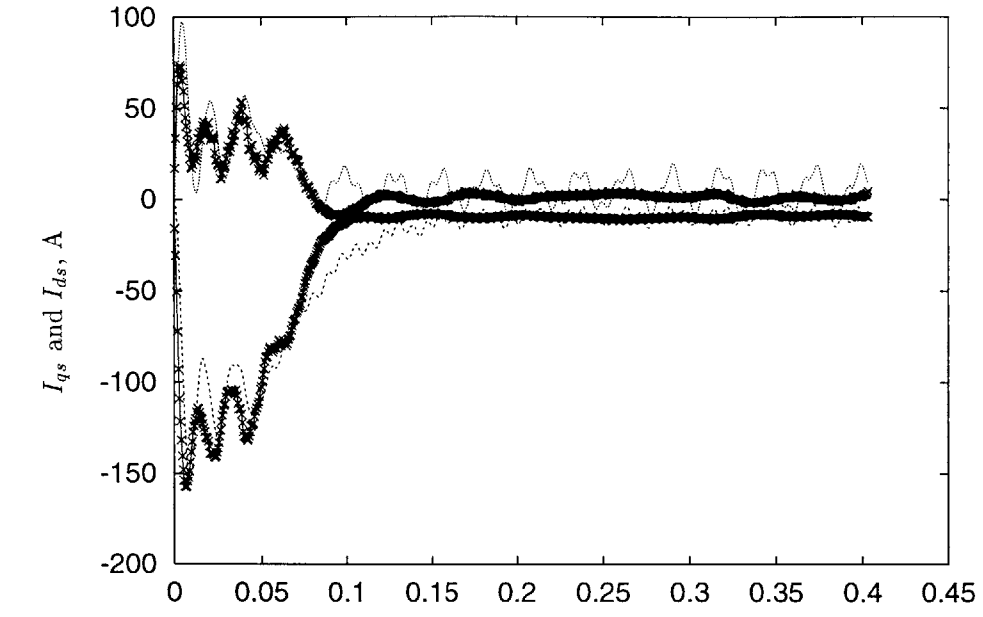


a.

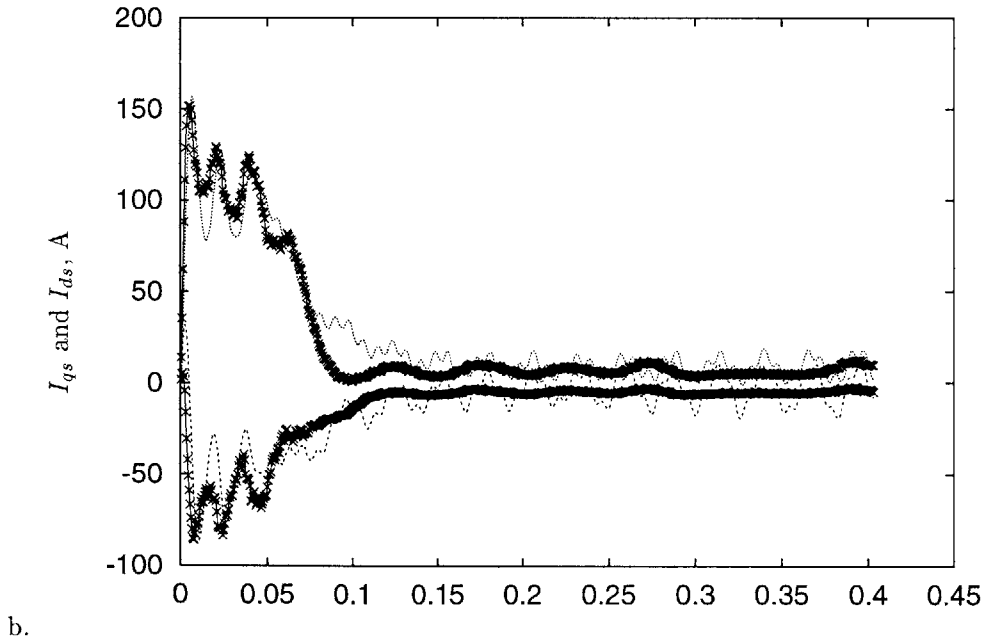


b.

Figure 4-30: Fit of joint model to experimental data sets for tight belt and open mixing box door (a) and loose belt and open mixing box door (b).

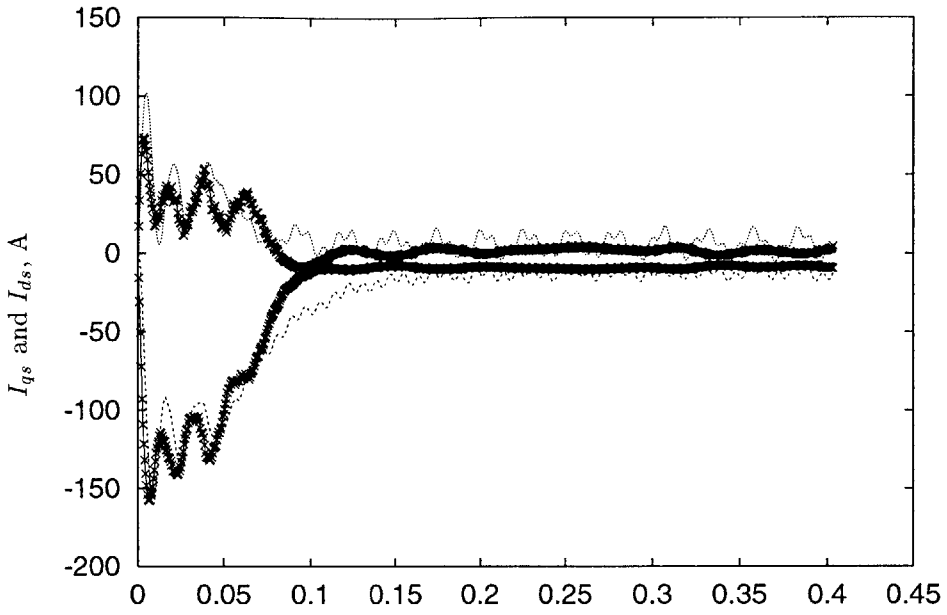


a.

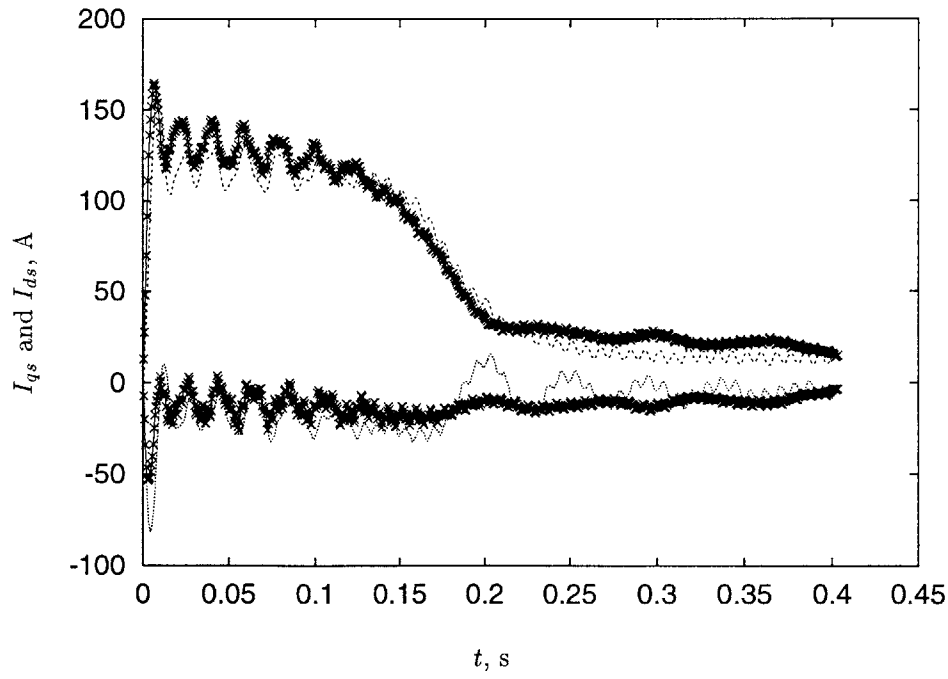


b.

Figure 4-31: Fit of joint model to experimental data sets for loose belt and 100% recirculation (a) and loose belt and open mixing box door (b).



a.



b.

Figure 4-32: Fit of joint model to experimental data sets for loose belt and 100% recirculation (a) and tight belt and open mixing box door (b).

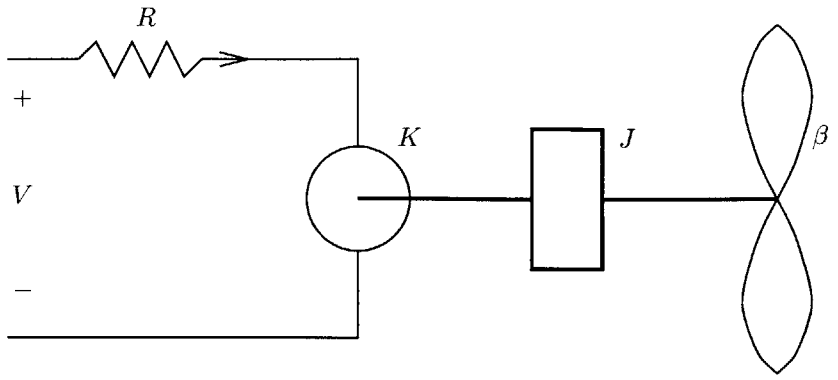


Figure 4-33: Model of automobile ventilation system.

4.3 Nonintrusive classification and identification

The combination of nonintrusive classification from Chapter 2 and system identification techniques of Chapter 3 was tested in two situations. In a DC automotive environment, fan transients were classified and dispatched for identification. In the mock building, combined classification and identification was tested using induction motor transients.

4.3.1 Fan load in an automobile

The measurements and instrumentation described in §4.1.1 were used to test combined classification and identification in the test vehicle.

Model

The ducted fan ventilation system in the test automobile can be modeled as shown in Fig. 4-33. There is a series resistance R, which is the combined effect of the motor and wire resistance and the ballast resistor used to set the fan speed. Also included in the model are the inertia J, motor constant K, and a drag coefficient β. The mechanical equation of motion is

$$J\frac{d\omega}{dt} = T_e - \beta\omega^2 \tag{4.10}$$

where T_e is the torque of electric origin, i.e.

$$T_e = KI. \tag{4.11}$$

Noting that

$$I = \frac{V - K\omega}{R}, \tag{4.12}$$

(4.10) may be rewritten

$$J\frac{d\omega}{dt} = K\frac{V - K\omega}{R} - \beta\omega^2. \tag{4.13}$$

This system apparently has four parameters, but under the substitution $x = \frac{JR}{K}\omega$, it can be rewritten

$$\frac{dx}{dt} = V - \alpha_1 x - \alpha_2 x^2 \tag{4.14}$$

with observations

$$I = \alpha_3(V - \alpha_1 x). \tag{4.15}$$

In terms of the original parameters,

$$\alpha_1 = \frac{K^2}{RJ} \tag{4.16}$$

$$\alpha_2 = \frac{\beta K}{J^2 R} \tag{4.17}$$

$$\alpha_3 = \frac{1}{R}. \tag{4.18}$$

The model may be identified using the techniques of Chapter 3 with the residual

$$r = I - \hat{I}(\mu) \tag{4.19}$$

where I is the measured current and $\hat{I}(\mu)$ is the modeled current obtained from (4.15). Note that this model cannot be identified using ordinary least squares techniques unless measurements of the state ω are available.

Results

Using the techniques presented in §3.3, the model developed above, and data dynamically dispatched from NITC, parameters were obtained for four configurations of the ventilation system. Figure 4-35 shows the typical quality of fit between the system model and the measured data. Parameters are given in Table 4.6.

The parameters in Table 4.6 make good physical sense. Switching from vent to heat involves changing the air flow from short pipes to longer pipes. In terms of the natural parameters of the system, R and K might be expected to stay the same; the fan motor is unchanged, and the fan was run at the same set speed. In terms of the identified parameters, α_3 should stay the same. However, the air mass coupled to the fan should increase as the duct length increases, so the lumped inertia

Scenario	Parameters		
	α_1	α_2	α_3
Fresh/Vent	.20	.094	.46
Fresh/Heat	.12	.021	.44
Recirc/Vent	.15	.077	.44
Recirc/Heat	.11	.022	.44

Table 4.6: Comparison of parameter estimates for automobile ventilation system with selector on "Vent" or "Heat".

J should increase. Larger J corresponds to smaller α_1.

Based on this physical reasoning, α_1 in Table 4.6 should be smaller for the "heat" settings than the "fresh" settings. Also, the estimates of α_3 in Table 4.6 should be roughly the same.

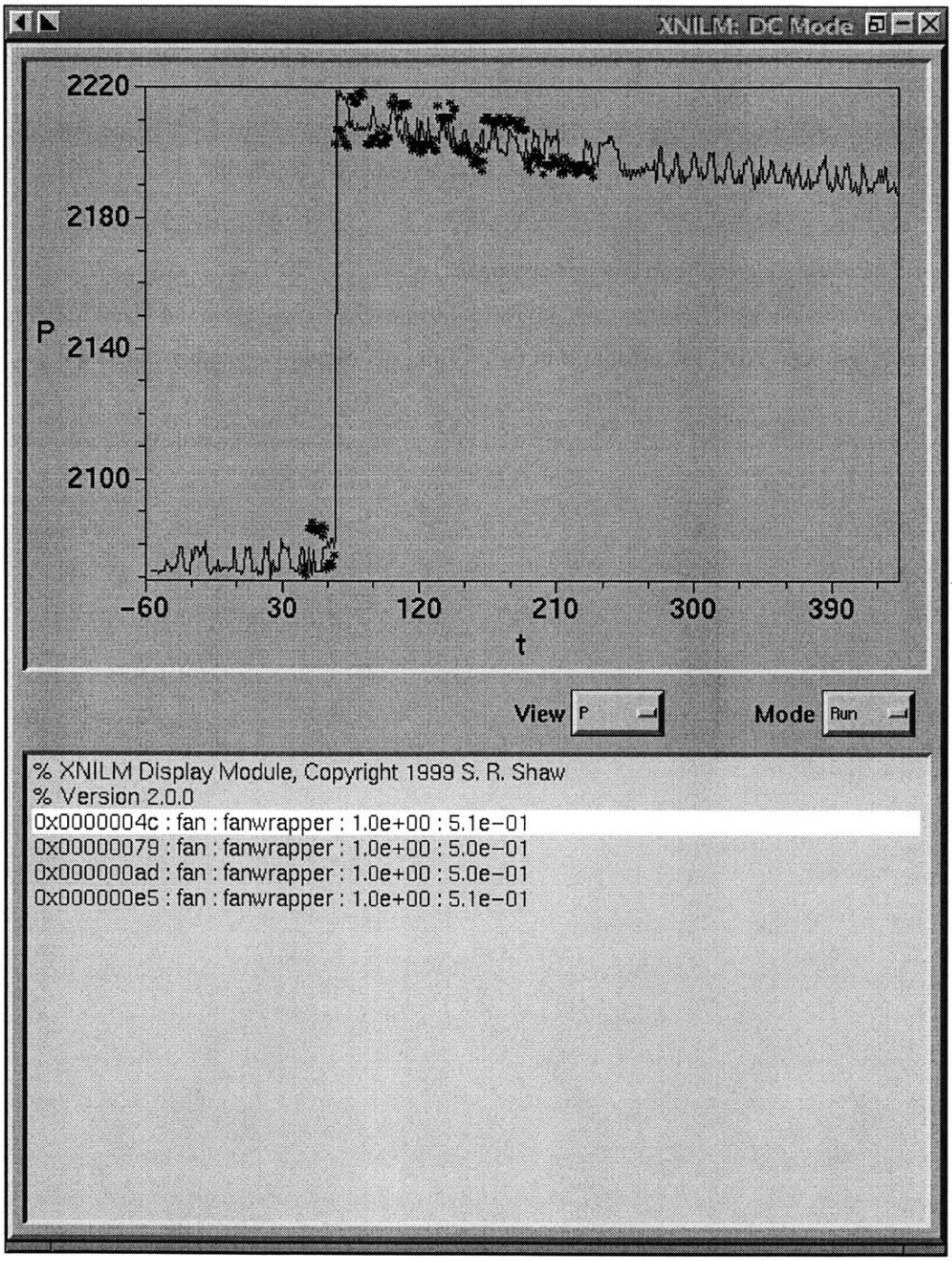


Figure 4-34: Screen shot of `xnilm` showing detection of ventilation fan transient in the automobile. Solid lines in the graph are measured data; the overlayed points show the fit of exemplar to transient.

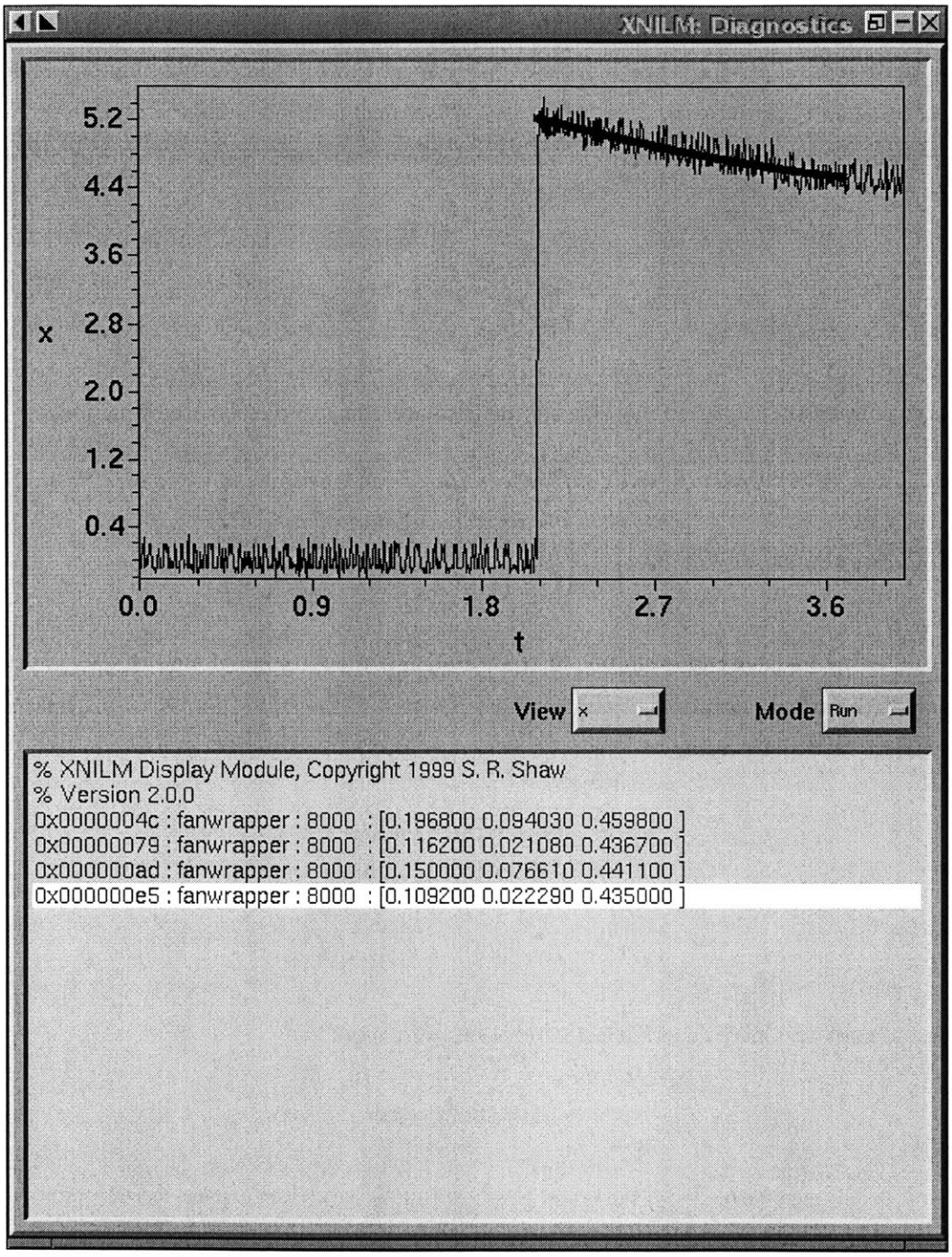


Figure 4-35: Screen shot of `xnilm` showing system identification results for the ventilation fan. The tags contain parameter information. Solid lines in the graph represent raw data; the points show the results of a simulation with the estimated parameters.

4.3.2 Induction motor in the mock building

A $\frac{1}{4}$ horsepower, 1725 rpm, Robins and Myers induction machine connected to the three-phase 120 VAC line-to-neutral service in the mock building was used to test the combination of nonintrusive classification and identification in an AC situation. A small pulley was used to change the inertial load seen by the motor and simulate circumstances where the parameters of the mechanical load are of interest.

Model

The dynamics of the induction machine are modeled according to the dq-space equations

$$\frac{d}{dt}\begin{pmatrix}\lambda_{qs}\\ \lambda_{ds}\\ \lambda_{qr}\\ \lambda_{dr}\end{pmatrix} = \begin{pmatrix}v_{qs}\\ v_{ds}\\ 0\\ 0\end{pmatrix} - \begin{pmatrix}r_s i_{qs} + \omega_0 \lambda_{ds}\\ r_s i_{ds} - \omega_0 \lambda_{qs}\\ r_r i_{qr} + (\omega_0 - \omega)\lambda_{dr}\\ r_r i_{dr} - (\omega_0 - \omega)\lambda_{qr}\end{pmatrix}, \tag{4.20}$$

where ω_0 is the frequency of excitation at the stator, ω is the rotor speed, and the λ's are the flux linkages with rotor quantities and parameters as reflected to the stator [58, 37]. Flux linkages and currents are related according to

$$\begin{aligned}\lambda_{qs} &= L_l i_{qs} + (L_l + L_m)(i_{qs} + i_{qr})\\ \lambda_{ds} &= L_l i_{ds} + (L_l + L_m)(i_{ds} + i_{dr})\\ \lambda_{qr} &= L_l i_{qr} + (L_l + L_m)(i_{qs} + i_{qr})\\ \lambda_{dr} &= L_l i_{dr} + (L_l + L_m)(i_{dd} + i_{dr}).\end{aligned}$$

The usual mass and inertia mechanical interactions were used, i.e.

$$\frac{d}{dt}\omega = 3K(\tau - \beta\omega) \tag{4.21}$$

where $\tau = \lambda_{qr} i_{dr} - \lambda_{dr} i_{qr}$ is proportional to the torque of electrical origin. To accommodate the single-phase raw data available from the mock building, the output equation was

$$i_{as} = i_{qs}\cos(\omega_0 t + \phi) + i_{ds}\sin(\omega_0 t + \phi), \tag{4.22}$$

where ϕ is the angle in the line cycle when the machine was switched on. Unlike the pump experiments in IEC/ERS, where ϕ was a parameter, the "wrapper" code determined ϕ from the voltage

Dataset	Load	r_s	r_r	L_m	L_l	K	β
1	Y	11.6	5.74	.778	.0197	613	.00191
2	Y	11.7	5.75	.776	.0195	613	.00189
3	N	11.8	5.69	.761	.0196	656	.00188
4	N	11.6	5.74	.774	.0195	638	.00189
5	N	11.6	5.73	.775	.0195	630	.00189
6	N	11.6	5.72	.770	.0196	637	.00189
7	Y	11.7	5.73	.774	.0194	610	.00190
8	Y	11.7	5.75	.775	.0194	618	.00190
9	Y	11.7	5.75	.773	.0194	613	.00190
10	Y	11.7	5.74	.774	.0194	610	.00190
11	N	11.6	5.77	.772	.0196	637	.00188
12	N	11.6	5.76	.772	.0197	642	.00189

Table 4.7: Parameter estimates for mock building induction motor classification and identification tests. Electrical parameters and β should be roughly the same as the motor and mechanical damping were not changed during the test. Since $K \propto \frac{1}{J}$, K should be smaller in loaded cases than when unloaded.

waveform before starting an iterative minimization of the residual

$$r = i - \hat{i}_{as}(\mu), \tag{4.23}$$

with measurements i and parameter vector

$$\mu = (r_s \quad r_r \quad L_m \quad L_l \quad K \quad \beta)'. \tag{4.24}$$

In the AC configuration of NITC, the preprocessor phase-locks in software to compute spectral envelopes; the "raw data" consists of current and voltage measurements resampled to a power of two points per line cycle. In this particular case the sampling rate was about 7.7 kHz or 128 points per line cycle.

Results

Figure 4-36 shows a typical screen shot of NITC matching a motor turn-on transient in the mock building. Notice that the "motorwrapper" model is specified in the tag. This model is invoked on the raw transient data, generating the output for another `xnilm` process to display identification results. Figure 4-37 shows a typical identification screen shot. The model fits well.

Parameters obtained using the identification routines dispatched by NITC were remarkably consistent. In Table 4.7, the electrical parameters and β should be essentially the same for all twelve trials. The estimates for K should reflect whether or not the load, a 2.5 inch diameter cast iron

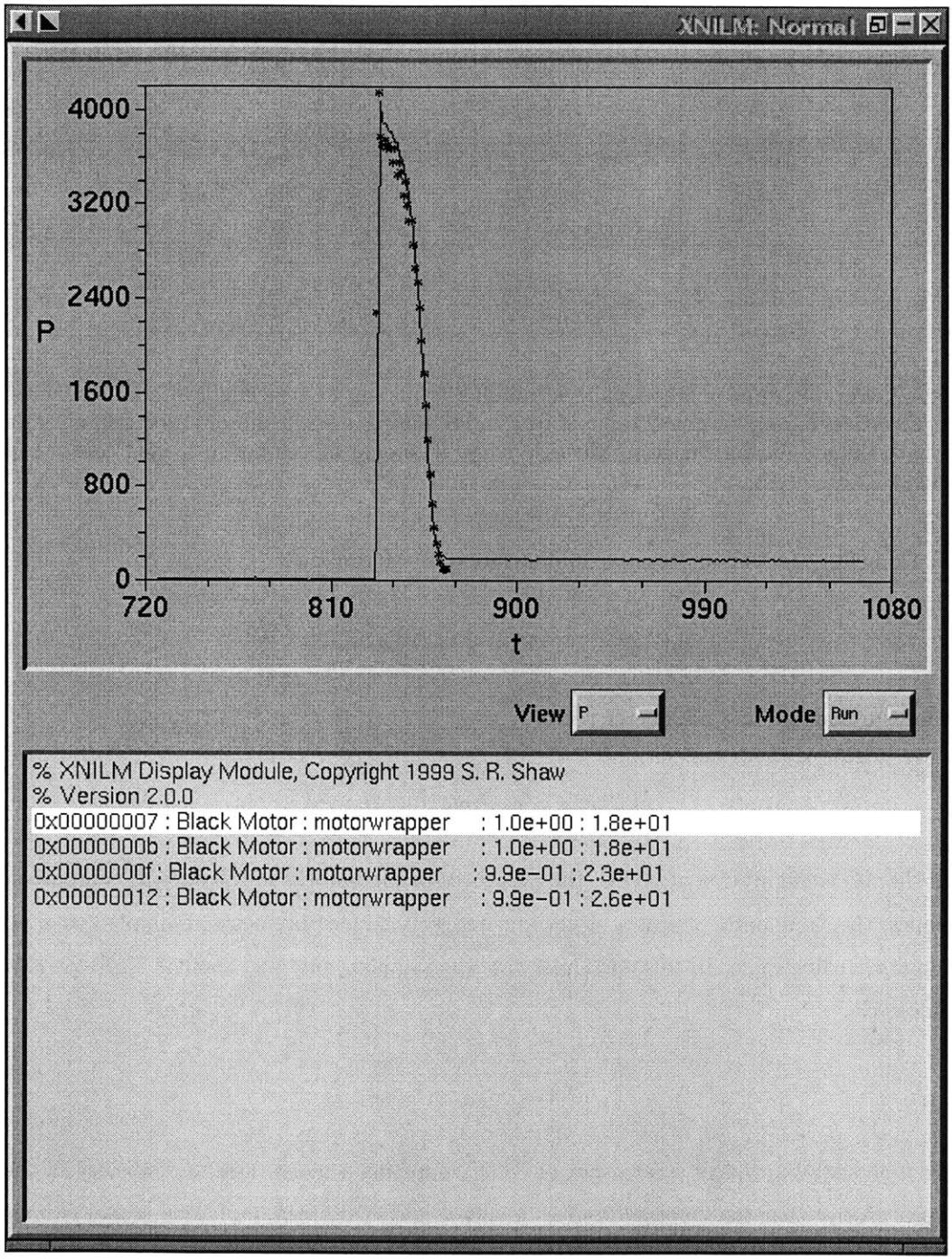


Figure 4-36: Screen shot of `xnilm` showing detection of induction motor transients in the mock building using NITC. Solid lines in the graph are spectral envelope data; the overlayed points show the fit of exemplar to transient.

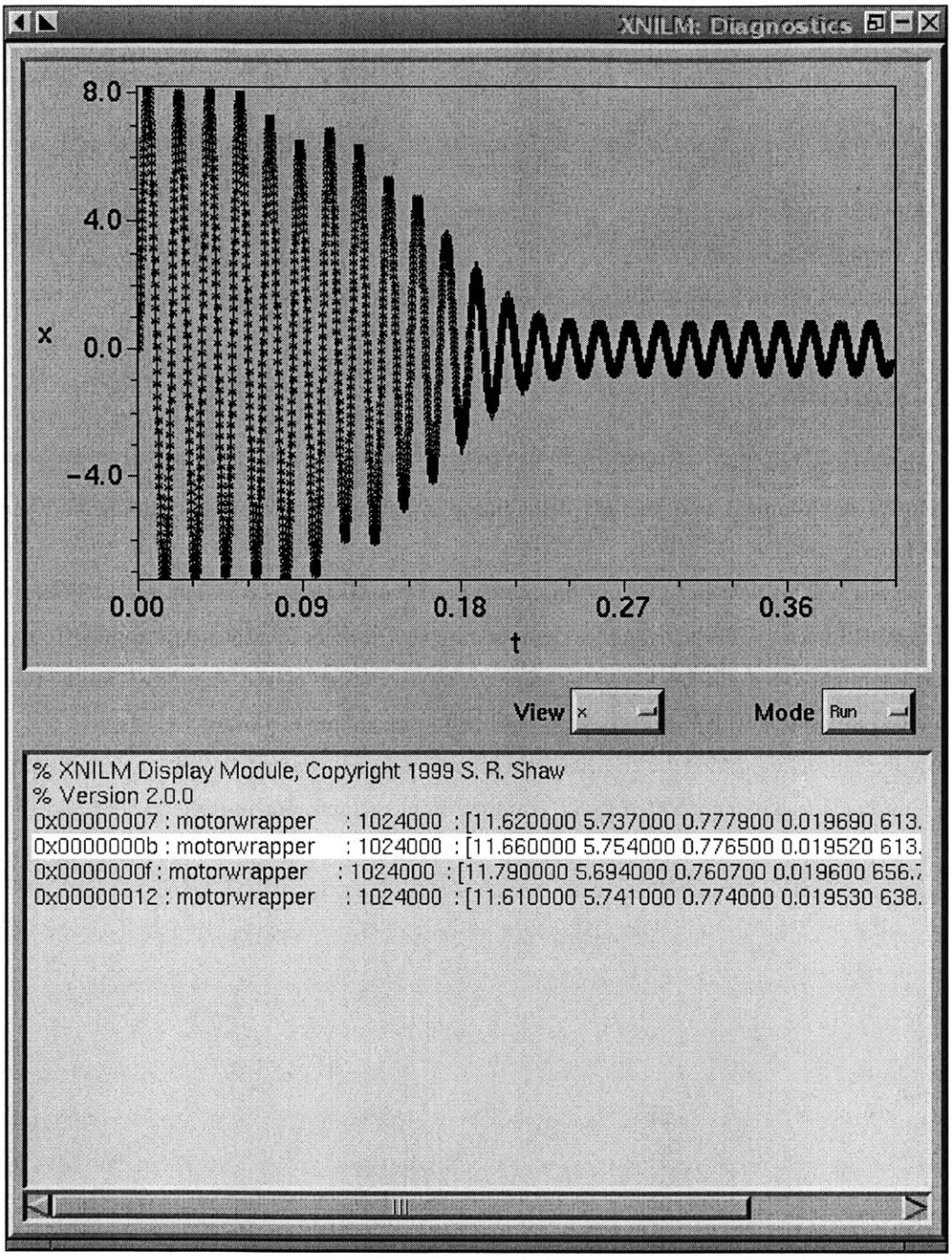


Figure 4-37: Screen shot of `xnilm` showing system identification results for the induction motor in the mock building. The tags contain parameter information. Solid lines in the graph represent raw data; the points show the results of a simulation with the estimated parameters.

single groove sheave, was attached to the shaft. Since $K \propto \frac{1}{J}$, the loaded trials should have smaller values of K. This is the case in Table 4.7.

Unlike the off-line study of the chilled water circulation pump at IEC/ERS, no synthetic datasets were used to estimate parameter distributions for the parameters in Table 4.7. The parameters in Table 4.7 were obtained in real time for the purpose of validating the dispatch of system identification code from NITC – there aren't enough trials to make a reasonable histogram comparing the loaded and unloaded conditions. However, it is interesting to note that the loaded condition (with sheave) has an average K value of 612.8 with a sample standard deviation of 2.9. The average value of K in the unloaded condition is 640.0, with a sample standard deviation of 8.7.

4.4 Summary

The nonintrusive transient classification (NITC) methods of Chapter 2 were demonstrated in a DC automotive environment and the AC distribution situations in the mock building and the Next House laundry service.

The system identification ideas of Chapter 3 were demonstrated on real data for HVAC systems in the IEC/ERS facility. These examples demonstrate that system identification applied to electrical transients might be useful in detecting changes on the order of the simulated faults that were introduced for testing. These results were obtained using submetered data.

Finally, the combination of transient classification from Chapter 2 and system identification from Chapter 3 was demonstrated using examples from the automotive environment and the mock building.

Chapter 5

Conclusions

This thesis proposes techniques that are essential to the extension of nonintrusive monitoring to include load diagnostics. These contributions include a new approach and implementation of the transient classification aspect of nonintrusive load monitoring; methods for system identification with a poor initial guess and no user intervention, i.e. in the context of nonintrusive monitoring; and examples of practical modeling, instrumentation and results from the field.

This thesis provides a framework to support nonintrusive monitoring applications, rather than a specialized solution. The tools developed in the thesis may be scaled from unambitious scripts assembling load usage data to a complicated collection of programs that classify and find parameters for load transients while serving the results to the web. The code needed to adapt the tools in this thesis to new problems and new situations can be added modularly, and using whatever language is preferred.

Aspirations of generality notwithstanding, the techniques and implementation described in this thesis are ready for almost immediate application. Nonintrusive load monitoring no longer requires rack-mounted equipment, costly custom parts, or training data stored on EPROM. Using a network or modem connection, data can be obtained immediately and a completely generic monitor can be tailored to a site, remotely. The installation in Next House only required a few days to build, install and train, and demonstrates how swiftly the implementation in this thesis can move from lab to the field.

Contributions in specific areas are summarized in the following sections.

5.1 Nonintrusive transient classification

The nonintrusive transient classification techniques proposed in Chapter 2 perform well in both AC and DC environments, in the lab and in the field. Although a comparison of the techniques

proposed in Chapter 2 to previous methods was not an objective of the thesis, the results in Chapter 4 demonstrate capabilities are at least equal to other efforts. The general idea of nonintrusive load monitoring is not new. However, novel results in Chapter 4 include demonstration of a new method, a new implementation using off-the-shelf hardware, the application of nonintrusive monitoring in an automobile, and the "bootstrap" training of a monitor in a remote location using a network. In addition, the implementation of nonintrusive classification in this thesis allows classification and identification processes to be combined.

The implementation in this thesis has advanced nonintrusive monitoring to the point where future work can and probably should be guided by observations from real applications. This development would probably proceed most smoothly in a commercial environment, incorporating feedback from customers who have a use for the technology.

5.2 System identification methods

The methods of Chapter 3 are motivated and tested on simulation examples in Chapter 3 and field situations in Chapter 4. These methods complement existing nonlinear estimation techniques.

The performance the identification methods on the IEC/ERS pump and fan data and the implications of these results for HVAC diagnostics are encouraging. In the tests performed, some truly remarkable details appear in the parameters. It is important to qualify this success appropriately, however. For example, the pump test results show that perturbations in the system *such as* an obstructed cooling coil can be detected using the techniques given. This does not imply that cooling coil obstructions can be detected to the exclusion of any other cause.

Despite the overall success of the experiments at IEC/ERS, there are aspects that could use improvement. A better fit might be achieved with an improved mechanical model of the fan data, at the risk of creating a model too complex to be supported by typical observations. One possibility for future work is to assume that the induction motor is essentially constant and characterize it accurately. This characterization could be done in several unloaded startup transient tests, for example. By fixing the induction motor parameters, and assuming that the interesting parameters are the mechanical ones, more complicated models might be feasible.

The fan and pump tests gravitated towards essentially the same model, and that the model worked fairly well in both cases. It is tempting to suppose that this model may have more general application to other HVAC situations. This may also prove an interesting direction for future work.

5.3 Nonintrusive classification and identification

The combination of transient classification followed by system identification worked quite well in examples given in the results, and is an exciting area for future work. One possibility is to write models for a collection of systems and explore how these perform in realistic nonintrusive monitoring conditions in the field. An equally important concern for future work should be to determine what information, in terms of load parameters in a building or vehicle, are important to potential industrial or commercial users. The usefulness of a model, and the parameters obtained by fitting it, depends on the intended application.

The methods developed in Chapter 3 may also benefit from further work. It is possible that an improved method might result from reconsidering the boundaries imposed between the continuation in time, the nonlinear least squares iteration, and the integration of the physical model.

5.4 Directions for future work

Future work could proceed efficiently in several directions. One plan, emphasizing the transient classification contributions of the thesis, might involve further deployment in field situations and the application of nonintrusive techniques to problems specific to those installations. Important areas of inquiry include efficient ways to collect and handle training data, the possibility of making generic sets of exemplars, performance in high event rate environments, and exemplar design. For example, in diagnostic applications a transient classifier that can distinguish individual loads might be more useful than a transient classifier that can distinguish classes of loads. An important challenge for future work is to understand the implications of different classification requirements in terms of exemplar design and test these conclusions in applications. It would also be extremely valuable to understand the practical tradeoffs involved as increasingly aggregated measurements are used for monitoring. Several nonintrusive load monitoring applications could follow directly from the results of this thesis in the form of postprocessing programs attached to the nonintrusive transient classifier. In some cases, these postprocessing programs might only be a few lines of Perl. Possibilities include postprocessing scripts that bound the operating times of individual loads using nonintrusive measurements, "watchdog" postprocessing programs that collect load leading up to and immediately after a fault, smart meters that sort power consumption by load, and monitors interfaced to building energy management and other control systems that validate commands sent to actuators by checking for the transients associated with these actuators.

The pre-estimation schemes suggested in Chapter 3 are another interesting area for future work. These techniques seem promising, especially if they can reliably produce estimates that are useful to generic nonlinear least squares codes. A first step might be to apply pre-estimation methods to data

collected in the field. This approach could build directly on the pre-estimators of Chapter 3 that were tested using simulation data. A more comprehensive approach might target concrete guidelines and simple tools for designing and testing pre-estimators.

A tool to help build and test models in the context of identification and pre-estimation would be very helpful in further work in nonintrusive identification and diagnostics. Such a tool would allow the user to write models and, given typical data, help assess what parts of the model might be difficult to identify or how the model might be re-parameterized to ease identification. Using essentially the same techniques to help the user with physical models, the program might also assist with the development of arbitrary models for purposes of pre-estimation and help the user evaluate these in the context of real data.

The combination of nonintrusive classification with system identification of key loads is an especially interesting area that should be explored beyond the results presented in this thesis. A first step in this direction might combine the nonintrusive identification results of Chapter 4 with the applications investigated in IEC/ERS. In principle, this would involve only involve finding exemplars suitable for IEC/ERS, and would directly demonstrate of the techniques of this thesis in a transient-rich environment.

Appendix A

Implementation and invocation

The techniques described in this thesis are implemented through an extensive collection of programs. This appendix describes these programs and how they can be connected together to reflect diagrams like Fig. 2-4. The description is split into three sections. The first section covers programs used in the nonintrusive classifier of Chapter 2. The second section follows with programs implementing the ideas of Chapter 3. In the third section, example command lines illustrate how to make these programs work together.

Table A.1 gives an overview of the dependencies and functions of software used. Table A.1 does not include universally available and commonly used libraries or packages.

A.1 Nonintrusive classification

The `nilm` program performs the tasks outlined in Chapter 2 for all hardware configurations and input types. This is achieved by dynamically loading a shared object module containing situation-specific preprocessing code and configuration information. Using this technique, new preprocessing capabilities can be added without recompiling or modifying the `nilm` source files in any way. The file `prep.so` (see dynamic loader documentation on the particular system in question for default location) must be a symbolic link to a shared object appropriate to the hardware being used, unless a alternate shared object is specified on the command line. Current shared object modules include `prep-car.so`, `prep-pcl818.so` and `prep-dm6420.so`.

A.1.1 `nilm`

The `nilm` program implements the nonintrusive classifier of Chapter 2. The basic command line is

```
nilm input_source exemplar_prefix [options].
```

Program	Requires	Function
`prep-car.so`	`/dev/ttySx` or stream	dynamically loaded car preprocessor
`prep-pcl818.so`	`pcl818.o` or stream	dynamically loaded pcl818 preprocessor
`prep-dm6420.so`	`dm6420.o` or stream	dynamically loaded dm6420 preprocessor
`prep.so`	one of the above	link or copy of one of the above
`nilm`	`prep.so`, exemplars	nonintrusive transient classifier
`prep`	`prep.so`	preprocessor
`train`	`prep`, input source	creates training file for `vsection`
`vsection`	training or exemplar file	edits exemplar files
`pcl818.o`	`/dev/pcl818`	driver for Advantech PCL818L ISA cards
`dm6420.o`	`/dev/dm6420`	driver for DM6420 PC/104 card
`setpcl`	`/dev/pcl818`	configuration program for above drivers
`xnilm`	X server, XForms and graphics stream	graphic display module for X
`w3nilm`	gnuplot, graphics stream	graphic display module and http server
`diag`	diagnostic stream, analysis programs	diagnostic dispatch program
`id-induction`		induction motor identification program
`id-dcmotor`		dc motor /w fan identification program
`id-iowa3`		identification for Iowa situation
`sim-induction`		induction motor simulation program
`sim-dcmotor`		dc motor /w fan simulation program
`sim-iowa3`		simulation for Iowa situation
`fanwrapper`	`id-dcmotor`	nonintrusive diagnostic wrapper for car ventilation system
`motorwrapper`	`id-induction`	nonintrusive diagnostic wrapper for induction motor
`counttags`	tag stream from `nilm`	contact totalizer

Table A.1: Overview of programs and program dependencies for nonintrusive classification and diagnostics.

The required argument `input_source` is usually the name of a device special file, e.g. `/dev/pcl818`, but it may also be a file or `stdin`.

The required argument `exemplar_prefix` specifies the prefix of the exemplar files to use for matching. The exemplars are stored in the files `exemplar_prefix.desc` and `exemplar_prefix.dat`. The `.desc` file contains user descriptions that are entered using `vsection`. This is a simple text file and may be edited – for example, the description file may be modified to make `nilm` aware of diagnostic capabilities for a particular exemplar. The `.dat` file is a matrix containing information about the exemplars – for format information, it is best to look at the `vsection` program. The `.dat` can be loaded and modified using `octave`.

The behavior of `nilm` is determined by the command line options. The flags `-t program`, `-g program` and `-d program` add tag, graphics or diagnostic programs (respectively) to the appropriate output queue. The option `-p shared_object` specifies a preprocessor shared object other than the default `prep.so`. This is particularly useful when running `nilm` off-line on a workstation or when testing several different configurations in one environment.

A.1.2 prep

The utility `prep` is a generic preprocessor program; it loads a shared object containing a particular preprocessor and uses the routines in that shared object to obtain preprocessed data from a device or stream. The command line is

```
prep input_source [options].
```

Input source is usually the name of a device special file, e.g. `/dev/pcl818`, but it may also be a file or `stdin`, which causes the program to read from standard input.

The options for `prep` are `-N number` and `-p shared_object`. As with `nilm`, `-p` allows explicit selection of a preprocessor module; otherwise `prep.so` is loaded. The option `-N number` specifies the number of *rows* of data to preprocess and output in ASCII form.

`prep` is used by the training programs, but may also be useful when debugging physical connections or for display of raw spectral envelopes.

A.1.3 train

The interactive Octave script `train` helps collect data for creating a set of exemplars with `vsection`. The idea is that the user will collect a bunch of training data at one time using `train`, then create exemplars later. The script `train` is a simple loop that collects a chunk of data on user command, looks for an index event, and accumulates the chunks in a file suitable for use by `vsection`. The command line is

```
train device datfile,
```

where `device` is either a regular or device special file, and `outfile` is the name of the intermediate file used with `vsection`.

A.1.4 `vsection`

The interactive Octave script `vsection` helps to create a set of exemplars from a data file created with `train`. The script `vsection` is menu-driven and allows the user to specify the areas of support of the sections in the exemplars, specify names and model types, and edit existing exemplars. The command line is

```
vsection [datfile].
```

If no `datfile` is specified, the user may edit existing exemplar files, but will have to load them using menu selections.

A.1.5 `xnilm`

The `xnilm` utility is a graphical display program, outlined in Chapter 2. The command line is

```
xnilm [-d] [-c].
```

In addition, `xnilm` handles all the normal options of X11 programs. Without flags, `xnilm` expects a graphics stream consistent with an AC environment, i.e. several spectral envelope estimate channels and corresponding exemplar matches are expected. The `-d` flag prepares `xnilm` for a graphics stream from a diagnostic application – i.e. a channel of raw data and an overlay corresponding to the fit of a physical model. The `-c` flag prepares `xnilm` for data from a DC environment.

A.1.6 `w3nilm`

The `w3nilm` program is a web-based graphical display utility, outlined in Chapter 2. The command line is

```
w3nilm [-d] [-c] [-p port].
```

The options are the same as `xnilm` except for the `-p` option, which determines the port that `w3nilm` will serve results from. The port specified to `w3nilm` should be unprivileged.

A.2 Diagnostics and system identification

The identification and simulation programs used to construct pre-estimators and in diagnostic wrappers are named `sim-` or `id-` followed by a model name. These programs have a common set of options.

A.2.1 `sim` programs

A `sim` program command line has the format

```
sim-[model] [-d] [-o option_string] [-p parameter_string] [-i inputfile] [-t]
```

The `-d` option causes the `sim` program to print the default parameters used by the simulator. The `-o` flag passes an "option string" to the model when it is initialized. For example, the induction motor model has option strings that control whether the output is in the $dq0$ frame, in the lab frame, or a single phase in the lab frame. Valid option strings depend on the model. The `-p` flag allows specification of a set of parameters to use for the simulation. The string should be a quoted list of numbers. The `-i` flag provides for specification of an input file. The first column of the input file is time, followed by other inputs as needed by the model. If no input is needed by the model, the input file may be omitted and a default time axis will be used. The `-t` flag formats the output so that it can be used as an input for the `id` series of programs – in particular, the time axis is preserved on the first column of the output if `-t` appears.

A.2.2 `id` programs

The `id` programs have the same options as the `sim` programs, with one addition and one modification. The command line is

```
id-[model] [-d] [-o option_string] [-p parameter_string] [-i inputfile]
[-t threshold] [-r report]
```

The `-t` flag has a different meaning in `id` than `sim`. In `id`, `-t` allows specification of the threshold used to discriminate between local and global minima when searching. The argument should be a liberal estimate of standard deviation of the disturbance. The `-r` flag allows specification of the level of reporting desired. If the `-r` is followed by `all`, then estimated parameters and a simulation using those estimates will be output. If `parms` is used, just the estimated parameters will be output. Finally, if `outputs` is specified, the results of simulation using the estimated parameters will be output.

A.3 Example command lines

A.3.1 Using xnilm and pcl818

```
% nilm /dev/pcl818 exemplars -g xnilm
```

This starts nilm, reading data from the device special file /dev/pcl818, using exemplar files exemplars.dat and exemplars.desc, with graphical output via xnilm. In a DC monitoring scenario, the option -g "xnilm -c" would be used. Quotes ensure that the -c option is passed to xnilm rather than nilm.

A.3.2 Using w3nilm and dm6420

```
% nilm /dev/dm6420 exemplars -g "w3nilm -p 8080"
```

This command line starts nilm, reading data from the device special file /dev/dm6420, using exemplar files exemplars.dat and exemplars.desc, with graphical output via w3nilm. The options to w3nilm specify that it should serve the display from port 8080, so to see results Netscape would have to be invoked as follows

```
% netscape http://hostname:8080
```

where hostname is the name of the machine on which w3nilm is running.

A.3.3 Using a stdin or a file

```
% nilm rawdata exemplars -g xnilm
% fastsim rawdata | nilm stdin exemplars -g xnilm
% cat /dev/pcl818 | tee rawdata | nilm stdin exemplars -g xnilm
```

The nilm part of the invocation for these examples is the same, but the input source is handled differently. The first example reads from the file rawdata, preprocesses and displays the results. However, as soon as the end of file is reached the program will exit. The second example uses the Perl script fastsim, which echos the file rawdata on the standard output and stops without exiting at the end of file. This allows the user to investigate results displayed by xnilm before termination. The final example shows how Unix commands cat and tee can be used to simultaneously run nilm on a data stream and log the data stream to disk. This might be used to validate nilm's performance in a particular installation.

A.3.4 Using tags

```
% nilm /dev/pcl818 exemplars -t "counttags > log.tags"
% nilm /dev/pcl818 exemplars -t "cat - > log.tags "
```

In the first example, `nilm` processes tag data using the script `counttags` (Program 2.1) and records the output in the file `log.tags`. The second example records tags in a file using `cat`.

A.3.5 Using identification

```
% nilm /dev/pcl818 exemplars -g xnilm -d "diag | xnilm -d "
% nilm /dev/pcl818 exemplars -g "w3nilm -p 8080" -d "diag | w3nilm -d -p 8081 "
```

For the first example, `nilm` opens two instances of `xnilm`. One instance is used to display matches between exemplars and preprocessed data. The other instance of `xnilm` receives data from the programs invoked by `diag` and displays them using diagnostic mode.

The second example is similar, except that matches are served by `w3nilm` on port 8080, while diagnostic data and fits are displayed on port 8081. To view this data simultaneously, a user would use two browser windows.

Appendix B

Gauss-Newton for nonlinear least squares

The Gauss-Newton method underlies Levenburg-Marquardt and is important to the methods discussed in Chapter 3. Explanations of Gauss-Newton and Levenburg-Marquardt can be found in several references including [57, 3, 55, 22, 4]. The presentation given here emphasizes two ways of approaching the problem and is consistent with the notation used in Chapter 3.

B.1 Problem Statement

The nonlinear least squares problem is to find an m dimensional parameter vector estimate $\hat{\mu}$ of μ satisfying in the least squared sense the n equations

$$\begin{aligned} f_1(u_1;\mu) &= y_1 \\ f_2(u_2;\mu) &= y_2 \\ &\vdots \\ f_n(u_n;\mu) &= y_n, \end{aligned} \tag{B.1}$$

where u_k is the k'th input, y_k is the k'th observation, f_k is the k'th function relating these quantities, μ is the parameter vector, and $n > m$. The f_k are nonlinear in the way that components of μ are related to y. The semi-colon in (B.1) is to distinguish parameters from inputs in the arguments of the function. The distinction between observations and inputs is that observations are assumed to be measured with errors, while inputs are assumed to be known without error. This assumption is important to the statistical interpretation of the final estimate $\hat{\mu}$.

The non-linear least squares problem is often expressed more formally as the minimization of a

loss function $V(\mu)$. In particular, the final estimate is

$$\hat{\mu} = \arg\min_{\mu} V(\mu). \tag{B.2}$$

This notation means that the final estimate $\hat{\mu}$ is the value of μ for which the loss function $V(\mu)$ attains a minimum. For the least squares problem suggested by (B.1), the loss function is

$$V(\mu) = \frac{1}{2}\sum_{k=1}^{n}(y_k - f_k(u_k;\mu))^2. \tag{B.3}$$

The loss function can also be written $V(\mu) = \frac{1}{2}r'r$, where r is the n dimensional residual

$$r = y - f(u;\mu), \tag{B.4}$$

and lack of subscripts is interpreted so that

$$r_k = y_k - f_k(u_k;\mu). \tag{B.5}$$

B.2 Gauss-Newton iteration

The Gauss-Newton method iteratively finds a solution to the nonlinear least squares problem by updating $\hat{\mu}$ with a step δ_{GN}, i.e.

$$\hat{\mu}^{(i+1)} = \hat{\mu}^{(i)} + \delta_{GN}^{(i)}, \tag{B.6}$$

where $\hat{\mu}^{(i)}$ is the estimate of the parameters at the i'th iteration. Unless specifically labeled, assume all quantities correspond to the i'th iteration. At each step, δ_{GN} is computed as the least-square solution to the linear problem

$$J\delta_{GN} = r, \tag{B.7}$$

where J is the n by m Jacobian evaluated at $\hat{\mu}$ with elements

$$J_{k,i} = -\left.\frac{\partial r_k}{\partial \mu_i}\right|_{\hat{\mu}} \tag{B.8}$$

and r is the residual defined in (B.4). There are two ways to motivate this update.

B.2.1 Linearization approach

One approach to Gauss-Newton is to consider a linear Taylor series approximation to $f(u;\mu)$. For notational simplicity, the dependence on u is neglected. As part of an iteration, this linearization

should be in the neighborhood of the current estimate $\hat{\mu}$, i.e.

$$f(\hat{\mu}+\delta) \approx f(\hat{\mu}) + \nabla f(\hat{\mu})\delta, \tag{B.9}$$

where $\nabla f(\hat{\mu})$ is the m dimensional row vector of partial derivatives

$$\nabla f(\hat{\mu}) = \left(\left.\frac{\partial f}{\partial \mu_1}\right|_{\hat{\mu}} \quad \left.\frac{\partial f}{\partial \mu_2}\right|_{\hat{\mu}} \quad \cdots \quad \left.\frac{\partial f}{\partial \mu_m}\right|_{\hat{\mu}} \right). \tag{B.10}$$

The goal is to obtain a δ to compute the next parameter estimate $\hat{\mu}+\delta$ by combining the linearization of (B.9) with the constraints. Substituting (B.9) into (B.1) evaluated at the proposed update $\hat{\mu}+\delta$ yields

$$\begin{aligned} f_1(\hat{\mu}) + \nabla f_1(\hat{\mu})\delta &= y_1 \\ f_2(\hat{\mu}) + \nabla f_2(\hat{\mu})\delta &= y_2 \\ &\vdots \\ f_n(\hat{\mu}) + \nabla f_n(\hat{\mu})\delta &= y_n. \end{aligned} \tag{B.11}$$

The term $\nabla f_k(\hat{\mu})$ is the k'th row of J defined in (B.8). Subtracting the f_k's from both sides of each equation, (B.11) may be rewritten as the linear system

$$J\delta = r. \tag{B.12}$$

Multiplying both sides by J' formally yields the Gauss-Newton step

$$J'J\delta_{GN} = J'r, \tag{B.13}$$

although an implementation would probably not solve the problem using $J'J$, since the condition number of $J'J$ is the square of the condition of J.

B.2.2 Newton approach

A different approach to Gauss-Newton begins with the scalar loss function $V(\mu)$ defined in (B.3). A minimum of $V(\mu)$ attained at some value μ has the property that the m dimensional gradient

$$g(\mu) = \nabla V(\mu)' \tag{B.14}$$

is zero. The gradient here is taken to be a row of partial derivatives, so that $g(\mu)$ is a column vector with m rows.

The problem is to find a solution $\hat{\mu}$ giving $g(\hat{\mu}) = 0$. This involves m equations in m unknowns, moreover, these equations have an *exact* solution. Taking a Taylor series expansion of $g(\mu)$ yields

$$g(\hat{\mu} + \delta) \approx g(\hat{\mu}) + \nabla g(\hat{\mu})\delta. \tag{B.15}$$

The goal is to force g evaluated at the updated step $\hat{\mu} + \delta$ to zero. Note that the object $\nabla g(\hat{\mu})$ is m by m. Newton's method prescribes the step δ_N obtained as the solution to

$$\nabla g(\hat{\mu})\delta_N = -g(\hat{\mu}). \tag{B.16}$$

The relationship of this step to Gauss-Newton is revealed by substituting $V(\mu)$ and collecting terms. First,

$$\begin{aligned} g(\mu) &= \nabla V(\mu)' \\ &= \frac{1}{2}\sum_{i=1}^{n} \left(\frac{\partial}{\partial \mu_1} \quad \frac{\partial}{\partial \mu_2} \quad \cdots \quad \frac{\partial}{\partial \mu_m} \right)' r_i^2 \\ &= \sum_{i=1}^{n} \left(\frac{\partial r_i}{\partial \mu_1} \quad \frac{\partial r_i}{\partial \mu_2} \quad \cdots \quad \frac{\partial r_i}{\partial \mu_m} \right)' r_i. \end{aligned} \tag{B.17}$$

The term $\left(\frac{\partial r_i}{\partial \mu_1} \quad \frac{\partial r_i}{\partial \mu_2} \quad \cdots \quad \frac{\partial r_i}{\partial \mu_m} \right)'$ above is a column of $-J'$, defined in (B.8). The sum is an inner product between rows of $-J'$ and r, therefore

$$g(\mu) = -J'r. \tag{B.18}$$

It is helpful to start with last line of (B.17) to obtain an expression for the Hessian $\nabla g(\mu)$. Expanding the ∇ operator,

$$\nabla g(\mu) = \left(\frac{\partial}{\partial \mu_1} \quad \frac{\partial}{\partial \mu_2} \quad \cdots \quad \frac{\partial}{\partial \mu_m} \right) \sum_{i=1}^{n} \begin{pmatrix} \frac{\partial r_i}{\partial \mu_1} \\ \frac{\partial r_i}{\partial \mu_2} \\ \vdots \\ \frac{\partial r_i}{\partial \mu_m} \end{pmatrix} r_i. \tag{B.19}$$

Interchanging the sum and the vector of partials and using the product rule,

$$\nabla g(\mu) = \sum_{i=1}^{n} r_i \begin{pmatrix} \frac{\partial^2 r_i}{\partial \mu_1 \partial \mu_1} & \cdots & \frac{\partial^2 r_i}{\partial \mu_1 \partial \mu_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 r_i}{\partial \mu_n \partial \mu_1} & \cdots & \frac{\partial^2 r_i}{\partial \mu_n \partial \mu_n} \end{pmatrix} + \sum_{i=1}^{n} \begin{pmatrix} \left(\frac{\partial r_i}{\partial \mu_1}\right)^2 & \cdots & \frac{\partial r_i}{\partial \mu_1}\frac{\partial r_i}{\mu_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_i}{\partial \mu_n}\frac{\partial r_i}{\partial \mu_1} & \cdots & \left(\frac{\partial r_i}{\partial \mu_n}\right)^2 \end{pmatrix}. \tag{B.20}$$

Certain terms in (B.20) can be recognized. First, the whole expression is the Hessian of $V(\mu)$. Proceeding from left to right, the matrix in the first summation is the Hessian of r_i. Up to a sign, this matrix is also the Hessian of f, so it contains second order information about the model. For

any particular i, the matrix in the second summation is the m by m outer product of the i'th row of the Jacobian with itself. The expression $\nabla g(\mu)$ can then be rewritten in terms of J

$$\nabla g(\mu) = A + J'J, \tag{B.21}$$

where A is the first summation expression in (B.20). Substituting expressions for $\nabla g(\mu)$ and $g(\mu)$ in (B.16) yields an expression for the step δ_N,

$$(A + J'J)\delta_N = J'r. \tag{B.22}$$

In a practical setting, obtaining the second order information required for A may be undesirable. If A is discarded and the Hessian of $V(\mu)$ is approximated by $J'J$, the Gauss-Newton step

$$J'J\delta_{GN} = J'r \tag{B.23}$$

is obtained.

Appendix C

Source Code

C.1 Nonintrusive classifier source code

Program C.1 *Main module* `nilm.c`.

The following code is the main part of the `nilm` *program. It loads the exemplars, presents data to the matching routines, and manages the output queues.*

```
/* generated Fri Dec 10 15:39:50 1999 on nitro */                    0

/*

                    Written by Steven R. Shaw, 1999

              Copyright (c) 1999 MASSACHUSETTS INSTITUTE
                  OF TECHNOLOGY. All rights reserved.

*/
                                                                     10


#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <math.h>
#include <err.h>
#include <assert.h>
                                                                     20
#include "prep.h"

#include "mgif.h"
#include "siom.h" /* synchronous I/O multiplexing */
#include "match.h" /* pattern matching */
#include "tee.h" /* I/O manifolds */
#include "blas.h"
#include "split.h"
```

```
#define ALPHA 1                                                          30

#ifdef MORE_DEBUG
#define CHECKPOINT() warnx("function %s, line %d", __FUNCTION__,
__LINE__);
#else
#define CHECKPOINT()
#endif

typedef struct
  {                                                                      40
    match_t match;
    matrix_t X;
    matrix_t iX;
    matrix_t pX;

    matrix_t raw;
    matrix_t iraw;
    int raw_stage_length;

    prep_t prep;                                                         50

    ll_t *tags, *graphics, *diagnostics;
    mgif_t *tmpg, *tmpo;

    int siominputkey;
    int cnt;
  }
nilm_t;

void                                                                     60
indexspace (float *y, int a, int b)
{
  int i;
  for (i = a; i <= b; i++)
    *y++ = i;
}

void
nilm_output (nilm_t * n)
{                                                                        70
  static char tag[100], str[200];
  char *spl[3];
  exemplar_t *ex;
  ring_t *hits;
  int ch, start, stop, M, N;

  hits = &(n->match.hits);

#ifdef MORE_DEBUG
  warnx ("%s : number of hits = %d",                                     80
  __FUNCTION__, ring_items (hits));
#endif

  N = ring_items (hits);

  while ((ex = (exemplar_t *) ring_get (hits)))
```

```
   {

     snprintf (tag, 80, "0x%8.8x : %s : %.1e : %.1e \n",
n->cnt, ex->name, ex->gain, ex->residual);

     tee_write (n->tags, tag, strlen (tag));

     if (n->graphics)
{
  M = 2 * matrix_columns (&(n->pX));

  ch = strlen (tag);
  *((int *) (tag + ch)) = M;
  tee_write (n->graphics, tag, ch + sizeof (int));
  *(tag + ch) = 0;

  exemplar_data_range (ex, &start, &stop);

  start = start - 100 + STAGE_LENGTH;
  stop = stop + 200 + STAGE_LENGTH;

  if (start < 1)
    start = 1;
  if (stop > matrix_rows (&(n->pX)))
    stop = matrix_rows (&(n->pX));

  mgif_setn (n->tmpg, 1 + stop - start);
  indexspace (mgif_xptr (n->tmpg), start - STAGE_LENGTH, stop -
      STAGE_LENGTH);

  for (ch = 1; ch <= M / 2; ch++)
    {

      dfxpy (mgif_yptr (n->tmpg),
     matrix_ref (&(n->pX), start, ch),
     0.0, 1 + stop - start);

      tee_write (n->graphics,
 mgif_ptr (n->tmpg),
 mgif_size (n->tmpg));

      exemplar_overlay (ex, n->tmpo, ch);

      tee_write (n->graphics,
 mgif_ptr (n->tmpo),
 mgif_size (n->tmpo));
    }
}

     strcpy (str, ex->name);
     split (spl, str, ":", 2);
     if (n->diagnostics && N == 1 && spl[1] != NULL)
{
  int nBytes, i;

  exemplar_data_range (ex, &start, &stop);
  start = ((start + STAGE_LENGTH / 2) * n->raw_stage_length) /
```

```
    STAGE_LENGTH;

  warnx ("start = %d", start);

  if (start < 1)
    start = 1;
                                                                          150
  nBytes = n->raw_stage_length * sizeof (double);

  snprintf (tag, 80, "0x%8.8x : %s : %d : %.1e : %.1e \n",
    n->cnt, ex->name, matrix_columns (&(n->raw)) * nBytes,
    ex->gain, ex->residual);

  tee_write (n->diagnostics, tag, strlen (tag));
  for (i = 1; i <= matrix_columns (&(n->raw)); i++)
    tee_write (n->diagnostics, matrix_ref (&(n->raw), start, i), nBytes);
}                                                                         160
   }
}

static void
matrix_shift (matrix_t * X, int k)
{
  int i, n;

  n = matrix_rows (X) - k;
                                                                          170
  for (i = 1; i <= matrix_columns (X); i++)
    memmove (matrix_ref (X, 1, i),
      matrix_ref (X, 1 + k, i),
      sizeof (double) * n);
}

static void
nilm_input (int fd, void *mydata)
{
  nilm_t *n = (nilm_t *) mydata;                                          180
  match_t *m = &(n->match);
  prep_t *p = &(n->prep);
  int rval;

#if  0
  double *ptr = matrix_data (&(n->iX));
  *ptr = 3.14159;
#endif

  rval = (*(p->read)) (p->data,                                           190
        matrix_data (&(n->iX)),
        matrix_lda (&(n->iX)),
        matrix_data (&(n->iraw)),
        matrix_lda (&(n->iraw)));

  if (rval == 1)
    {

#if 0
      if (*ptr == 3.14159)                                                200
```

```
 errx (1, "You're getting toasted");
#endif

      match_getevents (m, &(n->X));

      do
 {
   rval = match (m, &(n->X));
   CHECKPOINT ();
   nilm_output (n);                                                    210
 }
      while (rval);

      n->cnt += 1;

      matrix_shift (&(n->pX), STAGE_LENGTH);
      matrix_shift (&(n->raw), n->raw_stage_length);
      match_shift (m);
    }
  else if (rval == -1)                                                 220
    {
      siom_exitloop ();
      return;
    }
}

static void
nilm_usage (void)
{
  printf ("Usage:\n");                                                 230
  printf ("nilm input_resource pattern_resource [-c] [-t tags]");
  printf ("[-d diags] [-g graphics] [-p preprocessor.so] \n");
  printf ("$Revision:  $\n");
}

static void
nilm_commandline (nilm_t * n,
   int argc,
   char **argv,
   char *preprocessor)                                                 240
{
  extern char *optarg;
  int ch;

  while ((ch = getopt (argc, argv, "t:g:d:p:")) != EOF)
    {
      switch ((char) ch)
 {
 case 't':
   n->tags = tee_addoutput (n->tags,                                  250
    optarg,
    32000);
   break;
 case 'g':
   n->graphics = tee_addoutput (n->graphics,
        optarg,
    50 * sizeof (float) * 32 * STAGE_LENGTH);
```

```
    break;
  case 'd':
    n->diagnostics = tee_addoutput (n->diagnostics,                    260
    optarg,
    5000000);
    break;
  case 'p':
    strcpy (preprocessor, optarg);
    break;

  case ':':
    errx (1, "%s : flag missing required argument at line %d",
  __FUNCTION__, __LINE__);                                             270
    break;
  case '?  ':
  default:
    nilm_usage ();
    errx (1, "%s : unrecognized option at line %d",
  __FUNCTION__, __LINE__);
    break;
  }
     }
                                                                       280
   if (n->tags == NULL &&
       n->graphics == NULL &&
       n->diagnostics == NULL)
     warnx ("no outputs configured at line %d\n",
     __LINE__);

}

static void
nilm_init (nilm_t * n, int argc, char **argv)                          290
{
  static char preprocessor[200];

  CHECKPOINT ();

  siom_init ();
  n->cnt = 0;

  CHECKPOINT ();
                                                                       300
  n->tags = NULL;
  n->graphics = NULL;
  n->diagnostics = NULL;

  if (argc < 3)
    {
      nilm_usage ();
      errx (1, "%s : insufficient arguments at line %d\n",
     __FUNCTION__, __LINE__);
    }                                                                  310

  match_init (&(n->match), argv[2]);
  CHECKPOINT ();
```

```
  strcpy (preprocessor, "prep.so");

  nilm_commandline (n, argc - 2, argv + 2, preprocessor);
  CHECKPOINT ();

  prep_init (&(n->prep), preprocessor, argv[1], STAGE_LENGTH);                    320

  n->siominputkey = siom_add ((*(n->prep.handle)) (n->prep.data),
       "r", nilm_input, (void *) n);

  matrix_init (&(n->pX), 5 * STAGE_LENGTH, *(n->prep.columns));
  matrix_rowrange (&(n->X), &(n->pX), STAGE_LENGTH + 1,
    matrix_rows (&(n->pX)));
  matrix_set (&(n->pX), 0.0);
  matrix_rowrange (&(n->iX), &(n->X),
    matrix_rows (&(n->X)) - STAGE_LENGTH + 1,                                     330
    matrix_rows (&(n->X)));

  n->raw_stage_length = STAGE_LENGTH * (*(n->prep.rawshift));

  matrix_init (&(n->raw),
        5 * n->raw_stage_length,
        *(n->prep.rawcolumns));
  matrix_set (&(n->raw), 0.0);
  matrix_rowrange (&(n->iraw), &(n->raw),
    matrix_rows (&(n->raw)) - n->raw_stage_length + 1,                            340
    matrix_rows (&(n->raw)));

  warnx ("raw_stage_length = %d", n->raw_stage_length);

  CHECKPOINT ();

#ifdef MORE_DEBUG
  warnx ("matrix_rows(X) = %d", matrix_rows (&(n->X)));
#endif
                                                                                  350
#if (ALPHA==1)
  assert (matrix_rows (&(n->X)) == 4 * STAGE_LENGTH);
  assert (matrix_rows (&(n->iX)) == STAGE_LENGTH);
#endif

  n->tmpg = mgif_new (matrix_rows (&(n->pX)));
  n->tmpo = mgif_new (N_RECORD * (2 + MAX_SUBSECTIONS));

  CHECKPOINT ();
}                                                                                 360

static void
nilm_close (nilm_t * n)
{
  matrix_free (&(n->pX));

  matrix_free (&(n->raw));

  free (n->tmpg);
  free (n->tmpo);                                                                 370
```

```
  tee_free (n->tags);
  tee_free (n->graphics);
  tee_free (n->diagnostics);

  match_close (&(n->match));

  prep_close (&(n->prep));
}
                                                                    380
#ifdef MAIN_PROGRAM
int
main (int argc, char *argv[])
{
  nilm_t shawisgreat;

  CHECKPOINT ();
  nilm_init (&shawisgreat, argc, argv);
  CHECKPOINT ();
  siom_loop ();                                                     390
  CHECKPOINT ();
  nilm_close (&shawisgreat);
  CHECKPOINT ();

  return 0;
}
#endif
```

Program C.2 *Matching module* `match.c`.

The following code implements the exemplar matching procedures of §2.2.3. Most of the work is done by calls to the BLAS.

```
/* generated Fri Dec 10 15:39:50 1999 on nitro */                    0

/*

                  Written by Steven R. Shaw, 1999

            Copyright (c) 1999 MASSACHUSETTS INSTITUTE
                OF TECHNOLOGY. All rights reserved.

*/
                                                                    10


#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <math.h>
#include <err.h>

#include "mgif.h"                                                   20
#include "tools.h" /* C++ clean */
#include "filetolist.h"
#include "filter.h"
#include "match.h"
```

```
#ifdef MORE_DEBUG
#define CHECKPOINT() warnx("function %s, line %d", __FUNCTION__,
__LINE__);
#else
#define CHECKPOINT()                                                    30
#endif

static int __inline__
mini (double *v, int n)
{
  double save;
  int i, rval;

  save = *v;
  rval = 0;                                                             40

  for (i = 1; i < n; i++)
    if (save > v[i])
      save = v[rval = i];

  return rval;
}

void
exemplar_free (exemplar_t * e)                                          50
{
  if (e->sections)
    {
      free (e->sections);
      e->sections = NULL;
    }
}

int
exemplar_init (exemplar_t * e,                                          60
        char *name,
        matrix_t * M,
        int row)
{
  section_t *sp;
  int i, j;

  e->name = name;

  if (matrix_columns (M) != (5 + N_RECORD * 2))                         70
    errx (1, "%s : matrix appears to have the wrong number of columns",
   __FUNCTION__);

  e->nsections = matrix_element (M, row, 4) + 1;
  if (e->nsections >= MAX_SUBSECTIONS)
    errx (1, "%s : too many sections", __FUNCTION__);

  e->sections = (section_t *) malloc (sizeof (section_t) * (e->nsections));
  if (e->sections == NULL)
    errx (1, "%s : line %d : allocation error",                         80
   __FUNCTION__, __LINE__);
```

```
  if ((row + e->nsections - 1) > matrix_rows (M))
    errx (1, "%s : inconsistant row information",
   __FUNCTION__);

  sp = e->sections;
  for (i = 0; i < e->nsections; row++, i++, sp++)
    {
      sp->channel = matrix_element (M, row, 1);                    90
      sp->N = matrix_element (M, row, 2);
      sp->contactoffset = matrix_element (M, row, 5);

      if (i == 0 && sp->contactoffset != 0)
 errx (1,
     "%s : contact offset = %d on first section of %s must be zero!",
       __FUNCTION__, sp->contactoffset, name);

      if (sp->N > N_RECORD)
 errx (1, "%s : too many points in section",                  100
       __FUNCTION__);

      for (j = 0; j < sp->N; j++)
 {
   sp->x[j] = matrix_element (M, row, 6 + j);
   sp->t[j] = matrix_element (M, row, 6 + N_RECORD + j) + sp->
     contactoffset;
 }
    }
                                                               110
  e->mingain = .2;
  e->maxgain = 5.0;
  e->minstretch = .5;
  e->maxstretch = 2.0;

  return row;
}

void
exemplar_manifest (exemplar_t * e)                             120
{
  warnx ("%s : nsections = %d, name = %s",
  __FUNCTION__, e->nsections, e->name);
}

static void __inline__
dset (int N, double *p, int inc, double val)
{
  int i;
                                                               130
  for (i = 0; i < N; i++)
    {
      *p = val;
      p += inc;
    }
}

static void
```

```
section_adjust (section_t * s,
 double *b,                                                          140
 double *x,
 int *toff,
 double *xoff)
{

  static double P[N_RECORD * 2];
  static double B[N_RECORD * (2 * OFFSET_RANGE + 1)];
  static double work[2 * N_RECORD];
  static double v[2 * OFFSET_RANGE + 1];
  int i, j, ldp = N_RECORD;                                          150

  dset (s->N, P + ldp, 1, 1.0);
  dcopy (s->N, s->x, 1, P, 1);

  for (i = -OFFSET_RANGE; i <= OFFSET_RANGE; i++)
    for (j = 0; j < s->N; j++)
      B[(i + OFFSET_RANGE) * ldp + j] = x[s->t[j] + i];

  dgels ('N', s->N, 2, OFFSET_RANGE * 2 + 1,
  P, ldp, B, ldp, work, 2 * N_RECORD, &i);                           160
  if (i != 0)
    errx (1, "%s : arg %d of dgels had an illegal value.\n",
   __FUNCTION__, i);

  for (i = 0; i < (OFFSET_RANGE * 2 + 1); i++)
    v[i] = dnrm2 (s->N - 2, B + 2 + i * ldp, 1) / s->N;

  j = mini (v, (OFFSET_RANGE * 2 + 1));
  *toff = j - OFFSET_RANGE;
  *xoff = B[ldp * j + 1];                                            170

  for (i = 0; i < s->N; i++)
    b[i] = x[s->t[i] + (*toff)] - *xoff;
}

static void
exemplar_submatch (exemplar_t * ex,
    contact_t * cur,
    matrix_t * X)
{                                                                    180
  section_t *s;
  int i, nrows;
  static double A[N_RECORD * MAX_SUBSECTIONS];

  s = ex->sections;

  if (s->channel == cur->ch)
    {

      for (nrows = 0, i = 0; i < ex->nsections; i++, s++)            190
 {
   if ((cur->off + s->t[0]) < (1 - STAGE_LENGTH))
     warnx ("%s : causality failure at line %d, bad exemplars?  ",
    __FUNCTION__, __LINE__);
```

```
   section_adjust (s,
   ex->b + nrows,
   matrix_ref (X, 1, s->channel) + cur->off,
   ex->toff + i,
   ex->xoff + i);                                                    200

   ex->toff[i] += cur->off;
   dcopy (s->N, s->x, 1, A + nrows, 1);
   nrows += s->N;
 }

      ex->gain = ddot (nrows, A, 1, ex->b, 1) / ddot (nrows, A, 1, A, 1);
      daxpy (nrows, -(ex->gain), A, 1, ex->b, 1);
      ex->residual = dnrm2 (nrows, ex->b, 1) / nrows;
                                                                     210
      if (!finite (ex->gain) || ex->gain < ex->mingain ||
   ex->gain > ex->maxgain)
 ex->residual = HUGE;
    }
  else
    ex->residual = HUGE;
}

static void
exemplar_claim (exemplar_t * e,                                      220
 contact_t * cur,
 contact_t * contacts,
 int ncontacts)
{
  section_t *sp;
  contact_t *cp;
  int i, j, claimed, tau;

  CHECKPOINT ();
                                                                     230
  claimed = 0;
  sp = e->sections;

  for (i = 0; i < e->nsections; i++, sp++)
    {
      cp = contacts;
      tau = sp->contactoffset + cur->off;

      for (j = 0; j < ncontacts; j++, cp++)
 {                                                                   240
   if (cp->ch == sp->channel &&
       abs (cp->off - tau) < CLAIM_APERATURE &&
       cp->claim == 0)
     {
       cp->claim = 1;
       claimed++;
       break;
     }
 }
    }                                                                250

#ifdef MORE_DEBUG
```

```
  warnx ("%s : events claimed = %d/%d\n",
  __FUNCTION__, claimed, e->nsections);
#endif

  CHECKPOINT ();
}

exemplar_t *                                                          260
exemplar_firstbest (ll_t * exemplars)
{
  exemplar_t *rval, *ex;

  exemplars = ll_head (exemplars);
  rval = (exemplar_t *) ll_data (exemplars);

  while ((ex = (exemplar_t *) ll_foreach (&exemplars)))
    if (ex->residual < rval->residual)
      rval = ex;                                                      270

  if (rval->residual == HUGE)
    rval = NULL;

  return rval;
}

int
match (match_t * m, matrix_t * X)
{                                                                     280
  exemplar_t *ex, *hp;
  contact_t *cur;
  ll_t *ptr;
  int i;

  CHECKPOINT ();
#ifdef MORE_DEBUG
  warnx ("ne = %d\n", m->ncontacts);
#endif
                                                                      290
  ring_clear (&(m->hits));

  CHECKPOINT ();

  m->exemplars = ll_head (m->exemplars);

  CHECKPOINT ();

  cur = m->contacts;
                                                                      300
  CHECKPOINT ();

  for (i = 0; i < m->ncontacts && cur->off <= STAGE_LENGTH; i++, cur++)
    {
      if (cur->off < 1)
 errx (1, "%s : line %d : cur->off has illegal value",
       __FUNCTION__, __LINE__);

      CHECKPOINT ();
```

```
                                                                    310
     if (!cur->claim)
{

  CHECKPOINT ();

  ptr = m->exemplars;

  CHECKPOINT ();

  while ((ex = (exemplar_t *) ll_foreach (&ptr)))                   320
    {
      exemplar_submatch (ex, cur, X);
    }

  CHECKPOINT ();

  if ((ex = exemplar_firstbest (m->exemplars)))
    {

      CHECKPOINT ();                                                330

      if ((hp = (exemplar_t *) ring_put (&(m->hits))))
{

  CHECKPOINT ();

  memcpy (hp, ex, sizeof (exemplar_t));

#ifdef MORE_DEBUG
  warnx ("%s : i = %d : cur->off = %d : hp->toff[0] = %d/%d",       340
       __FUNCTION__, i, cur->off, hp->toff[0], ex->toff[0]);
#endif

  exemplar_claim (ex, cur, m->contacts, m->ncontacts);
}
      else
return 1;
    }
}
   }                                                                350

 CHECKPOINT ();
 return 0;
}

void
match_getevents (match_t * m, matrix_t * X)
{
  static double fx[STAGE_LENGTH + FILTER_LENGTH + 8];
  int i, j, M, N;                                                   360
  double *x;
  contact_t *cur;

  cur = m->contacts + m->ncontacts;

  M = FILTER_LENGTH + 1;
```

```
  N = STAGE_LENGTH + FILTER_LENGTH + 1;

  for (i = 1; i <= matrix_columns (X); i++)
    {                                                                    370
      x = matrix_ref (X, STAGE_LENGTH * 3, i);
      filter_s (__b__, M, x - FILTER_LENGTH, N, fx);

      for (j = 0; j <= STAGE_LENGTH; j++)
 fx[j] = fabs (fx[j] - x[j]);

      for (j = 1; j <= STAGE_LENGTH; j++)
 {
   if (((fx[j] > GETEVENTS_THRESHOLD) &&
        (fx[j - 1] <= GETEVENTS_THRESHOLD)) ||                           380
       ((fx[j] <= GETEVENTS_THRESHOLD) &&
        (fx[j - 1] > GETEVENTS_THRESHOLD)))
     {

       cur->ch = i;
       cur->claim = 0;
       cur->off = 3 * STAGE_LENGTH + j;
       cur++;
       assert ((m->ncontacts += 1) < N_EP);
       j += (GETEVENTS_SKIP - 1);                                        390
     }
 }
    }
}

void
match_shift (match_t * m)
{
  contact_t *ptr;
  int i, n;                                                              400

  ptr = m->contacts;
  for (i = 0, n = 0; i < m->ncontacts; i++, ptr++)
    if ((ptr->off -= STAGE_LENGTH) < 1)
      n++;

  if (n > 0)
    {
      m->ncontacts -= n;
      memmove (m->contacts,                                              410
        m->contacts + n,
        m->ncontacts * sizeof (contact_t));
    }
}

void
match_init (match_t * m, const char *prefix)
{
  static char strtmp[100];
  void *ta, *tb;                                                         420
  int i, ndd, row;

  matrix_t tz;
```

```
  ll_t *foo, *bar;

  m->nwork = N_RECORD * (4 + 2 * OFFSET_RANGE + 1 + MAX_SUBSECTIONS)
    + 2 * OFFSET_RANGE + 2;
  m->work = (double *) malloc (sizeof (double) * m->nwork);

  m->ncontacts = 0;                                                    430
  m->contacts = (contact_t *) malloc (sizeof (contact_t) * N_EP);

  if (m->contacts == NULL || m->work == NULL)
    errx (1, "%s : line %d : memory allocation failure",
   __FUNCTION__, __LINE__);

  ring_init (&(m->hits), N_HIT, sizeof (exemplar_t));

  CHECKPOINT ();
                                                                       440
  snprintf (strtmp, 100, "%s.dat", prefix);

  matrix_init (&tz, 1, 1);
  matrix_loadfile (&tz, strtmp);

  for (i = 1, ndd = 0; i <= matrix_rows (&tz); i++)
    if (matrix_element (&tz, i, 4) == 0.0)
      ndd++;

  CHECKPOINT ();                                                       450

  m->exemplars = ll_head (ll_makelist (ndd, sizeof (exemplar_t)));

  snprintf (strtmp, 100, "%s.desc", prefix);
  m->descriptions = ll_head (filetolist (strtmp, 100));

  row = 1;
  foo = m->exemplars;
  bar = m->descriptions;
  while ((ta = ll_foreach (&foo)) &&                                   460
  (tb = ll_foreach (&bar)))
    row = exemplar_init ((exemplar_t *) ta,
  (char *) tb,
  &tz,
  row);

  matrix_free (&tz);

  CHECKPOINT ();
                                                                       470
#ifdef DEBUG
  foo = m->exemplars;
  while ((ta = ll_foreach (&foo)))
    exemplar_manifest ((exemplar_t *) ta);
#endif
}

void
match_close (match_t * m)
{                                                                      480
```

```
  ll_t *ptr;
  void *this;

  if (m->contacts)
    {
      free (m->contacts);
      m->contacts = NULL;
    }

  if ((ptr = m->exemplars))                                          490
    {
      while ((this = ll_foreach (&ptr)))
 exemplar_free ((exemplar_t *) this);
      ll_free (m->exemplars);
      m->exemplars = NULL;
    }

  if ((ptr = m->descriptions))
    {
      while ((this = ll_foreach (&ptr)))                             500
 free (this);
      ll_free (m->descriptions);
      m->descriptions = NULL;
    }

  ring_free (&(m->hits));
}

void
exemplar_overlay (exemplar_t * hit,                                  510
   mgif_t * m,
   int ch)
{
  section_t *sp;
  float *yp, *xp;
  int i, n;

#ifdef MORE_DEBUG
  warnx ("%s : %s : nsections = %d",
  __FUNCTION__, hit->name, hit->nsections);                          520
#endif

  sp = hit->sections;
  for (i = 0, n = 0; i < hit->nsections; i++, sp++)
    if (sp->channel == ch)
      n += sp->N;

  mgif_setn (m, n);

  xp = mgif_xptr (m);                                                530
  yp = mgif_yptr (m);
  sp = hit->sections;

  for (i = 0; i < hit->nsections; i++, sp++)
    {
      if (sp->channel == ch)
 {
```

```
   dfaxpy (yp, sp->x, hit->gain, hit->xoff[i], sp->N);
   ifxpy (xp, sp->t, hit->toff[i], sp->N);
   yp += sp->N;                                                    540
   xp += sp->N;
 }
    }
}

void
exemplar_data_range (exemplar_t * hit,
      int *start,
      int *stop)
{                                                                  550
  section_t *s;
  int i, tmp;

  s = hit->sections;
  *start = hit->toff[0] + s->t[0];
  *stop = hit->toff[0] + s->t[s->N - 1];
  s++;

  for (i = 1; i < hit->nsections; i++, s++)
    {                                                              560

      tmp = hit->toff[i] + s->t[0];
      if (tmp < *start)
 *start = tmp;

      tmp = hit->toff[i] + s->t[s->N - 1];
      if (tmp > *stop)
 *stop = tmp;
    }
}                                                                  570
```

Program C.3 vsection.m.

The following Octave code is used to design exemplars using measured data.

```
#! /usr/local/bin/octave --silent                                  0
%-*-Octave-*-


clear;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% read exemplars from file fn
function [desc,ind,dat] = read_exemplars(fn)
   ind = [];
   desc = "";                                                      10
   dat  = [];

   eval(sprintf("load -force %s.ind", fn), "");
   eval(sprintf("load -force %s.dat", fn), "ind=[];");

   try
     [f,errmsg] = fopen(strcat(fn,".desc"),"r");
```

```
    if f == -1
      disp(errmsg);
      return;
    end;

    tmpstr = fgets(f,200);
    tmpstr = strrep(tmpstr,"\n","");
    desc(1,:) =tmpstr;

    for i = 2:length(ind)
      tmpstr = fgets(f,200);
      tmpstr = strrep(tmpstr,"\n","");
      desc = str2mat([desc; tmpstr]);
    end;

    fclose(f);
  catch
    ind = [];
    desc = "";
  end;
end;

%
% write exemplars to file fn.
function write_exemplars(fn,desc,ind,dat)
  if ind == []
     return;
  end;

  eval(sprintf("save -ascii %s.ind ind", fn), "error");
  eval(sprintf("save -ascii %s.dat dat", fn), "error");

  [f,errmsg] = fopen(strcat(fn,".desc"),"w");

  if f == -1
    disp(errmsg);
    return;
  end;
  for i = 1:length(ind)
    fputs(f,desc(i,:));
    fputs(f,"\n");
  end;

  fclose(f);
end;

%
% Mark events in the incoming stream y.
%
function [e,y] = event(y,b,thres)
  if columns(y) > rows(y)
    y = y';
  end;

  fy = filter(b,1,y);
  dy = y - fy;
```

```
  % find events -> make indicator variables.
  ify = find(abs(diff(abs(dy)>thres)) == 1);

  if ify == []                                                    80
    e = [];
    y = [];
    return;
  end;

  % cluster
  e = ify(1);
  le = ify(1);
  ee = 0;
                                                                  90
  for i=2:length(ify)
    if ify(i) > length(y)-100
 y = y(ify(i):length(y));
 ee = 1;
 break;
    end;

    if ify(i) > le+50
      e = [e ify(i)];
      le = ify(i);                                               100
    end;
  end;

  if ee == 0
    y = [];
  end;
end;


%                                                                 110
% this function makes the data entry for a single exemplar.
%
function [data] = makedata(X,Y)
    global N_RECORD;

    % make a new X, NX, consisting only of sections
    % with non-zero length and weight.
    NX = X(:, find( X(4,:) .* X(3,:) != 0));

    % sort so that the sub-events are arranged /wrt to offset     120
    [S,I] = sort(NX(2,:));

    % physically rearrange according to the order.
    NX = NX(:,I);

    % initialize data
    data = [];
    curr = 1;

    firstoff = NX(2,1);                                           130

    % for each channel...
```

```
    for q = 1:columns(Y)
      I = find(NX(1,:) == q);

      % make records, cat into data.
      for k = 1:length(I)
 ch   = NX(1,I(k));
 st   = NX(2,I(k))+NX(5,I(k))-NX(2,1);
 ste  = NX(2,I(k))-NX(2,1);                                          140
 N    = NX(3,I(k));
 last = columns(NX)-curr;
 curr = curr + 1;

 % where the record starts in the data stream.
 itmp = NX(2,I(k))+NX(5,I(k));

 % make a full-sized row record of the exemplar.
 record = [ch st N last ste Y(itmp:itmp+N, ch)'];
 record = [record zeros(1,N_RECORD - length(record))];               150

 % record = [ch; st; N; last; Y(NX(2,I(k)):NX(2,I(k))+N, ch)];
 % data = [data; record; zeros(N_RECORD-length(record),1)];


 % pack records in rows of the data matrix.
 data = [data; record];
      end;
    end;
end;                                                                 160




%
% This function has to compute where all the exemplars should start.
% Then, it should ask the user how long each exemplar should be...
%
function [desc,data] = edit(vdata,ch)
                                                                     170

  Y = vdata;

  X = [];

  global N_RECORD;
  global FILTER_LENGTH;
  global FILTER_CUTOFF;
  global GETEVENTS_THRESHOLD;
  thres = GETEVENTS_THRESHOLD;                                       180

  b = fir1(FILTER_LENGTH,FILTER_CUTOFF);

  try
  for i = 1:columns(Y)
    % each channel

     [ev,z] = event(Y(:,i),b,thres);
```

```
   if ev != []                                                                    190
     Xa = zeros(length(ev),5+2*N_RECORD);
     Xa(:,1) = i;   % channel
     Xa(:,3) = 5.0; % weight
     Xa(:,4) = 99;  % count must be fixed later
     for k=1:length(ev)
t = ev(k):min([50+ev(k) rows(Y)]);
Xa(k,2)=length(t);
Xa(k,5)=ev(k)-100; % event offset
Xa(k,6+N_RECORD:2*N_RECORD+5)=[(t-100-Xa(k,5)) zeros(1,N_RECORD - length(t))];
Xa(k,6:N_RECORD+5)=[Y(t,i)' zeros(1,N_RECORD - length(t))];                       200
     end;
     X = [X; Xa];
   end;
  end;
catch
   disp('internal error 1');
   pause(2);
   return;
end;
if X == []                                                                        210
   data = [];
   desc = [];
   disp('No sections to add...');
   pause(1);
   return;
end;

% initialization...
desc = [];
data = [];                                                                        220
plot_ch = -1;
plot_sect = -1;

sect = 1;

try

  while 1
    ch = X(sect,1);
                                                                                  230
    % display the whole transient, make sure to get the right channel
    % should display vsections here too?



    if plot_ch != ch || plot_sect != sect
figure(1);
title(sprintf("Add/Edit: Marked-up transient on channel %d", ch));
try
% find all the v-sections on this plot.                                          240
      q = find(X(:,1) == ch);
ty = [];
tx = [];
rx = [];
ry = [];
```

```
for i = 1:length(q)
         if X(q(i),3) != 0 % weight != 0 then
           if q(i) != sect
      ty = [ty; X(q(i),6:5+X(q(i),2))];                                          250
      tx = [tx; ((X(q(i),(6+N_RECORD:5+N_RECORD+X(q(i),2)))).+X(q(i),5))];
           else
      ry = X(q(i),6:5+X(q(i),2));
      rx = ((X(q(i),6+N_RECORD:5+N_RECORD+X(q(i),2))).+X(q(i),5));
    end;
  end;
end;

plot(-99:(rows(Y)-100),Y(:,ch),tx,ty,'ro',rx,ry,'b+');
catch                                                                            260
  disp('New Error');
end;
clear tx;
clear ty;
clear rx;
clear ry;
plot_ch = ch;
     end;

                                                                                 270

     % display the current section
     if plot_sect != sect
figure(2);
title(sprintf("Add/Edit: Section %d of %d", sect, rows(X)));

i0 = max([1 X(sect,6+N_RECORD)+X(sect,5)+100-50]);
i1 = min([X(sect,5+N_RECORD+X(sect,2))+X(sect,5)+100+50 rows(Y)]);
% i2 = X(sect,6+N_RECORD)+X(sect,5);
% i3 = min([rows(Y) X(sect,5)+X(sect,5+N_RECORD + X(sect,2))]);                  280

plot(i0-100:i1-100,Y(i0:i1,ch),
       ((X(sect,6+N_RECORD:5+N_RECORD+X(sect,2))).+X(sect,5)),
       X(sect,6:5+X(sect,2)),'b+');
clg;
plot_sect = sect;
     end;

     clc;
                                                                                 290
     printf("Current section length = %d, weight = %f\n\n", X(sect,2),X(sect,3));


     c = menu("Edit/Add Exemplar:","Previous section","Next section",
              "Change Examplar",
       "Change weight", "Done, add to exemplars",
              "Done, do not add");

     if c == 1
sect = max([1 sect-1]);                                                          300
     end;
     if c == 2
sect=min([rows(X) sect+1]);
```

```
     end;
     if c == 6
data = [];
desc = "";
       return;
     end;
                                                                          310
     if c == 5
% build up data[] here...
% remove zero weight transients
Z=zeros(length(find(X(:,3))),5+2*N_RECORD);
n=1;
len=length(find(X(:,3)));
for k=1:length(X(:,3))
  if(X(k,3)!=0)
    Z(n,:)=X(k,:);
    Z(n,4)=len-n;                                                         320
    n=n+1;
  end;
end;
data = [data; Z];
desc = input("Enter description : ", []);
return;
     end;

     % change weight of exemplar.
     if c == 4                                                            330
X(sect,3) = min([10.0 max( [0 input("Weight = ")])]);
     end;

     % change the exemplar
     if c == 3
str="";
str=input("Enter Range: ",str);
try
  str=strcat(str,";");
  range=eval(str);                                                        340
  if (length(range)>N_RECORD)
    disp('Error: Range must contain less than N_RECORD points');
  end;
  if (length(range)<5)
    disp('Error: Range must contain at least 5 points');
  end;
  for i=2:length(range)
    if(range(i)<=range(i-1))
      disp('Error: Range must be strictly increasing');
    end;                                                                  350
  end;
  X(sect,2)=length(range);
  X(sect,6+N_RECORD:5+2*N_RECORD)=[range zeros(1,N_RECORD-length(range))];
  X(sect,6:5+N_RECORD)=[Y(range.+100.+X(sect,5),ch)' zeros(1,N_RECORD-length(range))];
catch
  disp('Error: Illegal Range.');
end;
plot_sect=-1;
     end;
   end;                                                                   360
```

```
  catch
    disp('Internal Error 2');
    pause(2);
    return;
  end;
end;


% let the user interact with the current exemplars.
function [desc,ind,dat] = scan(desc,ind,dat,N)                    370

global N_RECORD;

  if ind == []
    disp('No exemplars to scan...');
    pause(2);
    return;
  end;

  plot_ex = -1;                                                   380
  ex = 1;

  ch = 1;

  while 1
     if plot_ex != ex
 figure(1);

 tx = [];
 ty = [];                                                         390

 k = 0;

 while 1
   % record = dat(ind(ex)+256*k:ind(ex)+256*(k+1)-1);

   record = dat(ind(ex)+k,:);

   if record(1) == ch
     tx = [tx record(6+N_RECORD:5+N_RECORD+record(2))];           400
     ty = [ty ; record(6:5+record(2))];
   end;

   if record(4) <= 1
     break;
   end;
   k = k + 1;
 end;

 disp(desc(ex,:));                                                410

 strmedo = sprintf('Scan exemplars : %s Channel %d',desc(ex,:), ch);
 title(strmedo);

 if tx == []
  tx = 0;
  ty = 0;
```

```
end;

plot(tx,ty,'b+');                                                    420

       plot_ex = ex;
    end;

    clc;
    printf("Current exemplar = %d \n Description: %s\n", ex, desc(ex,:));
    c = menu("Scan exemplars", "Previous exemplar", "Next exemplar",
"Delete current exemplar", "New description",
"Previous channel", "Next channel", "Exit");
                                                                     430
    if c == 1
ex = max([1 ex-1]);
    end;
    if c == 2
       ex = min([ex+1 length(ind)]);
    end;
    if c == 3

    end;
    if c == 4                                                        440
clear newd;
       newd = input("Description : ",[]);

tmpa = desc(1:ex-1,:);
tmpb = desc(ex+1:rows(desc),:);

desc = [tmpa; newd; tmpb];

clear tmpa;
clear tmpb;                                                          450
    end;
    % previous channel
    if c == 5
ch = max([1 ch-1]);
plot_ex = -1;
    end;
    % next channel
    if c == 6
ch = min([N ch+1]);
plot_ex = -1;                                                        460
    end;
    % return.
    if c == 7
       return;
    end;
  end;
end;

%
% Provide filtering options for the user.                            470
%
function [x,ch] = filts(x,ch)
  plot_lambda = -1;
  plot_ch = -1;
```

```
 lambda = .1;
 fx = x;

 try
 while 1                                                                    480
    % new lambda?   make sure fx is up to date.
    if plot_lambda != lambda
b = fir1(128,lambda);
for i = 1:columns(x)
  fx(:,i) = filtfilt(b,1,x(:,i));
       end;
    end;

    % update the plots.
    if plot_ch != ch || plot_lambda != lambda                               490
       if plot_ch != ch
   figure(1);
          title(sprintf("Channel %d",ch));
          plot(x(:,ch));
          plot_ch = ch;
end;

       figure(2);
title(sprintf("Filtered data, Channel %d, cutoff = %f",ch,lambda));
plot(fx(:,ch));                                                             500
plot_lambda = lambda;
    end;

    % present menu
    clc
    c = menu("Filter menu", "Cutoff", "Save and return", "return");

    if c == 1
lambda = input("New cutoff : ");
lambda = max([.01 lambda]);                                                 510
lambda = min([.99 lambda]);
    end;

    if c == 2
x = fx;
return;
    end;
    if c == 3
return;
    end;                                                                    520
 end;
 catch
    disp('Internal error in filts routine.');
    pause(3);
    return;
 end;
end;

%% MAIN PROGRAM %%
                                                                            530
% Load the file, if we can find it.
```

```
try
  eval(sprintf("load %s/%s", pwd,argv), "disp('File error'); return;");
catch
   disp('Usage : \n');
   disp('vsection <datafile> \n');
   exit;
end;

% make sure the file is reasonable.                                        540
if exist('vdata') != 1
  disp('File does not appear to be in the correct format.');
  exit;
end;

% cool.
if columns(vdata) != 1 && columns(vdata) != 8
  disp('invalid data : must have either 1 or 8 columns');
  exit;
end;                                                                       550

try
nilmconst
catch
  disp('Could not read nilmconst.m');
  exit;
end;



                                                                           560
% initialization
ind = [];
desc = "";
dat  = [];

plotch = -1;
plottrans = -1;
ch = 1;
trans = 1;
translength = 1000;         % is this right?                               570

if translength*floor(rows(vdata) / translength) != rows(vdata)
  disp('Number of rows in data not correct');
  exit;
end;

%%try

while(1)
  clc;                                                                     580

  % update the plot.
  if plotch != ch || plottrans != trans
     figure(1);
     title(sprintf("Channel %d",ch));
     plot(vdata((trans-1)*translength+1:trans*translength, ch));
     plotch = ch;
     plottrans = trans;
```

```
end;
                                                                    590
choice = menu("Main:", "Previous transient", "Next transient",
        "Add to exemplars", "Filters",
"Change channel",
                        "Scan/delete exemplars",
              "Write exemplars to disk", "Read exemplars from disk",
        "Quit");
% previous
if( choice == 1)
  trans = max([trans-1 1]);
end;                                                                600

% next
if(choice == 2)
   trans = min([trans+1 floor(rows(vdata)/translength)]);
end;

% edit/add
if( choice == 3 )
   [a,b] = edit(vdata((trans-1)*translength+1:trans*translength,:),ch);
   % non-trivial return from edit,                                  610
   % concatenate.
   if b != []
desc = [desc; a];
ind = [ind; rows(dat)+1];
% b is a bunch of rows...
dat = [dat; b];
   end;

   plotch = -1;
end;                                                                620

% filter
if(choice == 4)
   i0 = (trans-1)*translength+1;
   i1 = trans*translength;
   [vdata(i0:i1,ch),ch] = filts(vdata(i0:i1,ch),ch);
   plotch = -1; % force update
end;

% change channel                                                    630
if( choice == 5)
  ch = max([1 min([input("New channel number : ") columns(vdata)])]);
end;

% scan/delete exemplars
% Have a look at the exemplars we've got...
if (choice == 6)
   [desc,ind,dat] = scan(desc,ind,dat,columns(vdata));
   plotch = -1;     % force update
end;                                                                640

% write exemplars to disk
if( choice == 7)
    q = input("Name [exemplars] : ", []);
    if q == ""
```

```
 q = "exemplars";
      end;
      write_exemplars(q,desc,ind,dat);
  end;
                                                                          650
  % read exemplars from disk
  if(choice == 8)
    q = input("Name [exemplars] : ", []);
    if q == ""
      q = "exemplars";
    end;
    [desc,ind,dat] = read_exemplars(q);

    if(dat == [])
      disp('Error reading exemplars or file empty...');                   660
      sleep(2);
    end;
  end;

  % leave
  if(choice == 9)
    exit;
  end;
end;
                                                                          670
%catch
%  disp('Internal error 0');
%  pause(2);
%  return;
%end;
```

C.2 Identification and simulation programs

Program C.4 *Generic simulation program,* `sim.c`.

```
/* generated Fri Dec 10 15:39:53 1999 on nitro */                          0

/*

                    Written by Steven R. Shaw, 1999

              Copyright (c) 1999 MASSACHUSETTS INSTITUTE
                  OF TECHNOLOGY. All rights reserved.

*/
                                                                          10


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <err.h>
```

```
#include <math.h>

#include "split.h"                                                    20
#include "model.h"

#define MAXPARM 20

void MODELNAME (model_t * m, const char *options, double *parms);

static void
dump (matrix_t * t, matrix_t * o)
{
  int i, j;                                                           30

  for (i = 1; i <= matrix_columns (o); i++, printf ("\n"))
    {
      if (t)
 printf ("%.3e ", matrix_element (t, 1, i));
      for (j = 1; j <= matrix_rows (o); j++)
 printf ("%.3e ", matrix_element (o, j, i));
    }
}
                                                                      40
static void
usage (char **argv)
{
  printf ("usage : \n");
  printf ("%s [-o \"option-string\"] [-p \"mu0 .. muN\"] [-d]\n", *argv
    );
  printf ("-d prints the default options and parameters\n");
}

static void                                                           50
commandline (model_t * m,
      int argc,
      char **argv,
      matrix_t * t,
      matrix_t * inputs,
      matrix_t * outputs,
      int *timeaxis)
{
  static double mu[MAXPARM];
  static char opts[200];                                              60
  static char parms[200];
  static char inputfile[200];
  extern char *optarg;
  char *each[MAXPARM];
  int ch, i;

  *timeaxis = 0;

  strcpy (opts, "");
  strcpy (parms, "");                                                 70
  strcpy (inputfile, "");

  while ((ch = getopt (argc, argv, "tdo:p:i:")) != EOF)
    {
```

```
     switch ((char) ch)
{
case 'd':
  MODELNAME (m, opts, NULL);
  model_about (m);
  model_free (m);                                                    80
  exit (0);
  break;
case 'o':
  strcpy (opts, optarg);
  break;
case 'p':
  strcpy (parms, optarg);
  break;
case 'i':
  strcpy (inputfile, optarg);                                        90
  break;
case 't':
  *timeaxis = 1;
  break;
case ':':
  errx (1, "%s : flag missing required argument at line %d",
__FUNCTION__, __LINE__);
  break;
case '?  ':
default:                                                            100
  usage (argv);
  exit (1);
  break;
}
   }

 if (*parms)
   {
     split (each, parms, " :\t", MAXPARM);
     for (ch = 0; ch < MAXPARM; ch++)                               110
if (each[ch])
  sscanf (each[ch], "%lf", mu + ch);
     MODELNAME (m, opts, mu);
   }
 else
   MODELNAME (m, opts, NULL);

 if (*inputfile)
   {
     matrix_t hold;                                                 120
     matrix_init (&hold, 1, 1);
     matrix_loadfile (&hold, inputfile);
     matrix_duptrans (inputs, &hold);
     matrix_free (&hold);
     matrix_rowrange (t, inputs, 1, 1);
   }
 else
   {
     matrix_init (t, 1, 1000);
     for (i = 1; i <= matrix_cols (t); i++)                         130
*(matrix_ref (t, 1, i)) = (double) i *0.0004;
```

```
      matrix_clone (inputs, t);
    }

  matrix_init (outputs, model_noutputs (m), matrix_columns (t));
}

int
main (int argc, char **argv)
{                                                                    140
  matrix_t T, outputs, inputs;
  model_t m;
  int timeflag;

  commandline (&m, argc, argv, &T, &inputs, &outputs, &timeflag);
  model_simulate (&m, &outputs, &inputs, &T);

  if (timeflag)
    dump (&T, &outputs);
  else                                                               150
    dump (NULL, &outputs);

  model_free (&m);

  return 0;
}
```

Program C.5 *Generic identification program,* `id.c`.

```
/* generated Fri Dec 10 15:39:51 1999 on nitro */                     0

/*

                    Written by Steven R. Shaw, 1999

              Copyright (c) 1999 MASSACHUSETTS INSTITUTE
                  OF TECHNOLOGY. All rights reserved.

*/
                                                                     10


#define METHOD_TWO

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <err.h>
#include <math.h>                                                    20

#include "split.h"
#include "model.h"
#include "method.h"

#ifdef METHOD_TWO
#include "objective.h"
#include "method2.h"
```

```
#endif

#define MAXPARM 20

void MODELNAME (model_t * m, const char *options, double *parms);

static void
dump (matrix_t * o)
{
  int i, j;

  for (i = 1; i <= matrix_columns (o); i++, printf ("\n"))
    for (j = 1; j <= matrix_rows (o); j++)
      printf ("%.3e ", matrix_element (o, j, i));
}

static void
usage (char **argv)
{
  printf ("usage : \n");
  printf ("%s -i inputfile [-o \"option-string\"]", *argv);
  printf ("[-p \"mu0 .. muN\"] [-d]\n");
  printf ("-d prints the default options and parameters\n");
}

void
about (const char *name, matrix_t * m)
{
  printf ("matrix %s : lda = %d rows = %d columns = %d \n",
   name, matrix_lda (m), matrix_rows (m), matrix_columns (m));
}

static void
commandline (model_t * m,
      int argc,
      char **argv,
      matrix_t * t,
      matrix_t * inputs,
      matrix_t * observations,
      double *thres,
      double *reg,
      char *reportopts,
      int *mflag)
{
  static double mu[MAXPARM];
  static char opts[200];
  static char parms[200];
  static char inputfile[200];
  extern char *optarg;

  matrix_t hold;

  char *each[MAXPARM];
  int ch, i;
  int quiet;

  *thres = 1.0;
```

```
 *reg = .4;

 strcpy (opts, "");
 strcpy (parms, "");
 strcpy (inputfile, "");                                                    90

 while ((ch = getopt (argc, argv, "mdo:p:i:t:r:z:")) != EOF)
   {
     switch ((char) ch)
{
case 'd':
  MODELNAME (m, opts, NULL);
  model_about (m);
  model_free (m);
  exit (0);                                                                100
  break;

case 'o':
  strcpy (opts, optarg);
  break;

case 'm':
  *mflag = 1;
  break;
                                                                           110
case 'p':
  strcpy (parms, optarg);
  break;

case 'i':
  strcpy (inputfile, optarg);
  break;

case 't':
  sscanf (optarg, "%lf", thres);                                           120
  break;

case 'z':
  sscanf (optarg, "%lf", reg);
  break;

case 'r':
  strcpy (reportopts, optarg);
  break;
                                                                           130
case ':':
  errx (1, "%s : flag missing required argument at line %d",
__FUNCTION__, __LINE__);
  break;
case '?  ':
default:
  usage (argv);
  exit (1);
  break;
}                                                                          140
   }
```

```
  if (*parms)
    {
      split (each, parms, " :\t", MAXPARM);
      for (ch = 0; ch < MAXPARM; ch++)
 if (each[ch])
   sscanf (each[ch], "%lf", mu + ch);
      MODELNAME (m, opts, mu);
    }                                                                    150
  else
    MODELNAME (m, opts, NULL);

  if (*inputfile)
    {
      matrix_init (&hold, 1, 1);
      matrix_loadfile (&hold, inputfile);

      matrix_duptrans (inputs, &hold);
      matrix_free (&hold);                                               160
      matrix_rowrange (t, inputs, 1, 1);
      matrix_createref (observations,
 matrix_ref (inputs, 1 + model_ninputs (m), 1),
 matrix_lda (inputs),
 model_noutputs (m),
 matrix_columns (inputs));
    }
  else
    errx (1, "input file required");
}                                                                        170

int
main (int argc, char **argv)
{
  matrix_t T, outputs, inputs, observations;
  model_t m;
  double thres, reg;
  char reportoptions[200];
  int mflag = 0;
                                                                         180
  commandline (&m, argc, argv, &T, &inputs,
        &outputs, &thres, &reg,
        reportoptions, &mflag);

  matrix_dup (&observations, &outputs);

  if (mflag == 0)
    method (&m, &T, &observations, &inputs, reg, 50, 200, thres);
  else
    {                                                                    190
#ifndef METHOD_TWO
      warnx ("warning : not compiled with method two, using method one");
      method (&m, &T, &observations, &inputs, 4.0, 50, 200, thres);
#else
      objective_t objective_data;
      matrix_t mu0;
      matrix_t e;
      int info;
```

```
      matrix_dup (&mu0, model_parameters (&m));                          200
      matrix_init (&e,
    matrix_rows (&observations) *
    matrix_columns (&observations) +
    matrix_rows (&mu0), 1);

      objective_data.model = &m;
      objective_data.inputs = &inputs;
      objective_data.time = &T;
      objective_data.observations = &observations;
      objective_data.mu0 = &mu0;                                         210
      objective_data.gamma = 1.0;

      info = methodtwo (objective, (void *) &objective_data, &e,
 model_parameters (objective_data.model), 30,
 30, 1e-6, 1e-6, 500);

      matrix_free (&e);
      matrix_free (&mu0);

      if (info == 0)                                                     220
 warnx ("MaxIter exceeded");

      if (info == -1)
 warnx (
  "data set was not completed, improvements to error less than tol");

      if (info == -2)
 warnx ("data set was not completed, error less than tol");
#endif
    }                                                                    230

  if (!strcasecmp (reportoptions, "all") ||
      !strcasecmp (reportoptions, "parms"))
    dump (model_parameters (&m));

  if (!strcasecmp (reportoptions, "all") ||
      !strcasecmp (reportoptions, "outputs"))
    {
      model_simulate (&m, &outputs, &inputs, &T);
      dump (&outputs);                                                   240
    }

  model_free (&m);

  return 0;
}
```

Program C.6 *Method of Chapter 3.*

```
/* generated Fri Dec 10 15:39:52 1999 on nitro */                        0

/*

                Written by Steven R. Shaw, 1999

            Copyright (c) 1999 MASSACHUSETTS INSTITUTE
```

```
                OF TECHNOLOGY. All rights reserved.

*/                                                                          10


#include <stdio.h>
#include <stdlib.h>
#include <err.h>

#include "tools.h"
#include "method.h"
#include "minpack.h"
                                                                            20
#define VERB 0
#define MINDATA 20

typedef struct
  {
    double xtol, ftol, gtol, epsfcn;
    double *diag, *qtf, *wa1, *wa2, *wa3;
    int *ipvt;
    double *fvec, *wa4;
    double *fjac;                                                           30
    int m, n, ldfjac, maxfev, nfev, mode, nprint, info;
    double factor;
  }
lmdif_t;

typedef struct
  {
    lmdif_t l;

    model_t *model;                                                         40
    matrix_t *musave;

    matrix_t *inputs;
    matrix_t *t;
    matrix_t *observations;

    double regp;

    long nerreval;
    int natomic, ndata;                                                     50
  }
method_t;

static method_t *z;

static __inline__ void
fastblas_xmy (double *x, double *y, int n)
{
  int i;
                                                                            60
  for (i = 0; i < n - 4; i += 4)
    {
      *x -= *y;
```

```
      x++;
      y++;
      *x -= *y;
      x++;
      y++;
      *x -= *y;
      x++;
      y++;
      *x -= *y;
      x++;
      y++;
    }

  for (; i < n; i++)
    {
      *x -= *y;
      x++;
      y++;
    }
}

void
lmdif_free (lmdif_t * l)
{
  if (l->ipvt)
    {
      free (l->ipvt);
      l->ipvt = NULL;
    }

  if (l->fjac)
    {
      free (l->fjac);
      l->fjac = NULL;
    }

  if (l->diag)
    {
      free (l->diag);
      l->diag = NULL;
    }
}

static void
lmdif_init (lmdif_t * l, int m, int n, double tol)
{
  l->xtol = tol;
  l->ftol = tol;
  l->gtol = 0;
  l->epsfcn = tol;
  l->m = m;
  l->n = n;
  l->ldfjac = m;
  l->maxfev = 500;
  l->nfev = 0;
  l->mode = 1;
  l->nprint = 1;
```

```
  l->factor = 10;

  l->ipvt = (int *) malloc (sizeof (int) * n);
  l->diag = (double *) malloc (sizeof (double) * (n * 5 + m * 2));
  l->fjac = (double *) malloc (sizeof (double) * l->ldfjac * n);

  if (l->ipvt == NULL || l->diag == NULL ||
      l->fjac == NULL)
    {
      lmdif_free (l);                                                    130
      errx (1, "%s : memory allocation", __FUNCTION__);
    }

  l->qtf = l->diag + n;
  l->wa1 = l->qtf + n;
  l->wa2 = l->wa1 + n;
  l->wa3 = l->wa2 + n;
  l->fvec = l->wa3 + n;
  l->wa4 = l->fvec + m;
                                                                         140
}

static void
lmdif_error_function (int *M, int *N, double *mu, double *
       err, int *iflag)
{
  static matrix_t errors;
  static matrix_t tsized;
  int i, Nsim = (*M) - model_nparameters (z->model);
  double *ptr;                                                           150

  z->nerreval += Nsim;

  matrix_createref (&errors,
     err,
     matrix_rows (z->observations),
     matrix_rows (z->observations),
     matrix_cols (z->observations));

  matrix_createref (&tsized,                                             160
     matrix_data (z->t),
     matrix_lda (z->t),
     matrix_rows (z->t),
     Nsim / z->natomic);

  model_setparms (z->model, mu);

  if (model_simulate (z->model,
       &errors,
       z->inputs,                                                        170
       &tsized) == 0)
    {

      fastblas_xmy (err, matrix_data (z->observations), Nsim);

      ptr = matrix_data (z->musave);
      for (i = 0; i < (*M) - Nsim; i++)
```

```
 err[i + Nsim] = z->regp * (mu[i] - ptr[i]) / ptr[i];
    }
  else                                                                    180
    *iflag = -1;
}

static int
lmdif (lmdif_t * p, matrix_t * mu, int N)
{
  lmdif_ (lmdif_error_function, &N, &(p->n), matrix_data (mu), p->fvec, &(
        p->ftol), &(p->xtol),
   &(p->gtol), &(p->maxfev), &(p->epsfcn), p->diag, &(p->mode), &(p->
     factor),                                                             190
 &(p->nprint), &(p->info), &(p->nfev), p->fjac, &(p->ldfjac), p->ipvt
   ,p->qtf,
   p->wa1, p->wa2, p->wa3, p->wa4);

  if (p->info == 0)
    errx (1, "%s : improper parameters", __FUNCTION__);

  return p->info;
}
                                                                          200
static double
lmdif_scale (lmdif_t * p, matrix_t * mu)
{
  int iflag, i;
  double sum;

  iflag = 1;
  lmdif_error_function (&(p->m), &(p->n), matrix_data (mu), p->fvec, &iflag)
    ;
                                                                          210
  sum = dnrm2 (p->m, p->fvec, 1);

  iflag = 2;
  fdjac2_ (lmdif_error_function, &(p->m), &(p->n), matrix_data (mu),
    p->fvec, p->fjac, &(p->ldfjac), &iflag, &(p->epsfcn), p->wa4);

  for (i = 0; i < matrix_rows (mu); i++)
    {
      p->diag[i] = dnrm2 (p->m, p->fjac + (p->ldfjac) * i, 1);
      if (p->diag[i] == 0.0)                                              220
 p->diag[i] = 1.0;
    }

  return sum;
}

static int
method_body (int start, int inc, double thres)
{
  matrix_t save;                                                          230
  double crit;
  int n, rval, qval, m;

  rval = 0;
```

```
#if VERB == 1
  warnx ("start = %d, inc = %d, thres = %.3e", start, inc, thres);
  matrix_write (model_parameters (z->model), stderr, "%.3e ");
#endif
                                                                        240
  if (start >= MINDATA)
    {
      matrix_clone (&save, model_parameters (z->model));

      n = start;

      do
{
   m = n * (z->natomic);
                                                                        250
   if (n == z->ndata)
     {
       z->l.xtol = 1e-6;
       z->l.gtol = 1e-6;
       z->l.ftol = 1e-6;
       z->l.epsfcn = 1e-6;
     }
   else
     {
       z->l.xtol = 1e-3;                                                260
       z->l.gtol = 1e-3;
       z->l.ftol = 1e-3;
       z->l.epsfcn = 1e-6;
     }

   qval = lmdif (&(z->l),
model_parameters (z->model),
m + model_nparameters (z->model));

   if (qval < 0)                                                        270
     {
       warnx ("integration failure");
       rval = 2;
       break;
     }

   crit = dnrm2 (m, z->l.fvec, 1) / sqrt (m);

   if (crit > thres)
     {                                                                  280
       dswap (matrix_length (model_parameters (z->model)),
      matrix_data (model_parameters (z->model)), 1, matrix_data (&save), 1)
;

       rval = method_body (start / 2, inc / 2, thres);

       if (rval == 0)
{
  dswap (matrix_length (model_parameters (z->model)),
 matrix_data (model_parameters (z->model)), 1, matrix_data (&save), 1);   290
  thres *= 1.5;
```

```
 }
     }
   else
     {
#if VERB == 1
       warnx ("n = %d", n);
#endif

       if (n == z->ndata)                                                    300
 {
#if VERB == 1
   matrix_write (model_parameters (z->model), stderr, "%.3e ");
#endif
   rval = 1;
 }
       else if ((n += inc) > z->ndata)
 n = z->ndata;
     }
                                                                           310
 }
      while (rval == 0);

      matrix_free (&save);
    }

  return rval;
}

void                                                                       320
method (model_t * m,
 matrix_t * t,
 matrix_t * observations,
 matrix_t * inputs,
 double regp,
 int start,
 int inc,
 double thres)
{
  method_t sz;                                                             330
  matrix_t musave;
  int i;

  z = &sz;

  if (start > matrix_columns (t) || start > matrix_columns (inputs))
    errx (1, "bad args in function \'%s\', line %d", __FUNCTION__,
   __LINE__);

  lmdif_init (&(z->l),                                                     340
       matrix_rows (observations) * matrix_cols (observations) +
       model_nparameters (m),
       model_nparameters (m),
       1e-6);

  z->regp = 0;
  z->nerreval = 0;
  z->model = m;
```

```
  z->inputs = inputs;
  z->t = t;                                                              350
  z->observations = observations;

  matrix_dup (&(musave), model_parameters (m));
  z->musave = &musave;
  z->natomic = matrix_rows (observations);
  z->ndata = matrix_cols (observations);

  if (!matrix_contiguous (observations))
    errx (1,
   "%s : observations matrix is not contiguous - use matrix_dup",        360
   __FUNCTION__);

  z->regp = regp * lmdif_scale (&(z->l), model_parameters (z->model));

  z->l.mode = 2;

  method_body (start, inc, thres);

  lmdif_free (&(z->l));
  matrix_free (&musave);                                                 370
}
```

C.3 Models

Program C.7 induction.c *(§ 4.3.2)*

```
/* generated Fri Dec 10 15:39:52 1999 on nitro */                          0

/*

                  Written by Steven R. Shaw, 1999

             Copyright (c) 1999 MASSACHUSETTS INSTITUTE
                 OF TECHNOLOGY. All rights reserved.

*/
                                                                          10


#include <string.h>
#include <err.h>
#include <math.h>

#include "model.h"
#include "induction.h"

#ifdef DEBUG                                                              20
#define CHECKPOINT() warnx("function %s, line %d", __FUNCTION__,
__LINE__);
#else
#define CHECKPOINT()
#endif
```

```
#define RS(x) ((x)[0])
#define RR(x) ((x)[1])
#define LM(x) ((x)[2])
#define LL(x) ((x)[3])                                                    30
#define K(x)  ((x)[4])
#define B(x)  ((x)[5])

static void
g0 (double *x, double *u, double *y, double *mu)
{
  double w0 = 2 * M_PI * 60;
  double phi;
  double Lrr, Lm, Grr, Gm, ids, iqs;
                                                                          40
  Lrr = LM (mu) + LL (mu);
  Lm = LM (mu);

  Grr = Lrr / (Lrr * Lrr - Lm * Lm);
  Gm = Lm / (Lrr * Lrr - Lm * Lm);

  ids = Grr * x[0] - Gm * x[2];
  iqs = Grr * x[1] - Gm * x[3];

  phi = w0 * u[0];                                                        50

  y[0] = iqs * cos (phi) + ids * sin (phi);
}

static void
g1 (double *x, double *u, double *y, double *mu)
{
  const double w0 = 2 * M_PI * 60;
  double Lrr, Lm, Grr, Gm, ids, iqs;
  double phi, ang;                                                        60

  Lrr = LM (mu) + LL (mu);
  Lm = LM (mu);

  Grr = Lrr / (Lrr * Lrr - Lm * Lm);
  Gm = Lm / (Lrr * Lrr - Lm * Lm);

  ids = Grr * x[0] - Gm * x[2];
  iqs = Grr * x[1] - Gm * x[3];
                                                                          70
  phi = w0 * u[0];
  ang = 2.0 * M_PI / 3.0;

  y[0] = ids * cos (phi) + iqs * sin (phi);
  y[1] = ids * cos (phi - ang) + iqs * sin (phi - ang);
  y[2] = ids * cos (phi + ang) + iqs * sin (phi + ang);
}

static void
g2 (double *x, double *u, double *y, double *mu)                          80
{
  const double w0 = 2 * M_PI * 60;
```

```
  // Compute the currents.
  double Lrr, Lm, Grr, Gm;

  Lrr = LM (mu) + LL (mu);
  Lm = LM (mu);

  Grr = Lrr / (Lrr * Lrr - Lm * Lm);                                    90
  Gm = Lm / (Lrr * Lrr - Lm * Lm);

  y[0] = Grr * x[0] - Gm * x[2];
  y[1] = Grr * x[1] - Gm * x[3];
}

static void
g3 (double *x, double *u, double *y, double *mu)
{
  const double w0 = 2 * M_PI * 60;                                     100
  double Lrr, Lm, Grr, Gm;

  Lrr = LM (mu) + LL (mu);
  Lm = LM (mu);

  Grr = Lrr / (Lrr * Lrr - Lm * Lm);
  Gm = Lm / (Lrr * Lrr - Lm * Lm);

  y[0] = Grr * x[0] - Gm * x[2];
}                                                                      110

static void
f (int *neq, double *tim, double *x, double *xdot)
{
  double *mu = ((struct model_aux *) neq)->mu;
  double *in = ((struct model_aux *) neq)->u;

  static double vds = 180.0, vqs = 0.0, w0 = 2. * M_PI * 60.;
  static double i[5];
                                                                       120
  double Lrr, Lm, Grr, Gm;

  Lrr = LM (mu) + LL (mu);
  Lm = LM (mu);

  Grr = Lrr / (Lrr * Lrr - Lm * Lm);
  Gm = Lm / (Lrr * Lrr - Lm * Lm);

  i[0] = Grr * x[0] - Gm * x[2];
  i[1] = Grr * x[1] - Gm * x[3];                                       130
  i[2] = -Gm * x[0] + Grr * x[2];
  i[3] = -Gm * x[1] + Grr * x[3];

  xdot[0] = -(RS (mu) * i[0] - w0 * x[1] - vds);
  xdot[1] = -(RS (mu) * i[1] + w0 * x[0] - vqs);
  xdot[2] = -(RR (mu) * i[2] - (w0 - x[4]) * x[3]);
  xdot[3] = -(RR (mu) * i[3] + (w0 - x[4]) * x[2]);
  xdot[4] = (1.5 * 2. * (x[3] * i[2] - x[2] * i[3]) - B (mu) * x[4]) * K (mu);
}
                                                                       140
```

```
static void
i (double *x, double *mu)
{
  x[0] = x[1] = x[2] = x[3] = x[4] = 0.0;
}

void
induction (model_t * m, const char *opts, double *parms)
{
  double mu0[] =                                                          150
  {.35, .42, .069, .002, 30, .05};

  if (!strcasecmp ("ia", opts))
    {
      model_init (m, "induction motor model ia of lab frame", 5, 1, 1, 6);
      model_setfuncs (m, i, f, g0);
    }
  else if (!strcasecmp ("iabc", opts))
    {
      model_init (m, "induction motor model, lab frame", 5, 1, 3, 6);     160
      model_setfuncs (m, i, f, g1);
    }
  else if (!strcasecmp ("d", opts))
    {
      model_init (m, "induction motor model, d-axis output only", 5, 1, 3, 6);
      model_setfuncs (m, i, f, g3);
    }
  else
    {
      model_init (m, "induction motor model, idq outputs", 5, 1, 2, 6);   170
      model_setfuncs (m, i, f, g2);
    }

  if (parms)
    model_setparms (m, parms);
  else
    {
      model_setparms (m, mu0);
    }
}                                                                         180
```

Program C.8 dcmotor.c *(§ 4.3.1)*

```
/* generated Fri Dec 10 15:39:51 1999 on nitro */                          0

/*

                  Written by Steven R. Shaw, 1999

            Copyright (c) 1999 MASSACHUSETTS INSTITUTE
                OF TECHNOLOGY. All rights reserved.

*/
                                                                          10


#include <string.h>
```

```
#include <err.h>
#include <math.h>

#include "model.h"
#include "dcmotor.h"

#ifdef DEBUG                                                          20
#define CHECKPOINT() warnx("function %s, line %d", __FUNCTION__,
__LINE__);
#else
#define CHECKPOINT()
#endif

#define A1(x) ((x)[0])
#define A2(x) ((x)[1])
#define A3(x) ((x)[2])
#define V(x)  ((x)[3])                                                30
#define S(x)  ((x)[4])

static void
ga (double *x, double *u, double *y, double *mu)
{
  y[0] = A3 (mu) * (V (mu) - A1 (mu) * x[0]);
}

static void
fa (int *neq, double *tim, double *x, double *xdot)                   40
{
  double *mu = ((struct model_aux *) neq)->mu;

  xdot[0] = V (mu) - (A1 (mu) + A2 (mu) * x[0]) * x[0];
}

static void
gb (double *x, double *u, double *y, double *mu)
{
  y[0] = A3 (mu) * (u[1] - A1 (mu) * x[0]);                           50
}

static void
fb (int *neq, double *tim, double *x, double *xdot)
{
  double *mu = ((struct model_aux *) neq)->mu;
  double *u = ((struct model_aux *) neq)->u;

  xdot[0] = u[1] - (A1 (mu) + A2 (mu) * x[0]) * x[0];
}                                                                     60

static void
ia (double *x, double *mu)
{
  x[0] = 0.0;
}

static void
ib (double *x, double *mu)
{                                                                     70
```

```
  x[0] = S (mu);
}

void
dcmotor (model_t * m, const char *opts, double *parms)
{
  double mu0[] =
  {5.0, 2.0, 1.0, 12.0, 0.0};

  if (!strcasecmp ("vi", opts))                                          80
    {
      model_init (m,
   "dc motor model : voltage as input, from rest",
   1, 2, 1, 3);
      model_setfuncs (m, ia, fb, gb);
    }
  else if (!strcasecmp ("vp,sp", opts))
    {
      model_init (m,
   "dc motor model : voltage, starting speed as parameters",             90
   1, 1, 1, 5);
      model_setfuncs (m, ib, fa, ga);
    }
  else
    {
      model_init (m,
   "dc motor model : voltage as parameter, from rest",
   1, 1, 1, 4);
      model_setfuncs (m, ia, fa, ga);
    }                                                                   100

  if (parms)
    model_setparms (m, parms);
  else
    model_setparms (m, mu0);
}
```

Program C.9 iowa3.c *(§ 4.2.2)*

```
/* generated Fri Dec 10 15:39:52 1999 on nitro */                        0

/*

                  Written by Steven R. Shaw, 1999

             Copyright (c) 1999 MASSACHUSETTS INSTITUTE
                 OF TECHNOLOGY. All rights reserved.

*/
                                                                        10


#include <string.h>
#include <err.h>
#include <math.h>

#include "model.h"
```

```
#include "iowa3.h"

#ifdef DEBUG                                                          20
#define CHECKPOINT() warnx("function %s, line %d", __FUNCTION__,
__LINE__);
#else
#define CHECKPOINT()
#endif

#define RS(mu) (exp((mu)[0]))
#define RR(mu)  (exp((mu)[1]))
#define LM(mu) (exp((mu)[2]))
#define LL(mu) (exp((mu)[3]))                                         30

#define K_1(mu) (exp((mu)[4]))
#define A_1(mu) (exp((mu)[5]))
#define B_1(mu) (exp((mu)[6]))

#define K_2(mu) (exp((mu)[7]))
#define A_2(mu) (exp((mu)[8]))
#define B_2(mu) (exp((mu)[9]))

static void                                                           40
f (int *neq, double *tim, double *x, double *xdot)
{
  double *mu = ((struct model_aux *) neq)->mu;
  double *in = ((struct model_aux *) neq)->u;

  static double vds = 554.0, vqs = 0.0, w0 = 2. * M_PI * 60.;
  static double Lrr, Lm, Grr, Gm;
  static double i[5];

  vqs = in[1];                                                        50
  vds = in[2];

  Lrr = LM (mu) + LL (mu);
  Lm = LM (mu);
  Grr = Lrr / (Lrr * Lrr - Lm * Lm);
  Gm = Lm / (Lrr * Lrr - Lm * Lm);

  i[0] = Grr * x[0] - Gm * x[2];
  i[1] = Grr * x[1] - Gm * x[3];
  i[2] = -Gm * x[0] + Grr * x[2];                                     60
  i[3] = -Gm * x[1] + Grr * x[3];

  xdot[0] = -(RS (mu) * i[0] - w0 * x[1] - vds);
  xdot[1] = -(RS (mu) * i[1] + w0 * x[0] - vqs);
  xdot[2] = -(RR (mu) * i[2] - (w0 - x[4]) * x[3]);
  xdot[3] = -(RR (mu) * i[3] + (w0 - x[4]) * x[2]);
  xdot[4] = (1.5 * 2. * (x[3] * i[2]
  - x[2] * i[3]) - B_1 (mu) * x[4]) * K_1 (mu);

  vqs = in[3];                                                        70
  vds = in[4];

  i[0] = Grr * x[5] - Gm * x[7];
  i[1] = Grr * x[6] - Gm * x[8];
```

```
  i[2] = -Gm * x[5] + Grr * x[7];
  i[3] = -Gm * x[6] + Grr * x[8];

  xdot[5] = -(RS (mu) * i[0] - w0 * x[6] - vds);
  xdot[6] = -(RS (mu) * i[1] + w0 * x[5] - vqs);
  xdot[7] = -(RR (mu) * i[2] - (w0 - x[9]) * x[8]);                    80
  xdot[8] = -(RR (mu) * i[3] + (w0 - x[9]) * x[7]);
  xdot[9] = (1.5 * 2. * (x[8] * i[2]
  - x[7] * i[3]) - B_2 (mu) * x[9]) * K_2 (mu);
}

static void
g0 (double *x, double *u, double *y, double *mu)
{
  const double w0 = 2 * M_PI * 60;
  double Lrr, Lm, Grr, Gm, ids, iqs;                                   90

  Lrr = LM (mu) + LL (mu);
  Lm = LM (mu);
  Grr = Lrr / (Lrr * Lrr - Lm * Lm);
  Gm = Lm / (Lrr * Lrr - Lm * Lm);

  ids = Grr * x[0] - Gm * x[2];
  iqs = Grr * x[1] - Gm * x[3];
  y[0] = iqs;
  y[1] = ids;                                                         100

  ids = Grr * x[5] - Gm * x[7];
  iqs = Grr * x[6] - Gm * x[8];
  y[2] = iqs;
  y[3] = ids;
}

static void
i (double *x, double *mu)
{                                                                     110
  int i;

  for (i = 0; i < 10; i++)
    x[i] = 0.0;
}

void
iowa3 (model_t * m, const char *opts, double *parms)
{
                                                                      120
  double mu0[] =
  {-.223, -1.20, -2.67, -5.29, 5.01, -13.8, -2.99, 5.01, -13.8, -2.99};

  model_init (m,
       "Model for IEC fan situation, dq outputs. Requires inputs.",
       10, 5, 4, 10);
  model_setfuncs (m, i, f, g0);

  if (parms)
    model_setparms (m, parms);                                        130
  else
```

```
    model_setparms (m, mu0);
}
```

C.4 Preprocessing programs for nonintrusive diagnostics

Program C.10 `fanwrapper` *used to preprocess fan data (§ 4.3.1).*

```
#!/usr/local/bin/octave --silent                                          0
%-*-Octave-*-
% identification frontend for fan in car

clear;

function [ind] = clip(y,nt)
  N = rows(y)/3;
  a = mean(y(1:N));
  b = max(abs(y(1:N) - a));
  j = find(medfilt1(abs(medfilt1(y-a,5)) > 1.5*b,5) == 1);                10

  if j == []
    ind = [];
  else
    ind = [j(1) : min( [j(1)+nt-1 length(y)])];
  end;
end;

[str,xln] = fgets(stdin,200);
[data,count] = fread(stdin,[1000 1],"double");                            20

% get rid of DC offset

dcoff = mean(data(1:400));
data = data - dcoff;

% find the transient -- want 400 points
ind = clip(data,400);

T = (1:rows(data))' ./ 250;       % samples                               30
data = data ./ 25.0;

% apply scale factors and create a time vector
A = [T(ind)  ones(length(ind),1)*12.0 data(ind)];
Ap = [T data];

% write data
save -ascii dcmotor.dat A

% open a pipe to the real identification program                           40
p = popen('id-dcmotor -z ".04" -o "vi" -p ".339 .137 .362 12.0" -i
dcmotor.dat -r all', "r");
z = fscanf(p,"%f", Inf);
pclose(p);

% everything is in z -- break off the parameters from the simulation part
mu = z(1:3);
```

```
sim = z(length(mu)+1:length(z));
sim = reshape(sim,length(sim),1);
                                                                          50
%sim = A(:,2);
%mu = [0 1 1];

M = 2;
N = rows(sim);


% write results to a graphics stream
fd = dup2(stdout,stderr);     % dup stdout so Octave doesn't mangle binary data
fputs(fd,str(1:xln-1));                                                   60
fprintf(fd," : [");
fprintf(fd,"%f ", mu);
fprintf(fd,"]\n");
M = 2;
fwrite(fd,M,"int32");           % M sections follow
fwrite(fd,rows(Ap),"int32");     % N ordinates/abscissa follow
fwrite(fd,Ap(:,1), "float32");   % write time vector
fwrite(fd,Ap(:,2), "float32");   % write observations
fwrite(fd,rows(A),"int32");     % N ordinates/abscissa follow
fwrite(fd,A(:,1), "float32");   % write time vector                       70
fwrite(fd,sim, "float32");      % write simulations
```

Program C.11 motorwrapper *used to preprocess raw induction motor data for identification (§ 4.3.2).*

```
#!/usr/local/bin/octave --silent                                          0
%-*-Octave-*-
% identification frontend for fan in car

clear;

function x = trimss(y,N)
  x = y(1:N);

  for k = 1:128
    ind = (k:128:length(x))';                                             10
    xi = x(ind);
    mu(:,k) = [ones(size(ind)) ind] \ xi;
  end;

  xhat = zeros(size(y));
  for k = 1:128
    ind = (k:128:length(xhat))';
    xhat(ind,:) = [ones(size(ind)) ind] * mu(:,k);
  end;
                                                                          20
  x = y-xhat;
end;

function [ind] = clip(y,nt)
  N = rows(y)/3;
```

```
  a = mean(y(1:N));
  b = max(abs(y(1:N) - a));
  j = find(medfilt1(abs(medfilt1(y-a,5)) > 4.0*b,5) == 1);

  if j == []                                                          30
    ind = [];
  else
    ind = [j(1) : min( [j(1)+nt-1 length(y)])];
  end;
end;


% Find phi fitting the model
% y(t) = z(1)*sin(t+phi)
% with z(1) > 0.                                                     40
function [z,phi] = findphase(y, t)
  phi = 0.0;

  for i = 1:4
    z = [sin(t+phi) cos(t+phi) ones(size(t))] \ y;

    if(z(1) == 0)
      th = pi/2;
    else
      th = atan(z(2)/z(1)) + pi*(z(1) < 0);                           50
    end;
    phi = rem(phi+th+2*pi,2*pi);
  end;
end;



%%% input the data %%%
% with AC nilm we'll get 64000 doubles x 2 columns
[str,xln] = fgets(stdin,200);                                         60
[data,count] = fread(stdin,[2*64000 1],"double");
v = data(1:64000);
i = data(64001:128000);

%%% prep the data %%%
% Do periodic trend removal on the transient.
% model : i(k+128) = i(k) + k*\alpha

it = trimss(i,10000);
                                                                      70
% find the transient.
ind = clip(it,3200);

I = it(ind);
V = v(ind);

ang = (0:rows(V)-1)'* (2*pi/128);      % angles

% resolve the phase
[z,phi] = findphase(V,ang);            % send in angles, get out angle.  80

t = (ang+phi)/(2*pi*60);
```

```
% a scale factor of .1 is about right to give us amps in the
% mock building.
A = [t I/10];

th = 2*pi*60*t;

B = [t z(1)*sin(th)+z(3) V  [sin(th) cos(th) ones(size(th))]*z] ;                    90

save -ascii voltage.dat B z
save -ascii induction.dat A

%%% Identification %%%
% open a pipe to the real identification program
p =  popen('id-induction -z ".4" -o "ia" -p "7 4 .8 .03 1000 .002"  -i induction.dat
-r all', "r");
z = fscanf(p,"%f", Inf);
pclose(p);                                                                          100


% everything is in z -- break off the parameters from the simulation part
mu = z(1:6);
sim = z(length(mu)+1:length(z));
sim = reshape(sim,length(sim),1);

%%% output %%%
% write results to a graphics stream
M = 2;                                                                              110
N = rows(sim);

fd = dup2(stdout,stderr);     % dup stdout so Octave doesn't mangle binary data
fputs(fd,str(1:xln-1));
fprintf(fd," : [");
fprintf(fd,"%f ", mu);
fprintf(fd,"]\n");
M = 2;
fwrite(fd,M,"int32");           % M sections follow
fwrite(fd,N,"int32");     % N ordinates/abscissa follow                           120
fwrite(fd,A(1:N,1), "float32");   % write time vector
fwrite(fd,A(1:N,2), "float32");   % write observations
fwrite(fd,N,"int32");     % N ordinates/abscissa follow
fwrite(fd,A(1:N,1), "float32");   % write time vector
fwrite(fd,sim(1:N), "float32");      % write simulations



                                                                                    130
```

Bibliography

[1] S. Audoly, L. D'Angiò, M. P. Saccomani, and C. Cobelli. Global identifiability of linear compartmental models – a computer algebra algorithm. *IEEE Transactions on Biomedical Engineering*, 45(1):36–47, January 1998.

[2] Douglas M. Bates and Donald G. Watts. A generalized Gauss-Newton procedure for multiresponse parameter estimation. *SIAM Journal on Scientific and Statistical Computing*, 8(1):49–55, January 1987.

[3] Douglas M. Bates and Donald G. Watts. *Nonlinear regression analysis and its applications.* Probability and Mathematical Statistics. Wiley, 1988.

[4] Philip R. Bevington and D. Keith Robinson. *Data Reduction and Error Analysis for the Physical Sciences.* McGraw-Hill, second edition, 1992.

[5] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

[6] Paul T. Boggs, Richard H. Byrd, and Robert B. Schnabel. A stable and efficient algorithm for nonlinear orthogonal distance regression. *SIAM Journal on Scientific and Statistical Computing*, 8(6):1052–1078, November 1987.

[7] Peter N. Brown and Youcef Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific and Statistical Computing*, 11(3):450–481, May 1990.

[8] L. Carmichael. Nonintrusive appliance load monitoring system. *EPRI Journal*, pages 45–47, September 1990.

[9] A. Chaudhuri and M. Moallem. Application of autoregressive backpropagation network for parameter estimation of nonlinear systems. *Proceedings of the Fourteenth International Conference on Applied Informatics*, pages 296–299, 1996.

[10] Jian Shen Chen and Robert I. Jennrich. Diagnostics for linearization confidence intervals in nonlinear regression. *Journal of the American Statistical Association*, 90(431):1068–1074, September 1995.

[11] K. R. Cho, J. H. Lang, and S. D. Umans. Detection of broken rotor bars in induction motors using parameter and state estimation. *IEEE Transactions on Industrial Applications*, vol. 28, no. 1, May/June 1992, pp. 702-708.

[12] Kyong Rae Cho. Detection of broken rotor bars in induction motors using parameter and state estimation. Master's thesis, MIT, June 1989.

[13] M. Chow, R. Sharpe, and J. Hung. On the application and design of artificial neural networks for motor fault detection – part i. *IEEE Transactions on Industrial Electronics*, 40(2):181–188, April 1993.

[14] M. Chow, R. Sharpe, and J. Hung. On the application and design of artificial neural networks for motor fault detection – part ii. *IEEE Transactions on Industrial Electronics*, 40(2):189–196, April 1993.

[15] A. I. Cole and A. Albicki. Data extraction for effective non-intrusive identification of residential power loads. In *IEEE Instrumentation and Measurement Technology Conference*, pages 812–815, St. Paul Minnesota, May 1998.

[16] William C. Davidon. Conic approximations and collinear scalings for optimizers. *SIAM Journal on Numerical Analysis*, 17(2):268–281, April 1980.

[17] Alvin Drake. *Fundamentals of Applied Probability Theory.* McGraw-Hill, 1988.

[18] I. S. Duff, J. Nocedal, and J. K. Reid. The use of linear programming for the solution of sparse sets of nonlinear equations. *SIAM Journal on Scientific and Statistical Computing*, vol.8(no.2):99–108, March 1987.

[19] Howard C. Elman. Iterative methods for large, sparse, nonsymmetric systems of linear equations. Technical Report 229, Yale University, April 1982.

[20] A.E. Fitzgerald, C. Kingsley, and S.D. Umans. *Electric Machinery.* McGraw-Hill, 1983.

[21] Aileen Frisch. *Essential System Administration.* O'Reilly, 1995.

[22] Neil Gershenfeld. *The Nature of Mathematical Modeling.* Cambridge University Press, 1999.

[23] Gene H. Golub and Charles F. Van Loan. An analysis of the total least squares problem. *SIAM Journal on Numerical Analysis*, 17(6):883–893, December 1980.

[24] P. Goode and M. Chow. Using a neural/fuzzy system to extract heuristic knowledge of incipient faults in induction motors: Part i – methodology. *IEEE Transactions on Industrial Electronics*, 42(2):131–138, April 1995.

[25] P. Goode and M. Chow. Using a neural/fuzzy system to extract heuristic knowledge of incipient faults in induction motors: Part ii – application. *IEEE Transactions on Industrial Electronics*, 42(2):139–146, April 1995.

[26] R. B. Guenther and R. Schmidt. Remarks on parameter identification. *SIAM Journal on Applied Mathematics*, 37(3):619–623, December 1979.

[27] G. W. Hart. Residential energy monitoring and computerized surveillance via utility power flows. *IEEE Technology and Society*, pages 12–16, June 1989.

[28] G. W. Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891, December 1992.

[29] Wolfgang M. Hartmann and Robert E. Hartwig. Computing the moore-penrose inverse for the covariance matrix in constrained nonlinear estimation. *SIAM Journal of Optimization*, 6(3):727–747, August 1996.

[30] Rolf Johansson. *System Modeling and Identification.* Prentice Hall Information and System Sciences Series. Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[31] Lester F. Caudill Jr, Herschel Rabitz, and Attila Askar. A direct method for the inversion of physical systems. *Inverse Problems*, 10:1099–1114, 1994.

[32] U. A. Khan, S. B. Leeb, and M. C. Lee. A multiprocessor for transient event detection. *IEEE Transactions on Power Delivery*, 12(1):51–60, Jan. 1997.

[33] U. A. Khan, S. B. Leeb, S. R. Shaw, L. K. Norford, M. C. Lee, and S. Pandey. A multiprocessing platform for transient event detection. Technical report, LEES, May 1995.

[34] Jürgen M. Kleinhans, Georg Sigl, Frank M. Johannes, and Kurt J. Antreich. Gordian: Vlsi placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-aided Design*, 10(3):356–365, March 1991.

[35] G. B. Kliman, W. J. Premerlani, B. Yazici, R. A. Koegl, and J. Mazereeuw. Sensorless, online motor diagnostics. *IEEE Computer applications in power*, 10(2):39–43, April 1997.

[36] Marek A. Kowalski, Krszysztof A. Sikorski, and Frank Stenger. *Selected Topics in Approximation and Computation.* Oxford University Press, 1995.

[37] Paul C. Krause. *Analysis of Electric Machinery.* McGraw-Hill Book Company, 1986.

[38] S. B. Leeb. *A Conjoint Pattern Recognition Approach to Nonintrusive Load Monitoring.* Phd, MIT, Department of Electrical Engineering and Computer Science, February 1993.

[39] S. B. Leeb, U. A. Khan, and S. R. Shaw. *Multiprocessing transient event detector for use in a nonintrusive electrical load monitoring system.* U. S. Patent Number 5,717,325, Issued February 1998.

[40] S. B. Leeb and J. L. Kirtley. A multiscale transient event detector for nonintrusive load monitoring. In *Proceedings of the 1993 IECON International Conference on Industrial Electronics, Control, and Instrumentation*, 93CH3234-2, pp. 354–359.

[41] S. B. Leeb and J. L. Kirtley. *A Transient Event Detector for Nonintrusive Load Monitoring.* U. S. Patent Number 5,483,153, Issued January 1996.

[42] S. B. Leeb and S. R. Shaw. Harmonic estimates for transient event detection. *Proceedings of the 29th Universities Power Engineering Conference*, pages 133–166, 1994.

[43] S. B. Leeb, S. R. Shaw, and J. L. Kirtley. Transient event detection in spectral envelope estimates for nonintrusive load monitoring. *IEEE Transactions on Power Delivery*, 7(3):1200–1210, July 1995.

[44] Alberto Leon-Garcia. *Probability and Random Processes for Electrical Engineering.* Addison-Wesley, second edition, 1993.

[45] J. R. Levine, T. Mason, and D. Brown. *lex & yacc.* Unix Programming Tools. O'Reilly, 1992.

[46] Lennart Ljung. *System Identification: Theory for the User.* Prentice-Hall, second edition, January 1999.

[47] Richard E. Maine and Kenneth W. Iliff. Formulation and implementation of a practical algorithm for parameter estimation with process and measurement noise. *SIAM Journal on Applied Mathematics*, 41(3):558–579, December 1981.

[48] Timothy Masters. *Practical Neural Network Recipes in C++.* Academic Press, 1993.

[49] J. McElveen, K. Lee, and J. Bennett. Identification of multivariable linear systems from input/output measurements. *IEEE Transactions on Industrial Electronics*, 39(3):189–193, June 1992.

[50] Kenneth S. Miller and Donald M. Leskiw. *An introduction to Kalman filtering with applications.* Robert E. Kreiger Publishing Company, 1987.

[51] Kazuaki Minami. Model-based speed and parameter tracking for induction machines. Master's thesis, MIT, May 1989.

[52] V. V. Nguyen and E. F. Wood. Review and unification of linear identifiability concepts. *SIAM Review*, 24(1):34–51, January 1982.

[53] L. K. Norford and S. B. Leeb. Nonintrusive electrical load monitoring in commercial buildings base on steady-state and transient load detection algorithms. *Energy and Buildings*, 24(1):51–64, May 1996.

[54] Hannu Pihala. Non-intrusive appliance load monitoring system based on a modern kwh-meter. Licentiate thesis, Technical Research Centre of Finland, Espoo, May 1999.

[55] W. H. Press, S. A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C.* Cambridge University Press, second edition, 1992.

[56] George N. Saridis and Robert Hofstadter. A pattern recognition approach to the classification of nonlinear systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-4(4):362–371, July 1974.

[57] G. A. F. Seber and C. J. Wild. *Nonlinear Regression.* Probability and Mathematical Statistics. Wiley, 1989.

[58] S. R. Shaw. Numerical methods for identification of induction motor parameters. Master's thesis, MIT, February, 1997.

[59] S. R. Shaw, C. B. Abler, R. F. Lepard, D. Luo, S. B. Leeb, and L. K. Norford. Instrumentation for high performance nonintrusive electrical load monitoring. *ASME Journal of Solar Energy Engineering*, 120(3):224–229, August 1998.

[60] S. R. Shaw and S. B. Leeb. Identification of induction motor parameters from transient stator current measurements. *IEEE Transactions on Industrial Electronics*, 46(1):139–149, February 1999.

[61] Jonas Sjöberg, Qinghua Zhang, Lennart Ljung, Albert Benveniste, Bernard Delyon, Pierre-Yves Glorennec, Håkan Hjalmarsson, and Anatoli Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724, 1995.

[62] Gilbert Strang. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, 1986.

[63] K. J. Åström. Maximum likelihood and prediction error methods. *Automatica*, 16(5):551–574, 1980.

[64] K. J. Åström and P. Eykhoff. System identification – a survey. *Automatica*, 7:123–162, 1971.

[65] Miguel Vélez-Reyes. Speed and parameter estimation for induction machines. Master's thesis, MIT, May 1988.

[66] Miguel Vélez-Reyes. *Decomposed Algorithms for Parameter Estimation.* PhD thesis, MIT, September 1992.

[67] Miguel Vélez-Reyes, Kazuaki Minami, and George C. Verghese. Recursive speed and parameter estimation for induction machines. In *Proceedings of the 1989 IEEE Industry Applications Society Annual Meeting*, 89CH2792-0, pp. 607-611.

[68] Larry Wall, Tom Christiansen, and Randal Schwartz. *Programming Perl.* O'Reilly, second edition, 1995.

[69] Y. Wang and C. Lin. Runge-kutta neural network for identification of dynamical systems in high accuracy. *IEEE Transactions on Neural Networks*, 9(2):294–307, March 1995.

[70] S. Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos.* Number 2 in Texts in Applied Mathematics. Springer-Verlag, 1990.

[71] R. Williamson and U. Helmke. Existance and uniqueness results for neural network approximations. *IEEE Transactions on Neural Networks*, 6(1):2–13, January 1995.

[72] Michael T. Wishart and Ronald G. Harley. Identification and control of induction machines using artificial neural networks. *IEEE Transactions on Industrial Applications*, 31(3):612–619, May-June 1995.

[73] Herbert H. Woodson and James R. Melcher. *Electromechanical Dynamics*, volume II: Fields, Forces and Motion. Krieger, 1968.

[74] S. J. Wright and J. N. Holt. Algorithms for nonlinear least squares with linear inequality constraints. *SIAM Journal on Scientific and Statistical Computing*, 6(4):1033–1048, October 1985.

[75] B. Yazici, G. B. Kliman, W. J. Premerlani, R. A. Koegl, and A. Abdel-Malek. An adaptive, on-line, statistical method for detection of broken bars in motors using stator current and torque estimation. In *IEEE Industrial Applications Society*, volume 1, pages 221–226, October 1997.

[76] B. Yazici, G. B. Kliman, W. J. Premerlani, R. A. Koegl, G. B. Robinson, and A. Abdel-Malek. An adaptive, on-line, statistical method for bearing fault detection using stator current. In *IEEE Industrial Applications Society*, volume 1, pages 213–220, October 1997.

[77] A. M. Zoubir and B. Boashash. The bootstrap and its application in signal processing. *IEEE Signal Processing Magazine*, 15(1):56–76, January 1998.