

**An Improved Algorithm for Identification of
Time Varying Parameters using Recursive Digital Techniques**

by

Bernard Christopher Patrick Maloney

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Mechanical Engineering

APPROVED:

Robert G. Leonard, Chairman

Robert H. Fries

Harry H. Robertshaw

**September, 1986
Blacksburg, Virginia**

**An Improved Algorithm for Identification of
Time Varying Parameters using Recursive Digital Techniques**

by

Bernard Christopher Patrick Maloney

Robert G. Leonard, Chairman

Mechanical Engineering

(ABSTRACT)

Identification is the process of determining values for the characteristic quantities, called parameters, of a system. Examples of such quantities are mass, inductance, resistance, spring coefficient, gain, et cetera. The decreasing cost of digital processors and the versatility of digital programming make digital methods an attractive means of accomplishing identification. It is important, however, that an identifier be able to track any change in a parameter if its output is to be used in any predictive capacity, such as in an adaptive controller. Most studies of digital identification have avoided the topic of time variations by using batch processing methods which implicitly assume constant parameters; this thesis does not.

This thesis first investigates the parameter-tracking capabilities of a popular, real-time digital identification algorithm, the recursive weighted least squares method. This method is claimed to be able to track only slowly time-varying parameters. Based on the results of this study, a method of improving the accuracy of estimates of time-varying parameters is developed. This method, called conditioning, is a post-processor to the recursive weighted least squares algorithm. The results of tests of this method using three different plant simulations are presented, demonstrating the improved accuracy achieved by conditioning estimates of time varying parameters.

Acknowledgements

There are many more than these to thank for the encouragement and support they have offered me.

Dr. Robert G. Leonard, my thesis advisor. His nearly constant accessibility over the past year has allowed me to work at my own pace and is deeply appreciated. He is a living definition of Patience.

Dr. Robert H. Fries and Dr. Harry H. Robertshaw who have served on my advisory and examining committees. Their interest and opinions have helped me shape this work.

Mrs. Jan Riess, the department's Program Support Technician. More succinctly, she's Mom. She is a clearing-house for everything, a dry shoulder when needed, a good friend.

Dr. and Mrs. Charles J. Hurst who have most graciously let me stay with them during the final weeks of this work.

All my friends for their interest, encouragement and inspiration. In particular, Sampson Lee Blair, Jerry Elder, Anne Louise Goode, Becky Lanier Merrifield and Laura Powers have listened to my problems, weathered my frustrations and revived my sagging enthusiasm.

Susan and Dervish who have tirelessly supported me over the last 3000 miles or so. Too, Their Royal Puppynesses, Sir Cedric Mutley of Woofdom and Lady Folly Diamond for their attention, enthusiasm and affection which have never failed to brighten my day.

Finally, I must thank my family for never doubting me, never discouraging me and always Being There. Each and every one a Maloney.

For the last crew of the *Challenger*.

Table of Contents

List of Illustrations	vi
List of Tables	vii
1. Introduction	1
2. Literature Review	4
2.1 Summary	10
3. Capabilities of Recursive Weighted Least Squares Estimation	12
3.1 Test Plant Selection	12
3.2 Estimation of Constant Parameters	13
3.3 Estimation of Changing Parameters	18
3.3.1 Quantification of Slow Parameter Change	24
3.3.2 Forgetting Factor Effects	33
3.4 Summary of Behavior	40
4. Estimate Conditioning	42
4.1 Conditioning Process Logic	43
4.2 Behavior of Conditioned Estimates	47
4.2.1 Test Plant of the Previous Chapter	48
4.2.2 Random Second Order Plant	53
4.2.3 Overdamped Second Order Plant	55
5. Conclusions	60

5.1 Recommendations for Future Studies	63
References	72
Appendix A.	73
Appendix B.	82
Appendix C.	85
Vita	93
Acknowledgements	iii

List of Illustrations

Figure 1. Signal Flow in an Adaptively Controlled Plant.	2
Figure 2. Plant Response to Constant Step Input.	14
Figure 3. Estimate Response to Constant Step Input: Equation Error.	17
Figure 4. Estimate Response to Constant Step Input: Output Error.	19
Figure 5. Reduced Weight with Age by Forgetting Factor.	21
Figure 6. Estimate Response to Constant Step Input with Forgetting Factor = 0.95.	23
Figure 7. Noisy Forcing Function.	25
Figure 8. Plant Response to Noisy Forcing Function.	26
Figure 9. Estimate Response to Noisy Forcing Function.	27
Figure 10. Ramping-Step Change of a Parameter.	28
Figure 11. Normalized Estimates of k/m for Several Spring Coefficient Changes.	30
Figure 12. Normalized Estimates of d/m for Several Spring Coefficient Changes.	31
Figure 13. Normalized Estimates of $1/m$ for Several Spring Coefficient Changes.	32
Figure 14. Normalized Estimates of k/m for Several Forgetting Factors.	36
Figure 15. Normalized Estimates of d/m for Several Forgetting Factors.	37
Figure 16. Normalized Estimates of $1/m$ for Several Forgetting Factors.	38
Figure 17. Flow Diagrams of RWLS Estimates and Conditioned RWLS Estimates.	44
Figure 18. Flow Chart of the Estimate Conditioner.	46
Figure 19. Estimates for a Spring Coefficient Increase of 1%.	49
Figure 20. Estimates for a Spring Coefficient Increase of 10%.	50
Figure 21. Estimates for a Spring Coefficient Increase of 100%.	51
Figure 22. Estimates for a Spring Coefficient Increase of 1000%.	52
Figure 23. Conditioned Estimates Produced with Adjusted Tolerances	54
Figure 24. Estimates of Randomly Generated Plant Parameters: Damper Variation.	56
Figure 25. Estimates of Randomly Generated Plant Parameters: Spring Variation.	57
Figure 26. Estimates of an Overdamped Second Order Plant: Spring Variation.	59

Figure 27. Plant for Reduced Order Identification.	64
Figure 28. Reduced Order Estimates for Added Root at $s = -50$	66
Figure 29. Reduced Order Estimates for Added Root at $s = -10$	67
Figure 30. Reduced Order Estimates for Added Root at $s = -2$	68
Figure 31. Reduced Order Estimates for Added Root at $s = -0.5$	69
Figure 32. Reduced Order Estimates with Parameter Variation.	70

List of Tables

Table 1. Maximum Error of Parameter Estimates for Several Spring Coefficient Changes. .	34
Table 2. Maximum Error of Parameter Estimates for Several Forgetting Factors.	39

1. Introduction

Identification is the process of establishing values for and tracking any changes in characteristic quantities in a plant such as mass, spring coefficient, resistance or gains. The characteristic quantities are called **parameters** and the results of identification are called **estimates** of the parameters. Identifiers can be used to monitor parameters which cannot be easily measured. A record of measurements of parts produced by an NC engine lathe could be used to estimate tool wear and predict when the lathe will begin producing out-of-tolerance parts. Identification is also important in adaptive control. By identifying a plant's parameters, an adaptive controller can predict the plant's behavior and react accordingly to maintain a desired performance.

Figure 1 is a diagram of signal flows between the various elements in an adaptively controlled plant. Using measurements of plant variables, the identifier calculates the coefficients of these variables in a mathematical model of the plant. The adaptive controller then uses the plant's mathematical model to predict plant behavior, adjusting the forcing function, F , to achieve the performance established by the reference signal.

Because of the decreasing cost of digital processors and the versatility of digital programming, digital methods are an attractive alternative to analog circuits for identification purposes. To track changing parameters in a timely manner it is necessary to use a recursive digital algorithm. The most widely used recursive algorithm, recursive weighted least squares (RWLS), is claimed to be able to track only slow parameter changes. The purpose of this thesis is to propose and examine a modification to improve both the accuracy of estimates produced using RWLS estimation and the algorithm's ability to track rapid parameter changes.

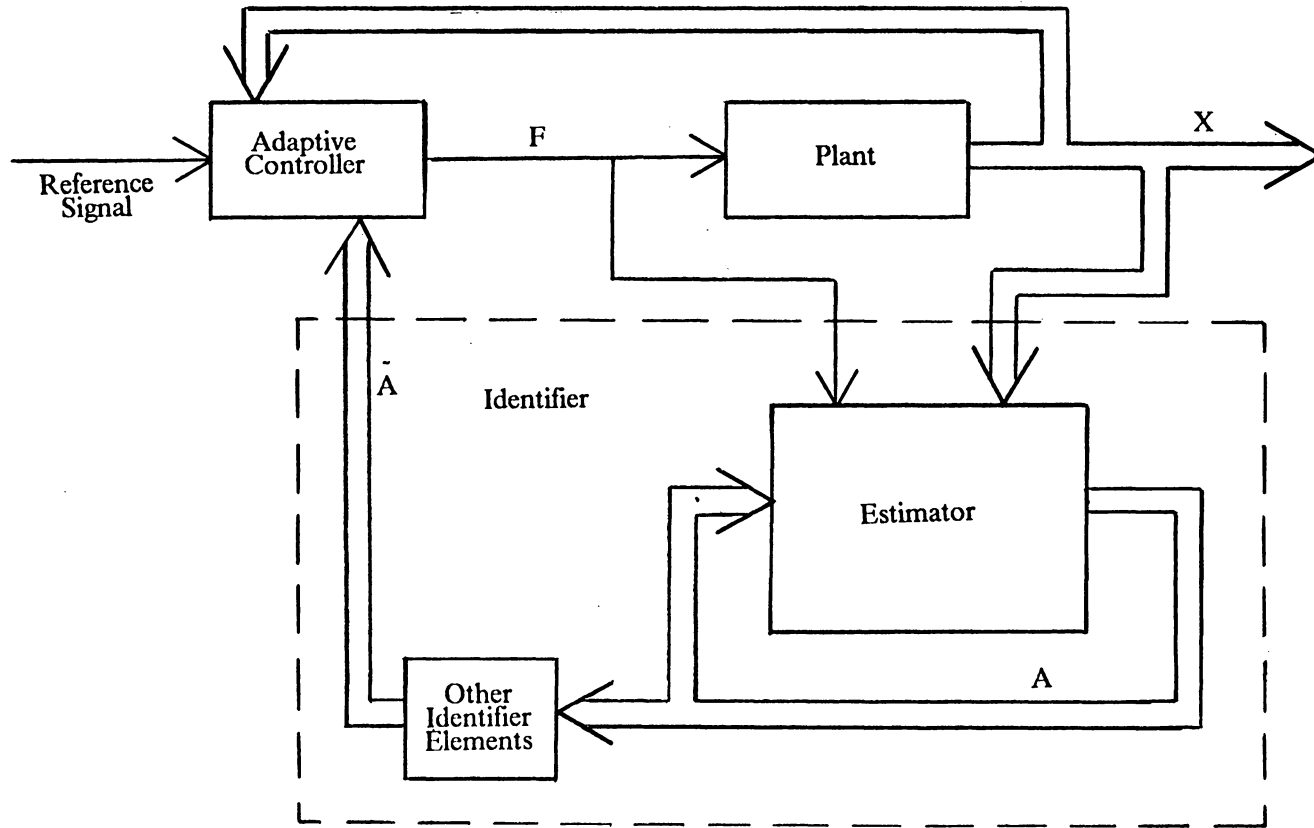


Figure 1. Signal Flow in an Adaptively Controlled Plant.

Some distinction must be made here between *estimation* and *identification*. Traditionally, these two terms have been used interchangeably. However, this thesis proposes to post-process the results of estimation, hence *estimation* is considered a sub-process of *identification*, as shown by Figure 1. The post-processor, termed a conditioner, improves the performance of the identification process.

Beginning with a literature review to establish current needs, this thesis proceeds with an investigation of the capabilities of RWLS estimation. Based on these findings, the conditioning process is outlined and its performance compared to that of unadulterated RWLS estimation for several plants. Finally the results of this investigation are summarized and recommendations are made for future studies.

2. Literature Review

Adaptive control is an attractive tool for the control engineer. Conventional automatic control design techniques require a fairly good model of the plant including parameter values. Variations in physical parameters such as damping, gains, resistances, moments of inertia, et cetera, may cause the plant/controller combination to produce undesired results or become unstable. Adaptive control can maintain a desired plant response to a reference signal even if plant dynamics are ill-known or changing. This requires, however, accurate tracking of any time variations in the plant's performance.

Parameter identification is one way to track plant performance. In his 1980 thesis, Mounfield [9] showed that analog computers could be used to accomplish this task. At that time analog circuits were inexpensive compared to eight and sixteen bit microprocessors which were relatively new; personal computers were virtually unheard of outside hobbyist's circles. Analog technology does have drawbacks. Mounfield notes that measurement noise with frequencies greater than 50Hz and a noise-to-signal ratio of 0.10 can cause incorrect parameter identification¹. Analog circuits are constructed from electrical components, such as resistors and op-amps, and tuned to a specific plant. These components are limited in the amount of voltage they can withstand. This, in turn, limits the range of numerical values which the analog computer can handle and requires the user to have some *a priori* knowledge of parameter values. Modification is not quick and easy as this may require rewiring and retuning the circuit. Digital technology, on the other hand, offers a versatility to overcome these deficiencies. The available range of numerical values is much greater and modifications are as simple as recoding and recompiling a computer program. In the conclusion

¹ Mounfield, p. 37

to his thesis, Mounfield indicated that digital techniques might offer a cost or performance advantage. The ever dropping cost of eight, sixteen and now thirty-two bit microprocessors, but most especially the profusion of personal computers, has driven this speculation nearer to reality over a very brief period.

A survey of the literature on digital identification reveals that most studies use batch methods to produce parameter estimates. As examples, see the studies by Raisinghani [10] and Liu [7]. In a batch method, these parameter estimates are updated after n observations where n is a number set by the user. If the observation interval is Δt , this means no updated estimates are available over a time interval of $n\Delta t$. Thus, batch methods implicitly assume parameter values are constant or change very slowly over the period $n\Delta t$. Recursive algorithms are an alternative to batch processing. These methods update the parameter estimates after each observation. Batch processing is faster than recursive processing since estimates are computed only once, instead of n times, for n observations. However, if a change occurs in the system, we may not be able to wait the potential lag time of $n\Delta t$ to take corrective action. Hence, the chief problem in adaptive control is the real-time identification of plant parameters.

Several recursive algorithms for parameter estimation exist, many of them developed from batch methods. Among these, the recursive least squares method seems the most widely used. Graupe [5], who terms this method sequential least squares, gives three reasons: The recursive least squares algorithm

- is "stochastically convergent" even if the plant model "is unstable or [has] roots on the unit circle;"
- converges as fast or faster than any other algorithm;
- "is the most robust, namely, the most insensitive to computer round off errors (which are of major concern in short word-length computers, such as microcomputers, which are the hard-

ware most likely to be used in ... adaptive control situations), of all the [least squares] algorithm[s]."

As with other digital identification schemes, this method assumes that, for an m th order plant model, values for the corresponding m states are available either by measurement or computation. Graupe cites the square root algorithm as the only other least squares method which can compare in the above features to recursive least squares. However, he prefers recursive least squares because it requires less computational effort per recursion.

Several sources in the literature contain developments of the recursive least squares algorithm, among them Graupe [5], Eykhoff [4] and Ljung [8]. Least squares methods attempt to minimize a cost index. In the recursive least squares algorithm this index is formed from the measured and predicted values of a state of the plant. The algorithm may be condensed into two equations, the first to update the estimate and the second to update a matrix through which past observations influence the current estimate; a "memory matrix" for lack of a more descriptive term. The cost index is the final term of the first of these equations, $(\delta_i - U_i^T A_{i-1})$. Adopting a uniform notation, the above sources give these equations as:

$$A_i = A_{i-1} + (U_i^T P_{i-1} U_i + 1)^{-1} P_{i-1} U_i (\delta_i - U_i^T A_{i-1}) \quad (1)$$

and

$$P_i = P_{i-1} - (U_i^T P_{i-1} U_i + 1)^{-1} P_{i-1} U_i U_i^T P_{i-1} \quad (2)$$

where

i is the observation index

A_i is the $(m \times 1)$ vector of parameter estimates, $[a_{i,1}, a_{i,2}, \dots, a_{i,m}]^T$

m is the number of parameters

U_i is a $(m \times 1)$ vector of observations of the forcing function and the plant's derivatives, not including plant output

δ_i is the latest observation of the state of the plant which, with its estimate given by $U_i^T A_{i-1}$, forms the cost index

P_{i-1} is the previous update of the $(m \times m)$ "memory matrix."

It is worthwhile to note that this formulation requires no matrix inversion and that very little information (in fact, only the estimates, A_i and the "memory matrix," P_i) need be remembered between successive iterations. Additionally, no *a priori* information about the plant parameters is needed. To start the estimation process, Graupe and Ljung suggest $A_0 = [0, \dots, 0]^T$ and $P_0 = \epsilon I$, where ϵ is a large constant.

As mentioned above, equations (1) and (2) were derived as a recursive version of a batch method. Batch methodology assumes nearly constant parameters; all observations processed with equations (1) and (2) receive equal weights. To track time varying parameters, some method of decreasing the influence of old observations must be included lest the algorithm's tracking ability be retarded.

Eykhoff and Ljung both develop a method of data deweighting for the recursive least squares algorithm. This is done through the introduction of a "forgetting factor" which gradually discounts each observation. The resulting algorithm, hereafter referred to as Recursive Weighted Least Squares (RWLS), is summarized by two equations. Using the same symbols as before, these equations are:

$$A_i = A_{i-1} + (U_i^T P_{i-1} U_i + \phi)^{-1} P_{i-1} U_i (\delta_i - U_i^T A_{i-1}) \quad (3)$$

and

$$P_i = \frac{1}{\phi} (P_{i-1} - (U_i^T P_{i-1} U_i + \phi)^{-1} P_{i-1} U_i U_i^T P_{i-1}) \quad (4)$$

where ϕ is the forgetting factor in the range $0 < \phi \leq 1$. It is easy to see that equations (1) and (2) are special cases of equations (3) and (4), respectively, with $\phi = 1$. Andersson [1] and Brogan [2] suggest commonly used forgetting factors in the range 0.95 to 0.99. Most investigators use an algorithm to detect parameter changes, then adapt the forgetting factor within this range to track the

variations. For forgetting factors close to unity, such as these, Ljung gives an exponential-decay time constant of

$$\tau_{\phi} = \frac{1}{1 - \phi}.$$

This value is the number of recursions which occur before the weight of a given datum decreases to less than $e^{-1} \sim 36\%$ of the most recent observation. Eykhoff and Ljung each claim that this method is capable of tracking only "slowly" varying parameters. Neither defines "slowness" nor do they present reasons for avoiding values of ϕ less than 0.95. The faster forgetting provided by lower than usual forgetting factors could improve the algorithm's tracking abilities.

While not directly addressing these questions, two studies published in the past year propose refinements to the RWLS algorithm. They use different techniques to improve the algorithm's ability to track time varying parameters. Andersson's strategy is designed to handle sudden parameter jumps. Conventional methods of adapting the forgetting factor to track such changes would have to significantly decrease ϕ over only a few observations. This virtually empties the algorithm's memory of data collected before a change was sensed, essentially restarting the algorithm. If the detector raised a false alarm, that information is lost. Instead of running a single RWLS estimation algorithm and adjusting ϕ to meet tracking needs, Andersson proposes using multiple, parallel RWLS estimators. When a change is sensed, the forgetting factor is reduced by switching estimators. Thus, no information is completely lost until the algorithm is certain a change has occurred. This is the best feature of this method which Andersson calls adaptive forgetting through multiple models (AFMM).

Andersson never fully treats the subject of the extra computational burden imposed by his method. He mentions only that the computational requirement is approximately the number of estimators times the burden of a single RWLS estimator. From his investigation he concludes that the number of estimators is not critical, and suggests that this value be of the same order as the number of parameters which are estimated. Because he requires multiple estimators as well as supervisory logic,

the computational load will always be at least twice that of a single RWLS estimator with conventional ϕ adaptation. Andersson concedes that his method was not designed with gradual parameter changes in mind and admits "slowly changing parameters, ... can be difficult to handle with the ordinary AFMM algorithm."²

De Keyser's [3] approach is more conventional, with forgetting conditioned on the basis of a detection criterion. This detector measures the cross-covariance between the estimated process output, $U_i^T A_{i-1}$ and the difference between this value and the measured plant output, δ_i . During periods when the parameters are constant these values should be uncorrelated. De Keyser normalizes this cross-covariance by an estimate of its expected value to obtain his detection criterion. This criterion increases during parameter changes due to an increasing correlation between $U_i^T A_{i-1}$ and $(\delta_i - U_i^T A_{i-1})$. Based on the value of this criterion, forgetting is increased (e. g., the forgetting factor could be decreased in inverse proportion to the growing correlation) to track the changing parameter.

De Keyser doesn't neglect the loss of information through increased forgetting. He adds a probing component to his algorithm which stimulates the plant into producing non-zero derivatives. This action, too, is conditioned on the basis of the detection criterion, increased probing occurring simultaneous to increased forgetting. The addition of probing creates sufficiently rich measurements of the plant's states to supersede the information in observations which have been lost through forgetting. But because detection, forgetting and probing are based on a criterion which basically compares estimated to measured plant output, De Keyser's approach is from the overview of (adaptive) process control. ~~As~~ As such, his algorithm ignores some parameter variations if they won't affect the overall output of the plant.

² p. 1191.

2.1 Summary

An adaptive controller is an attractive alternative to conventional automatic control. This is because an adaptive controller needs little *a priori* information beyond a mathematical model of the plant. Physical parameter values are obtained through calculations based on the measurement of the plant's states and input. This is the chief problem in adaptive control and is called parameter identification.

Analog circuits have been successfully used to identify plant parameters. However, analog technology can be difficult to modify and has limited resolution due to component voltage restrictions. The decreasing cost of digital processors and the greater versatility of digital programming make digital techniques an attractive alternative to analog identification.

Most previous studies of digital identification have used batch processing to produce estimates. Such a method accumulates a batch of observations before using these data to update the estimates. Thus there is a time delay between successive estimates. A batch method will not indicate a parameter change occurring during this time delay. Corrective action at the time of the next update may be ineffective.

A recursive estimation scheme provides more timely estimate tracking through estimate updates with each observation. Recursive least squares is one of the most widely used recursive identification algorithms. It produces estimates which quickly converge, even if the plant model is unstable, and it is the algorithm least sensitive to computer round-off errors. To track time varying parameters a forgetting factor, in the range 0 to 1, may be introduced into the algorithm to deweight older data. In this thesis this method is referred to as recursive weighted least squares (RWLS). Commonly used forgetting factors are in the range 0.95 to 0.99. Eykhoff and Ljung each claim this method can track only "slowly" varying parameters. Neither defines "slowness" nor do they present

reasons for avoiding forgetting factors less than 0.95. Such a reduced value of ϕ would provide faster forgetting and perhaps improve the algorithm's tracking abilities.

Recent papers have proposed refinements to improve the tracking abilities of the RWLS algorithm. Andersson's adaptive forgetting through multiple models tracks sudden parameter jumps well. This algorithm was not designed to handle slowly changing parameters and may have trouble tracking them. Also, as the name implies, this algorithm essentially uses multiple RWLS estimators which increases the computational burden by at least a factor of two over conventional RWLS estimation. De Keyser's method, though simpler, ignores some parameter changes due to its detection criterion.

From this review of literature on digital identification, current needs seem to be:

- a delineation, if possible, between *slow* and *fast* parameter changes;
- a study of what effect forgetting factors less than 0.95 have on the estimates;

and, based on the results of the above investigations,

- a modification to the RWLS algorithm which is computationally simple and accurately tracks parameter changes whether they are rapid or would have little total effect on control of the process.

Addressing these points was the object of this investigation. The first two points are studied in the next chapter where the capabilities of the RWLS algorithm, given by equations (3) and (4), are examined. This forms the foundation for the estimate conditioning process, proposed and studied in the chapter, **Estimate Conditioning**. The final chapter contains conclusions about this process, as well as recommendations for future investigations.

3. Capabilities of Recursive Weighted Least Squares Estimation

This chapter covers an investigation of the behavior of the recursive weighted least squares algorithm (RWLS). Beginning with the selection of an initial test plant, the digital estimation of constant parameters by RWLS is briefly studied. Next, a RWLS formula claimed to track "slowly" varying parameters is presented. Tracking is accomplished through the introduction of a weighting factor which reduces the influence of a given measurement upon successive estimations. This formulation is explained and its ability to track time varying parameters is examined. This chapter concludes with a summary of the deficiencies of RWLS estimation and a sketch of a possible improvement. The results presented herein were obtained from computer simulations on an IBM3090. The source code was written in Waterloo FORTRAN/77 and is presented in Appendix A.

3.1 Test Plant Selection

A linear, second-order plant was selected with fairly typical characteristics. These were a spring coefficient of 100 units, a mass of 100 units and a damping coefficient of 60 units. Since this study used no experimental data, selection of an actual system of units was unnecessary. This should not mislead the reader. If one chooses to think of the spring coefficient in $\frac{\text{lbf}}{\text{in}}$, one must think of

displacement in inches and force in pounds-force. A similar argument holds if one thinks of acceleration in $\frac{m}{s^2}$. However, units of seconds for time, rather than a non-dimensional time such as

$$\frac{\text{time}}{\text{a characteristic time of the plant}}$$

are used throughout this presentation to accommodate the usual manner of thought. The computer simulation of this plant, and all plants described in this thesis, was carried out with an integration time step of 0.01s. The selected plant settles to within 10% of its final value after about 8s when its input is a step. It has a natural frequency of $1\frac{\text{rad}}{\text{s}}$.

To illustrate the plant response, a constant force of 20 units was applied. In the steady-state, the equation of motion becomes

$$\Sigma F_x = kx_{ss}$$

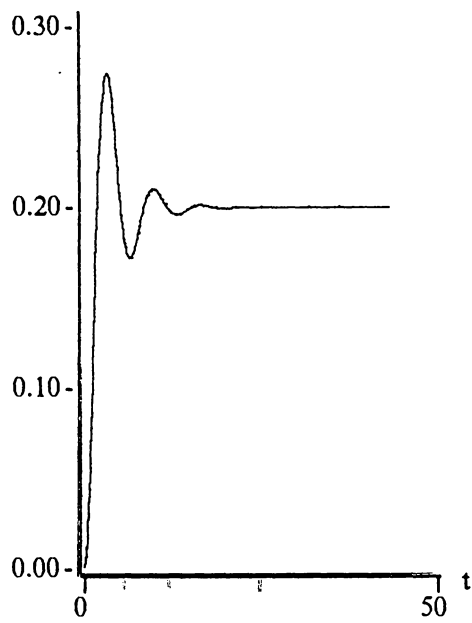
which for our test plant yields

$$x_{ss} = \frac{F}{k} = \frac{20}{100} = 0.20$$

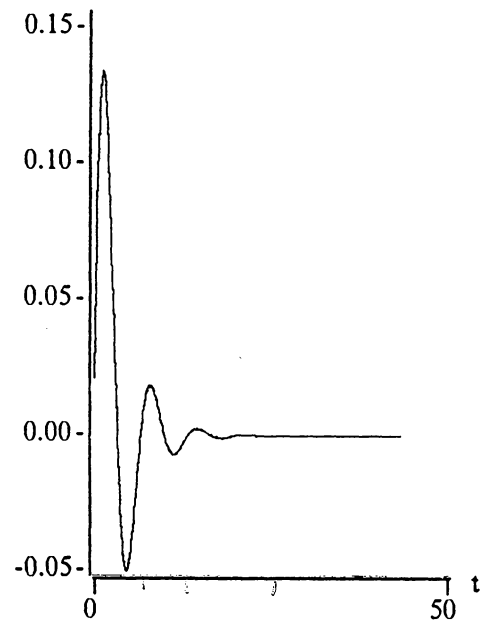
The step responses obtained for position, velocity and acceleration are plotted in Figure 2. Note that at about 8s, the plant position is within 10% of its final value. From these plots we see steady state is reached at about 25s, when both derivative variables are zero. Keeping these characteristics in mind, we can move to an examination of the estimation algorithm.

3.2 Estimation of Constant Parameters

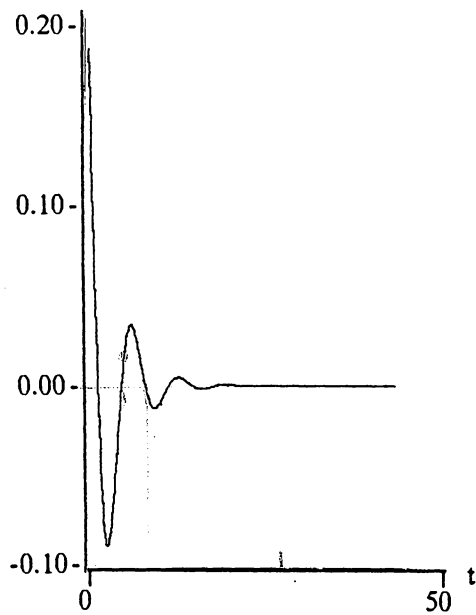
Least squares methods attempt to minimize a cost index. In the Recursive Weighted Least Squares Algorithm this index is the error between the measured and predicted values of a state of the plant.



a) Position



b) Velocity



c) Acceleration

Figure 2. Plant Response to Constant Step Input.

Recursive Weighted Least Squares estimation involves two equations, the first to update the estimate and the second to update a matrix through which past observations influence the current estimate; a "memory matrix" for lack of a more descriptive term. The cost index is the final term of the first of these equations, which are derived by Eykhoff [4] and Ljung [8] and may be presented as:

$$A_i = A_{i-1} + (U_i^T P_{i-1} U_i + 1)^{-1} P_{i-1} U_i (\delta_i - U_i^T A_{i-1}) \quad (1)$$

and

$$P_i = P_{i-1} - (U_i^T P_{i-1} U_i + 1)^{-1} P_{i-1} U_i U_i^T P_{i-1} \quad (2)$$

where

i is the observation index

A_i is the $(m \times 1)$ vector of parameter estimates, $[a_{i,1}, a_{i,2}, \dots, a_{i,m}]^T$

m is the number of parameters

U_i is a $(m \times 1)$ vector of observations of the forcing function and the necessary plant derivatives

δ_i is the latest observation of the state of the plant which, with its estimate given by $U_i^T A_{i-1}$, forms the cost index

P_{i-1} is the previous update of the $(m \times m)$ "memory matrix."

Hence the cost index to be minimized by adjusting the elements of A_i is $(\delta_i - U_i^T A_{i-1})$. Eykhoff gives an alternative form of (1) as:

$$A_i = A_{i-1} + P_i U_i (\delta_i - U_i^T A_{i-1}) \quad (1a)$$

The former presentation will be used.

The cost index $(\delta_i - U_i^T A_{i-1})$ drives the estimation update. If we take δ_i to be the lowest plant derivative, this yields an output-error formulation of equation (1). However, the equation of motion of a plant is usually written with the highest derivative alone on one side of the equation with

a coefficient of 1. Using this highest derivative as δ_1 produces an equation-error formulation of equation (1). Studying this more common manner first, the equation of motion for our test plant is

$$\ddot{x} = -\frac{d}{m}\dot{x} - \frac{k}{m}x + \frac{1}{m}F$$

where

- m is the plant mass
- d is its damping coefficient
- k is its spring coefficient
- F is the forcing function.

The parameters are thus $\frac{d}{m}$, $\frac{k}{m}$ and $\frac{1}{m}$. Hence, $\delta_1 = \ddot{x}$, $U_1 = [\dot{x}, x, F]^T$ and $A_1 \sim [-\frac{d}{m}, -\frac{k}{m}, \frac{1}{m}]^T$. Remember that A_1 is merely the estimate of the parameters, and thus is not always equal to them.

To start the estimation process, Graupe [5] and Ljung suggest $A_0 = [0, 0, 0]^T$ and $P_0 = \varepsilon I$, where ε is a large constant. A value of $\varepsilon = 5 \times 10^4$ was selected. Once again, a 20 unit step forcing function was simulated. An estimation interval of 0.05s was chosen and used through, the investigations of this thesis. Too, measurement noise was neglected. (This should not alarm the reader as several sources in the literature document techniques with which the RWLS algorithm may compensate for noise. For an example, see section 5.6 of Ljung.) Figure 3 contains plots of the parameter estimates, respectively normalized by their true values. The RWLS algorithm converges very quickly to the correct estimates, and holds them well.

Output-error formulations are generally believed to have superior noise-rejection properties. As was mentioned in the previous paragraph, measurement noise was neglected in this study. However, to be sure that there are no behavioral differences between output and equation error formulations, the above analysis was repeated for an output-error formulation. The equation of motion of the plant becomes

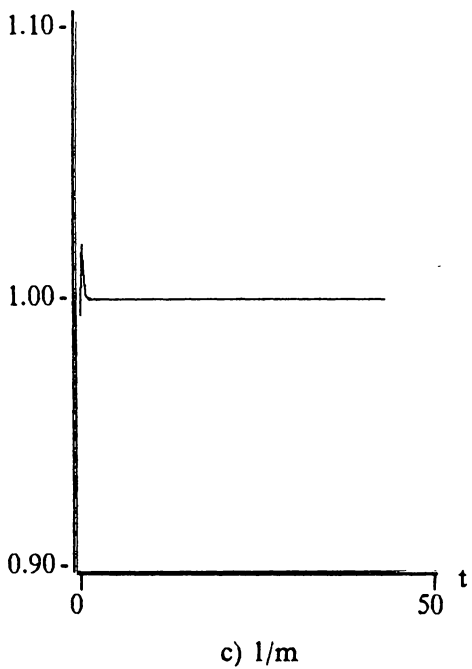
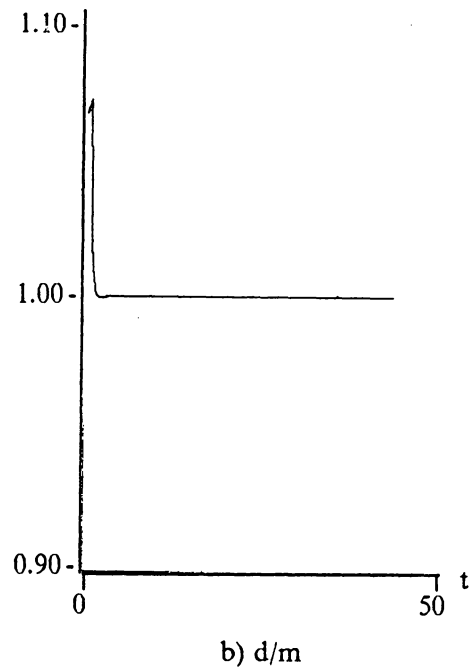
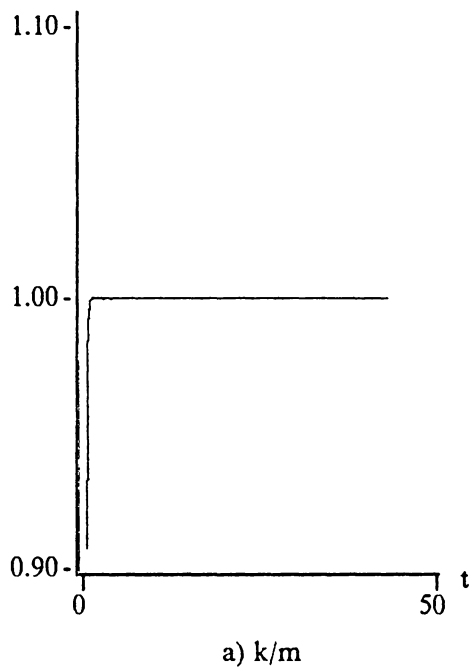


Figure 3. Estimate Response to Constant Step Input: Equation Error.

$$\ddot{x} = -\frac{m}{k}\ddot{x} - \frac{d}{k}\dot{x} + \frac{1}{k}F$$

in which case $\delta_i = x_i$, $U_i = [\ddot{x}, \dot{x}, F]^T$ and $A_i \sim [-\frac{m}{k}, -\frac{d}{k}, \frac{1}{k}]^T$. The forcing term and initial memory matrix were identical to that used in the equation error section. Parameter estimates for this formulation are graphed in Figure 4, again normalized by their respective true values. Because there is no great distinction between the behavior of the two formulations, as is expected for noise-free studies, equation error was used for the remainder of the investigation.

3.3 Estimation of Changing Parameters

To track time varying parameters, Eykhoff and Ljung introduce a "forgetting factor" and modify equations (1) and (2) to produce:

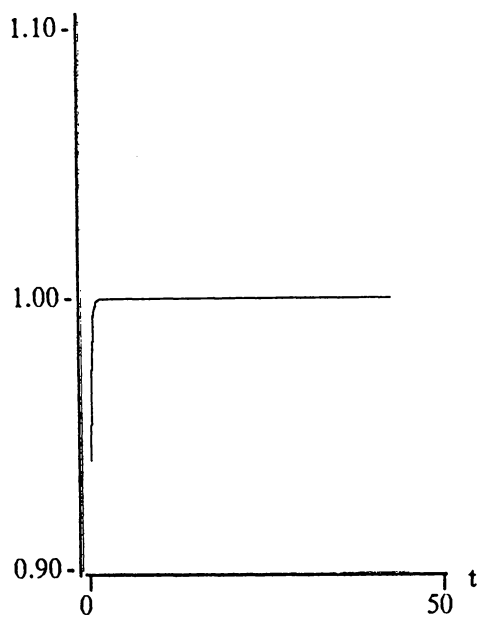
$$A_i = A_{i-1} + (U_i^T P_{i-1} U_i + \phi)^{-1} P_{i-1} U_i (\delta_i - U_i^T A_{i-1}) \quad (3)$$

and

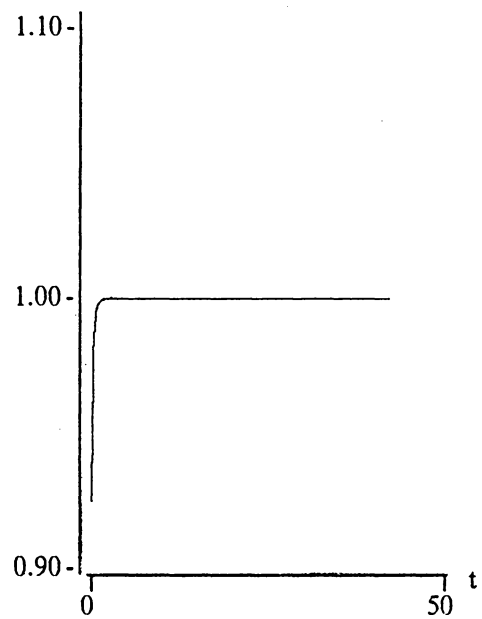
$$P_i = \frac{1}{\phi} (P_{i-1} - (U_i^T P_{i-1} U_i + \phi)^{-1} P_{i-1} U_i U_i^T P_{i-1}) \quad (4)$$

where ϕ is the "forgetting factor" in the range $0 < \phi \leq 1$. It is easy to see that equations (1) and (2) are special cases of equations (3) and (4), respectively, with $\phi = 1$.

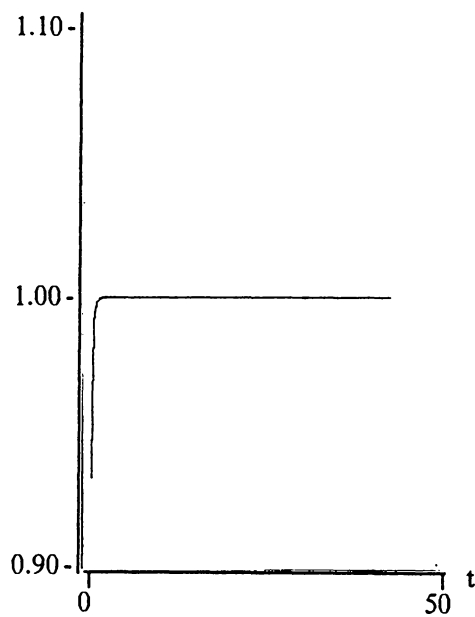
Mathematically, the forgetting factor increases the influence of the error term, $(\delta_i - U_i^T A_{i-1})$. Examine equation (3). For $0 < \phi < 1$, the value of the scalar inversion $(U_i^T P_{i-1} U_i + \phi)^{-1}$ is greater than it would have been if $\phi = 1$. This term in turn multiplies the error, increasing the correction applied to the previous estimates A_{i-1} . But the forgetting factor doesn't stop there. In equation (4), $0 < \phi < 1$ increases every element of P_i . This updated memory matrix becomes P_{i-1} when



a) m/k



b) d/k



c) $1/k$

Figure 4. Estimate Response to Constant Step Input: Output Error.

the next datum is taken and equation (3) is again applied. The increased P_{i-1} also multiplies the error, further increasing the correction applied to the estimates.

Practically, the effect of a past datum on current estimations can be approximated by

$$\phi^n = w$$

where

- ϕ is the forgetting factor
- n is the age of the datum - the number of estimations since it was taken
- w is the weight of the datum in current calculations.

Say we wanted to find the age at which a piece of the data is virtually forgotten by the RWLS algorithm. Brogan [2] suggests commonly used forgetting factors in the range 0.999 to 0.95. Let's choose $\phi = 0.95$ and define "virtually forgotten" as a weight of 0.01 (i. e. the influence of the datum has become 1% of its original effect). Solving the above equation, we find that about 90 subsequent estimations must be performed before the RWLS algorithm ignores a given datum. Repeating this calculation for several values of ϕ we obtain a more dramatic demonstration of the forgetting factor's effect. Figure 5 is a plot of weight by datum age for several forgetting factors. Notice that our approximation predicts that a datum processed with a low forgetting factor rapidly loses its influence on current estimates while the weight of a datum processed with $\phi = 1$ doesn't decrease.

Eykhoff and Ljung claim that the RWLS method is capable of tracking only slowly varying parameters. They do not quantify the property of *slowness*. Further, they leave forgetting factor selection entirely to the user. These voids left an obvious structure for a study of the estimation of time varying parameters: 1) quantify, if possible, *slow* change and 2) investigate the effect on estimates of different forgetting factors. But first, a preliminary study of equations (3) and (4) was conducted with $\phi = 0.95$. This was to insure that the loss of information through forgetting would not adversely affect the estimates.

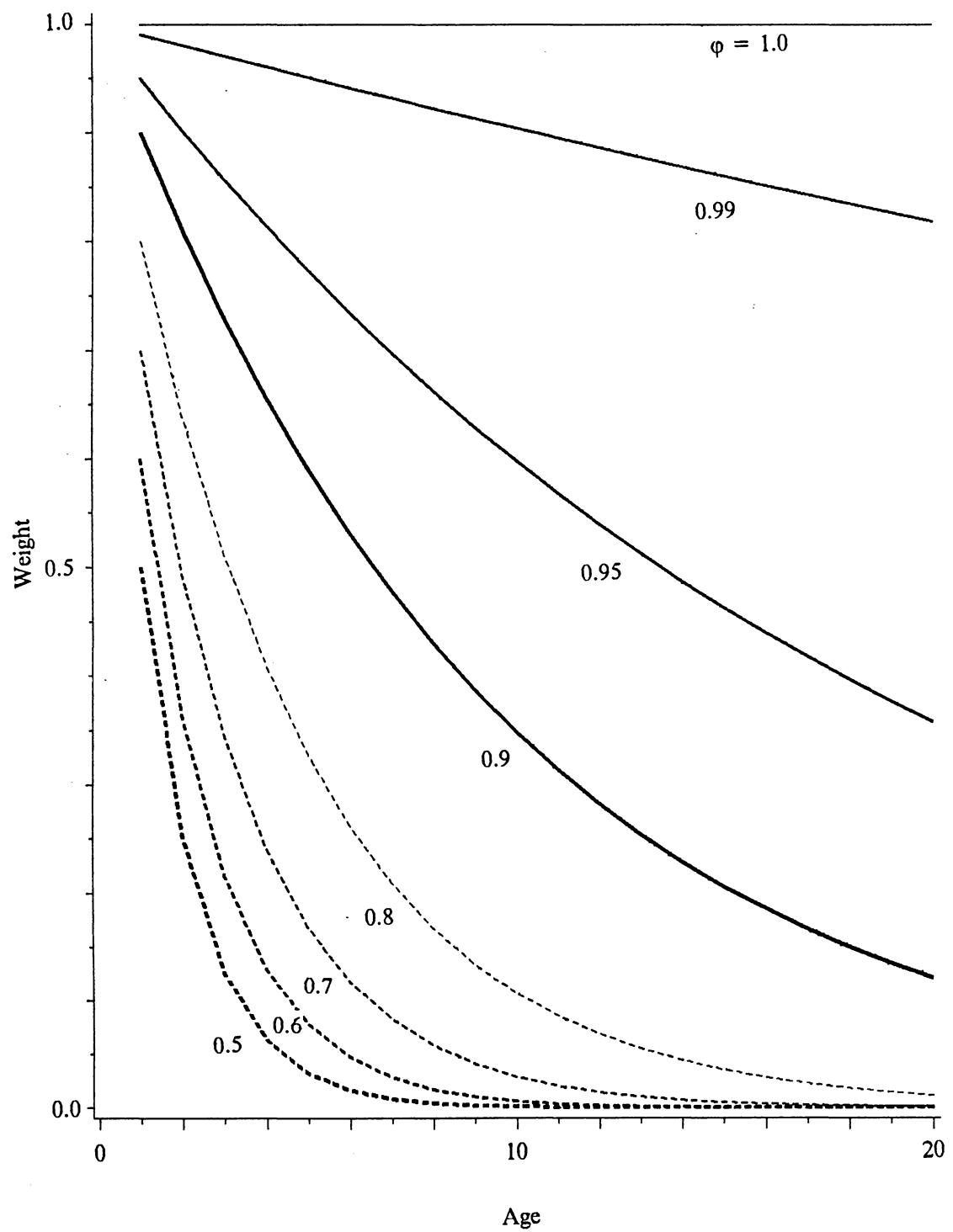


Figure 5. Reduced Weight with Age by Forgetting Factor.

The forcing function and initial memory matrix were kept at their previous values: $F = 20$ units and $P_0 = 5 \times 10^4 I$. The estimates produced by equations (3) and (4) with $\phi = 0.95$ are plotted in Figure 6, normalized by their true values. As was the case with $\phi = 1$, the RWLS algorithm quickly converges to the correct estimates. But notice the jump in the normalized estimates beginning at about 25s. This is approximately the same time at which the plant reaches steady-state.

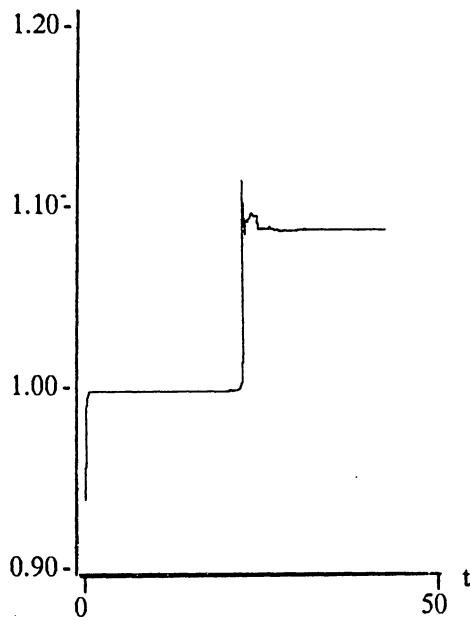
As the plant reaches steady-state, the derivative terms in the equation of motion become zero and the position reaches a constant value. Thus the error term in equation (3), $(\delta_i - U_i^T A_{i-1})$, becomes

$$0 - y_{ss} \frac{k}{m} + F \frac{1}{m}$$

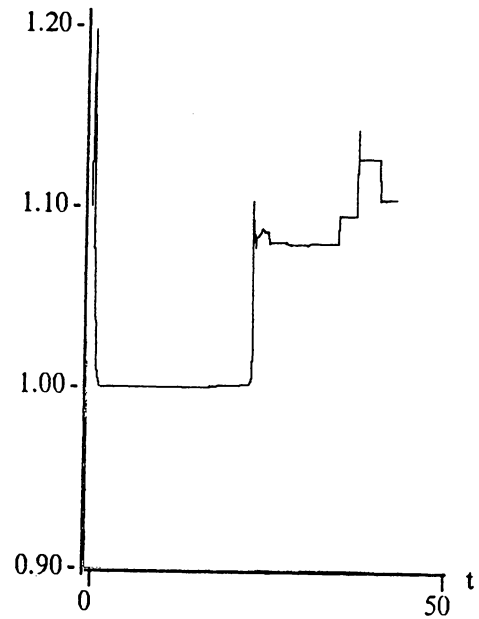
The estimates are forced away from their correct values by this non-zero error term whose influence has been increased by $0 < \phi < 1$. De Keyser [3] notes this phenomenon in his paper. He mentions two possible solutions: either use $\phi = 1$ so that the gain term, $(U_i^T P_{i-1} U_i + \phi)^{-1} P_{i-1} U_i$ in equation (3), is so small that the parameters are frozen, or assure that the plant continuously produces useful data. By useful data, De Keyser probably means non-zero derivatives. This would require preventing the plant from ever reaching steady-state. Another alternative is to avoid estimation altogether when the plant is not producing sufficient information, as indicated by a zero value for the highest plant derivative.

De Keyser's solution to this dilemma is to provide a probing component in his algorithm. This component is used in conjunction with $\phi < 1$ and excites the plant sufficiently to produce continuous derivatives. As this problem has been solved, it will not be addressed here. Instead, for the remainder of this investigation, the plant was caused to produce derivatives by enriching the forcing function with a random component.

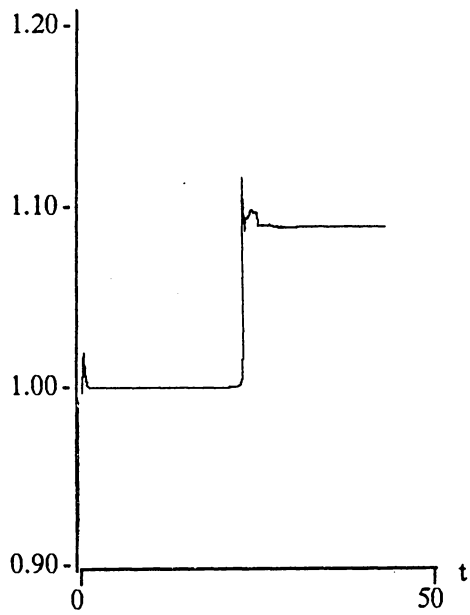
The enriched forcing function consisted of a constant term and additive noise. The noise had an amplitude of nine-tenths the average value of the forcing function. Lower amplitude noise was tested and it was found that an additive noise amplitude approximately 10% that of the forcing



a) k/m



b) d/m



c) $1/m$

Figure 6. Estimate Response to Constant Step Input with Forgetting Factor = 0.95.

function's average value was sufficient to avoid a jump in estimates for this plant. The much larger amplitude was selected to insure that the plant would produce sufficient information to avoid the estimate jump. Too, De Keyser shows that a constant, low level of his probing technique can lead to a remarkable improvement in estimate accuracy³. Figure 7 shows a forcing function which has a mean value of 20 units with additive noise of 18 units amplitude. This was the forcing function used for the remaining studies in this chapter. The plant response to this input is plotted in Figure 8. Parameter estimates produced by RWLS for this input are diagrammed for $\phi = 1$ and $\phi = 0.95$ in Figure 9, normalized by their true values.

The RWLS algorithm needs sufficient information to assure the stability of parameter estimates. This information is lost through the use of a forgetting factor less than one. With such information replaced by a noisy forcing function, the investigation of time varying parameter estimates was begun in earnest.

3.3.1 Quantification of Slow Parameter Change

To quantify *slowness*, the ability of the RWLS algorithm to track a single varying parameter was studied. This was done by varying that parameter at several rates and observing the estimates. But, which parameter should be varied? The noisy input caused plant dynamics which might couple with a changing parameter to produce poor estimates. Most plants act as filters with respect to any noise in the forcing function. To exploit this property, it was decided to vary the parameter corresponding to the lowest derivative in the plant; i. e. the parameter $\frac{k}{m}$ which multiplies the plant output, y . The equation error formulation involves mass in each of the three parameters. To vary a single parameter and exploit plant filtering it was necessary to vary the spring coefficient. For simplicity, a ramping step change in the spring coefficient, such as the one shown in Figure 10, was used. The suggestion of 0.95 as a commonly used forgetting factor was kept.

³ De Keyser, p. 76

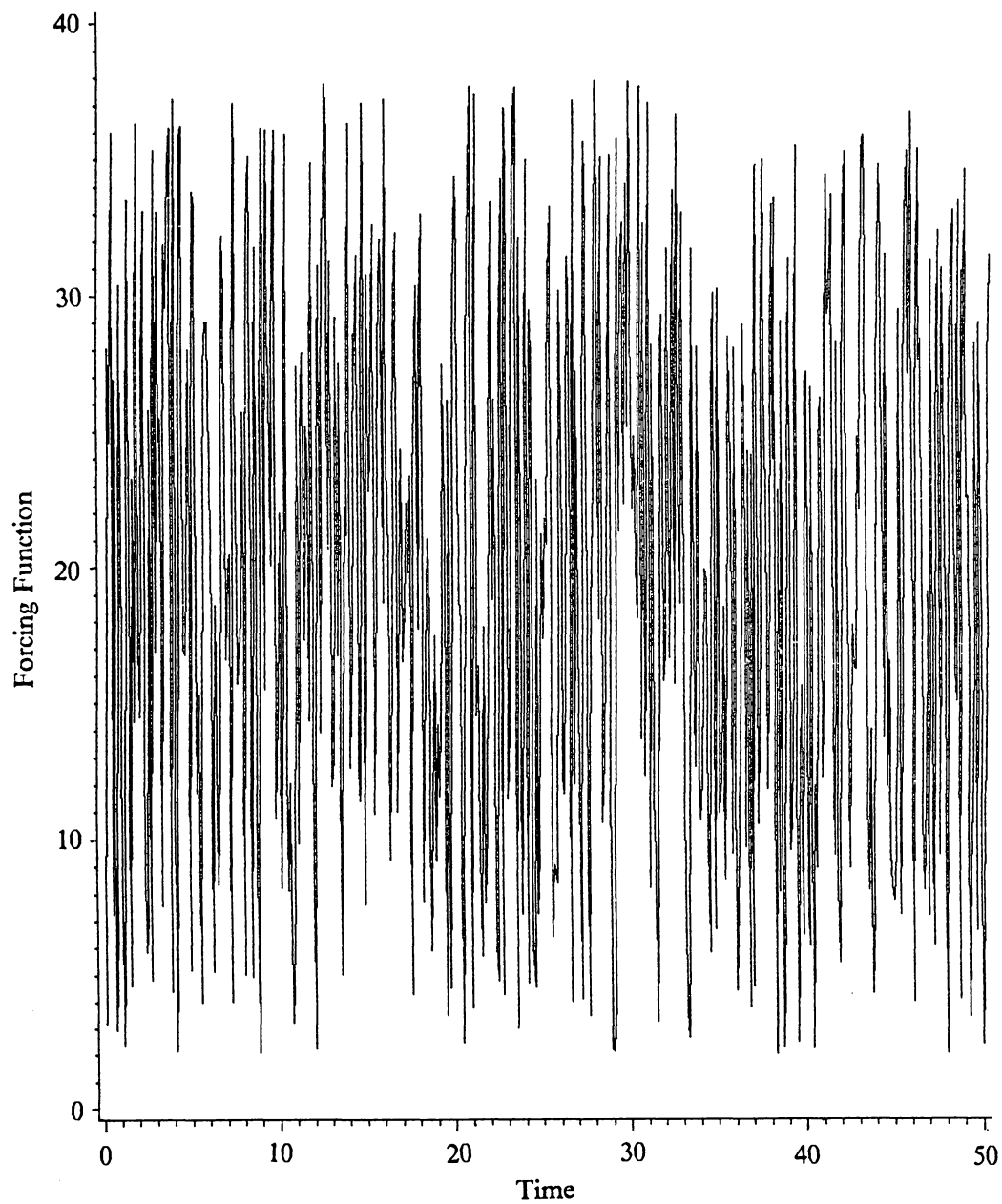
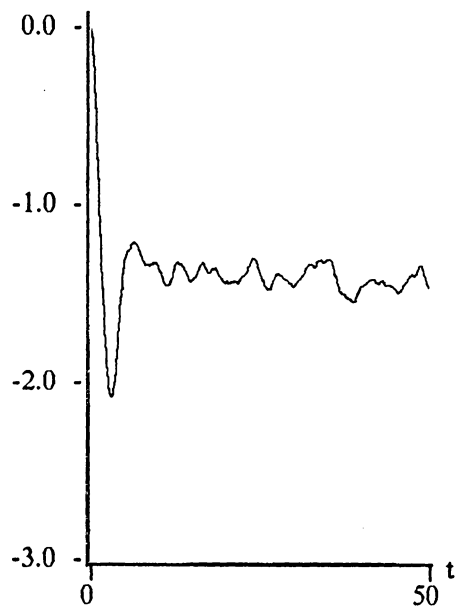
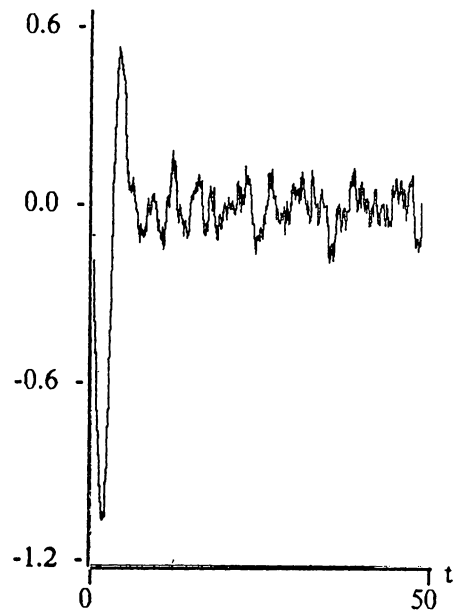


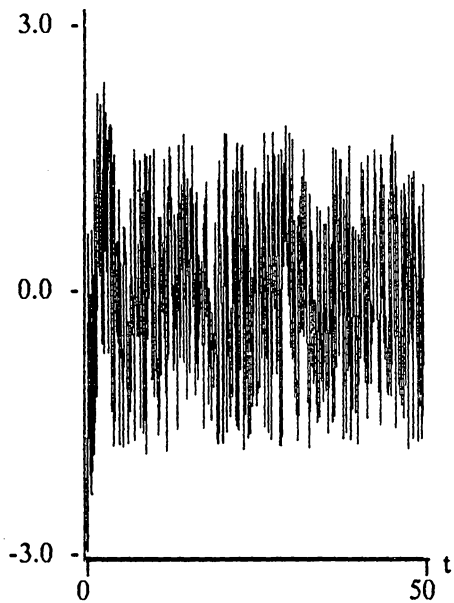
Figure 7. Noisy Forcing Function.



a) Position



b) Velocity



c) Acceleration

Figure 8. Plant Response to Noisy Forcing Function.

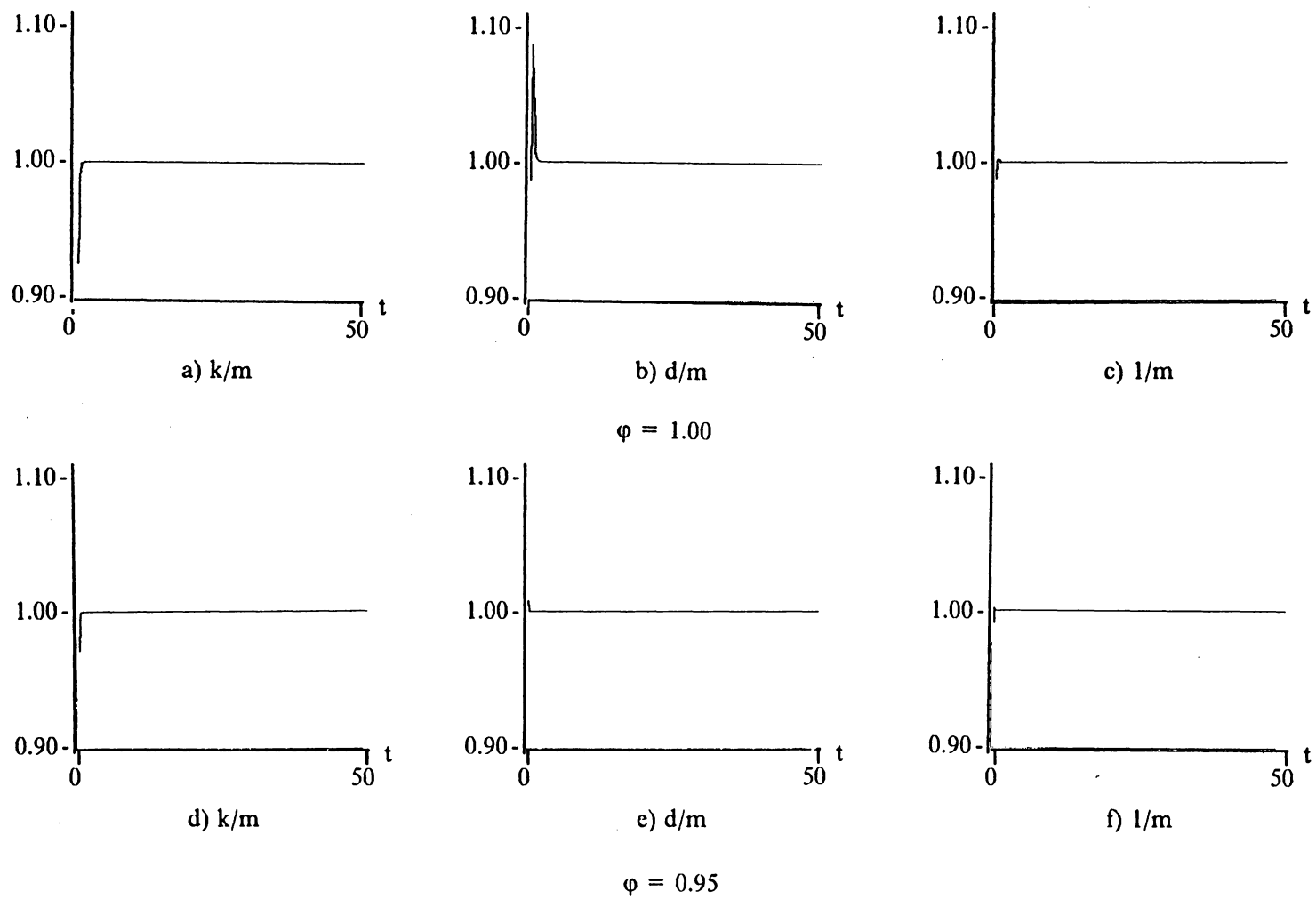


Figure 9. Estimate Response to Noisy Forcing Function.

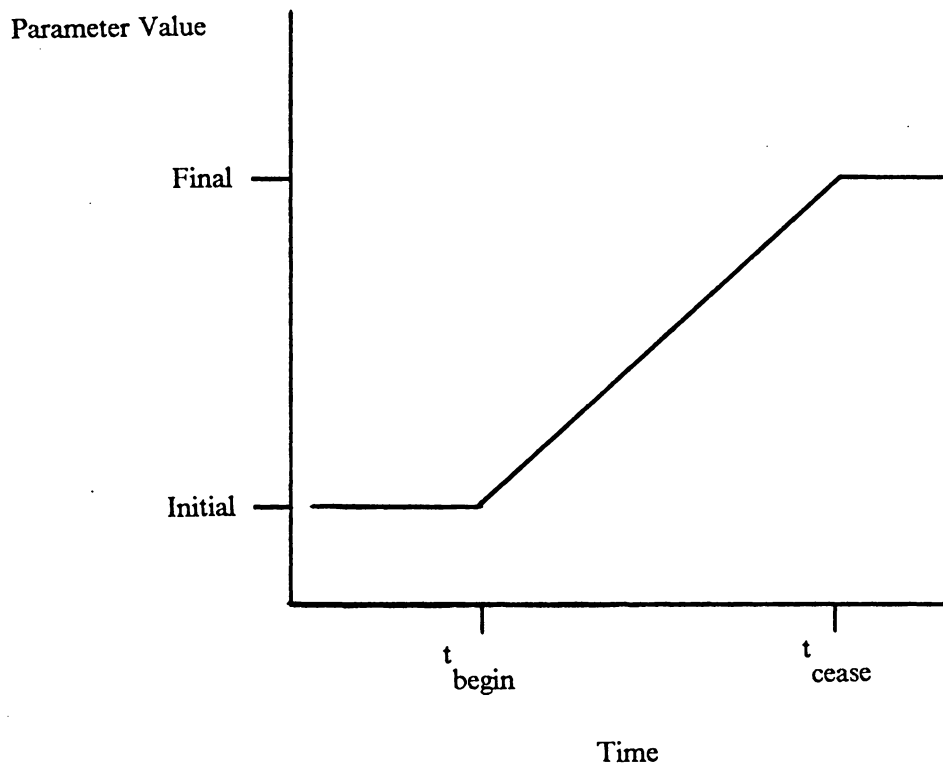


Figure 10. Ramping-Step Change of a Parameter.

To avoid repetitious re-selection of a proper integration time step, it was decided to fix the period over which the parameter would change. Variation of the spring coefficient began after about one and one-half settling times of the plant, at 12s, to minimize errors which might be induced by plant start-up characteristics. Transient behavior due to the ramping would need to die before the ramp ended. An ending time of 48s was chosen to yield better than four plant settling periods over which the variation would occur. It was anticipated that this period, from 12s to 48s, would allow time for the plant-estimator combination to reach a steady condition.

The rate of parameter change was varied from a very small to a very large value. Since the period of the variation was fixed, the rate of change covered this required range by altering the amount of the change in the spring coefficient on a percentage basis. Parameter variations from 0.1% to 1000% over the established four settling periods of the plant should respectively satisfy any reasonable definitions of slow and fast. Selected quantities of the percentage change in the spring coefficient were chosen to span this range. The three parameter estimates, $\frac{k}{m}$, $\frac{d}{m}$ and $\frac{1}{m}$ were observed during each of these rates of change.

Parameter estimates, normalized by their true values, for the changes of the spring coefficient studied in this section are plotted in Figure 11 through Figure 13. These, and all remaining plots, have been clipped to fit within the scales shown in each respective figure. From Figure 11 notice that the RWLS algorithm is able to track the time varying parameter to within 5% of its true value for variations of less than 100% over the 36s period of change. This ability is degraded at the greatest changes examined (500% and 1000%), the worst tracking occurring within the first plant settling period (8s) after the ramping begins.

Normalized estimates of the parameter $\frac{d}{m}$ are plotted in Figure 12. This parameter seems more sensitive to the change in spring coefficient. The RWLS algorithm estimates this parameter to within 10% of its true value for spring coefficient variations of less than 10% over four plant settling periods. Estimates of $\frac{1}{m}$ fare slightly better, as shown by the normalized plots of Figure 13. This

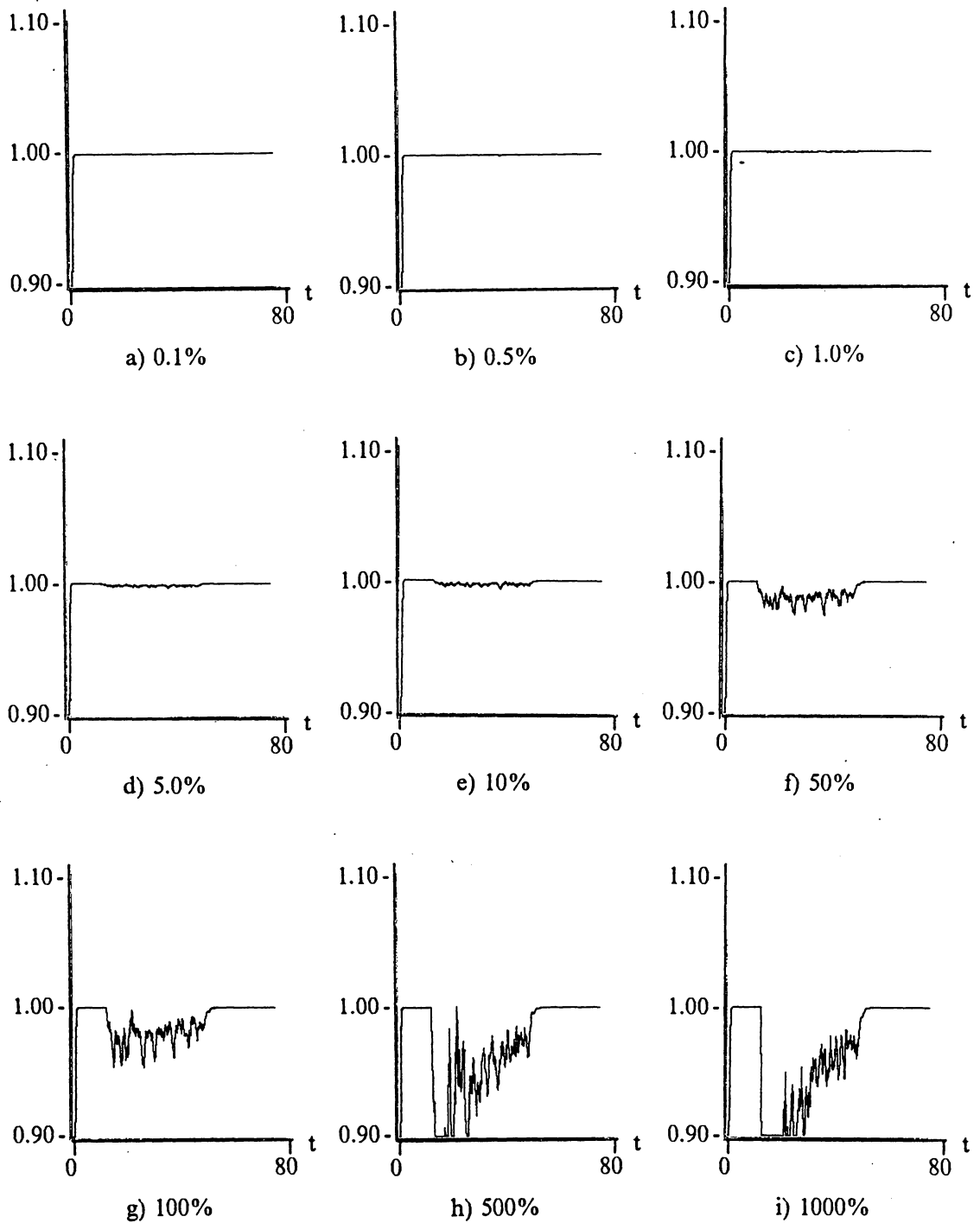


Figure 11. Normalized Estimates of k/m for Several Spring Coefficient Changes.

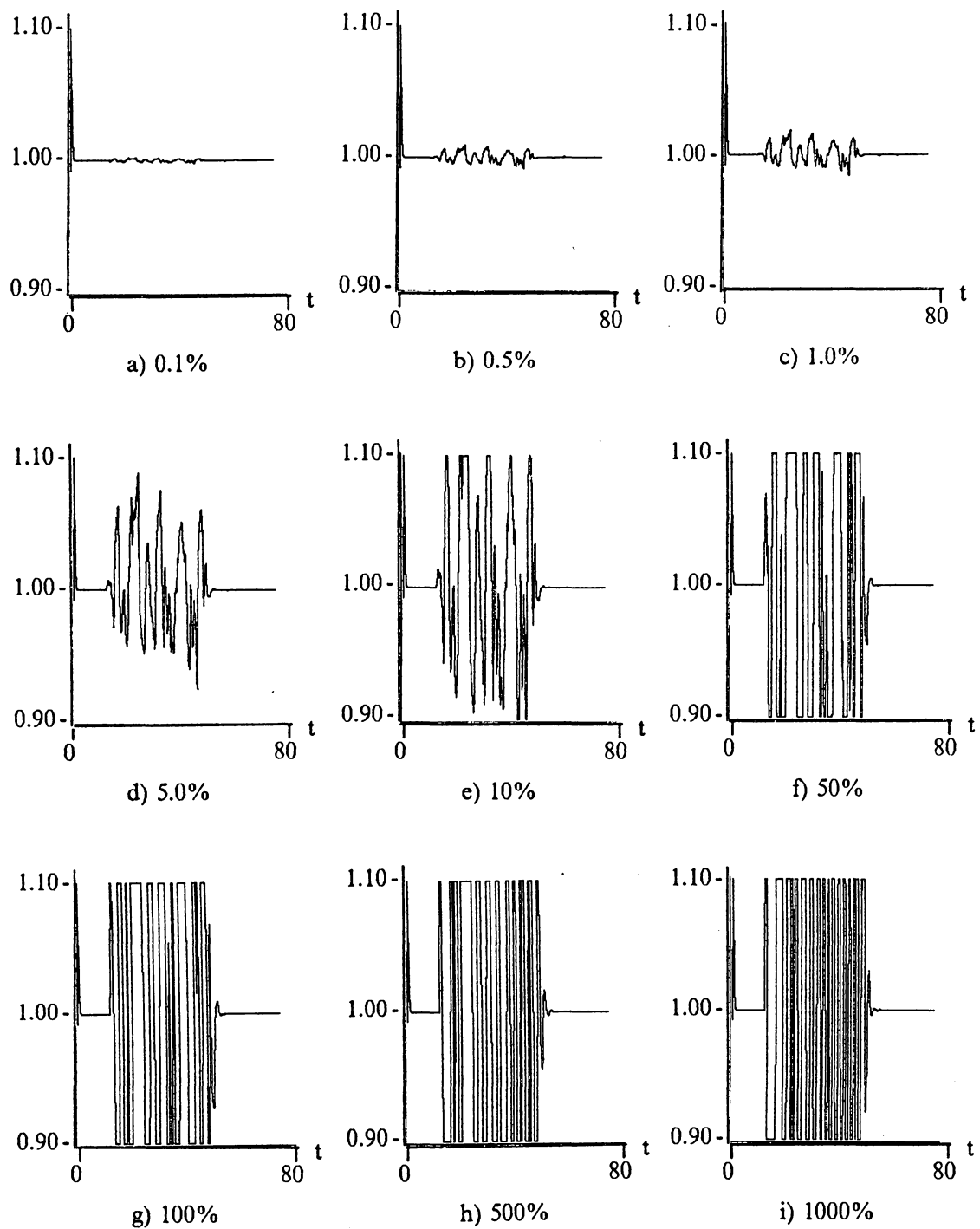


Figure 12. Normalized Estimates of d/m for Several Spring Coefficient Changes.

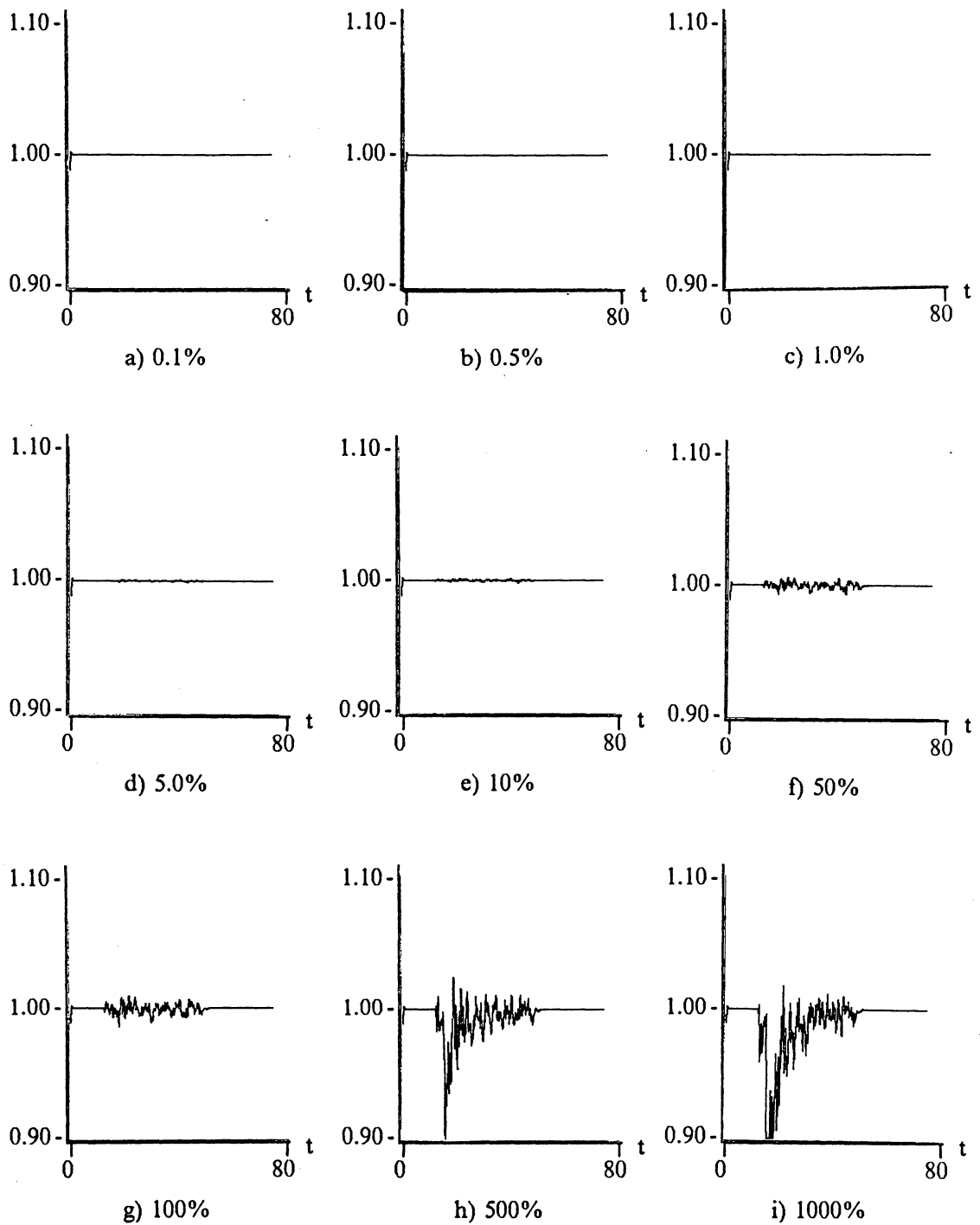


Figure 13. Normalized Estimates of $1/m$ for Several Spring Coefficient Changes.

parameter is accurately estimated by the RWLS algorithm to within 1.5% of its true value for spring coefficient changes of less than 100%.

These time invariant parameter estimates exhibit oscillations about their true values during the spring coefficient variation in every test case. These oscillations increase in amplitude with increasing parameter change. As the spring coefficient reaches the end of the ramping-step at 48s, estimates of all parameters rapidly return to their correct values.

Table 1 lists the maximum error of the estimate for each parameter and each variation of the spring coefficient studied. At none of the observed changes in spring coefficient do all three estimates suddenly become less accurate. This table, and the figures from which it was compiled, show that there is no clear division, even in our simple second-order plant, between *slow* and *fast* parameter variations. Such a definition would seem to depend on the plant in question and the accuracy desired in estimates of time-invariant parameters.

3.3.2 Forgetting Factor Effects

From the coverage in the literature, forgetting factor selection appears to depend on experience. This part of the investigation sought to discover what effect the forgetting factor might have on estimates and their accuracies during a parameter change.

The plant input was the noisy forcing function diagrammed in Figure 7. The spring coefficient was changed 10% over the established 12s to 48s period. This rate was chosen because it produced accurate estimates with $\phi = 0.95$. However, these estimates were not so accurate that accuracy changes due to forgetting factor variations would be undetectable.

The forgetting factor was varied from 1.0 to 0.5, and the parameter estimates, $\frac{k}{m}$, $\frac{d}{m}$ and $\frac{1}{m}$, were observed. By definition, ϕ may not exceed 1.0. The lower value was chosen to provide rapid

Table 1. Maximum Error of Parameter Estimates for Several Spring Coefficient Changes.

Percentage Change of Spring Coefficient	Parameter		
	$\frac{k}{m}$	$\frac{d}{m}$	$\frac{1}{m}$
0.1	< 0.5%	< 0.5%	< 0.5%
0.5	< 0.5%	1.0%	< 0.5%
1.0	< 0.5%	2.0%	< 0.5%
5.0	< 0.5%	9.0%	< 0.5%
10.0	1.0%	> 10.0%	< 0.5%
50.0	2.5%	> 10.0%	1.0%
100.0	5.0%	> 10.0%	1.5%
500.0	> 10.0%	> 10.0%	10.0%
1000.0	> 10.0%	> 10.0%	> 10.0%

memory loss to the RWLS algorithm. Recall our approximation of data deweighting, shown in Figure 5. This approximation predicts that $\phi = 0.5$ produces virtual ignorance of a datum after only 8 subsequent estimations; very quick forgetting indeed!

Figure 14 through Figure 16 contain plots of the normalized parameter estimates for the forgetting factor values studied. These diagrams exhibit surprising improvements in the estimates as ϕ was reduced from 1 to 0.95 or less. For instance, Figure 14 shows that the RWLS algorithm was able to track the change in spring coefficient to within 5.5% when $\phi = 0.999$; reducing the forgetting factor to 0.95 improved tracking to within 0.3%. Further, marginal gains in tracking ability were made when ϕ was reduced below 0.95.

The behavior of the estimate of $\frac{d}{m}$ was only slightly different. Figure 15 shows that this estimate began oscillating about its true value when the forgetting factor was reduced from 0.999 to 0.99. This was accompanied by a loss of estimate accuracy. Reducing ϕ reduced the maximum amplitude of the oscillations, hence improving the accuracy of the estimate. Figure 15c shows that, with $\phi = 0.99$, the estimate of $\frac{d}{m}$ was within 23% of its true value. The estimate error decreased to within 18.5% when ϕ was reduced to 0.95. Greater improvement was made by lowering the forgetting factor to 0.8: this produced estimates of $\frac{d}{m}$ accurate to within 9.0% of their true value. The oscillations in this estimate were reduced very little when the forgetting factor was less than 0.8.

The estimates of $\frac{1}{m}$ were always within 1% of their true value, as exhibited by Figure 16. Even though these estimates were highly accurate, they did improve when ϕ was less than 0.999. Plot c of Figure 16 shows this estimate to have been within 0.3% of its true value when the forgetting factor was 0.99. Reducing ϕ to 0.8 further improved this accuracy to within 0.1%.

Table 2 is a concise record of the maximum error for estimates of each parameter for each of the forgetting factors studied. It shows the dramatic improvement in estimates achieved by reducing ϕ below 0.95.

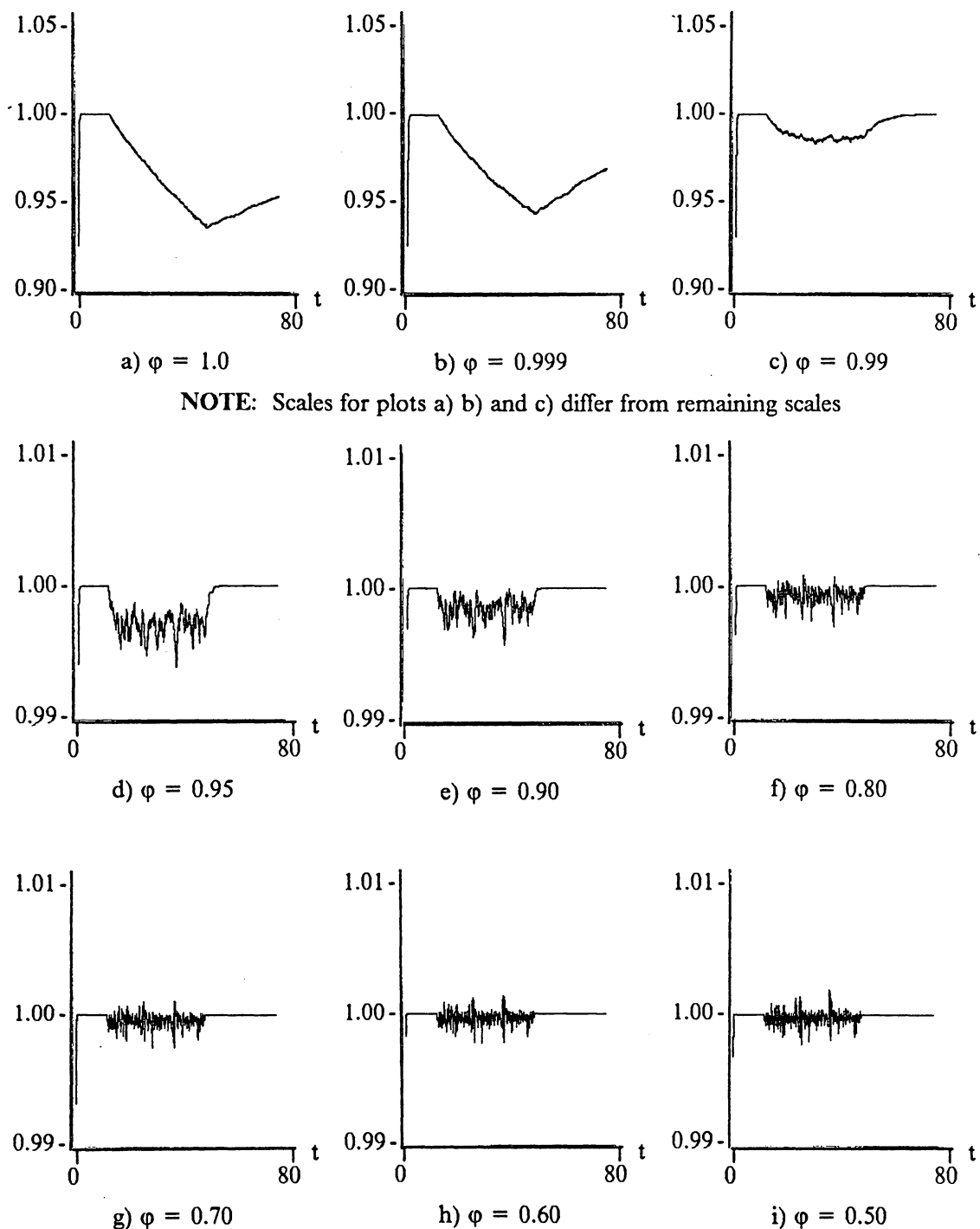


Figure 14. Normalized Estimates of k/m for Several Forgetting Factors.

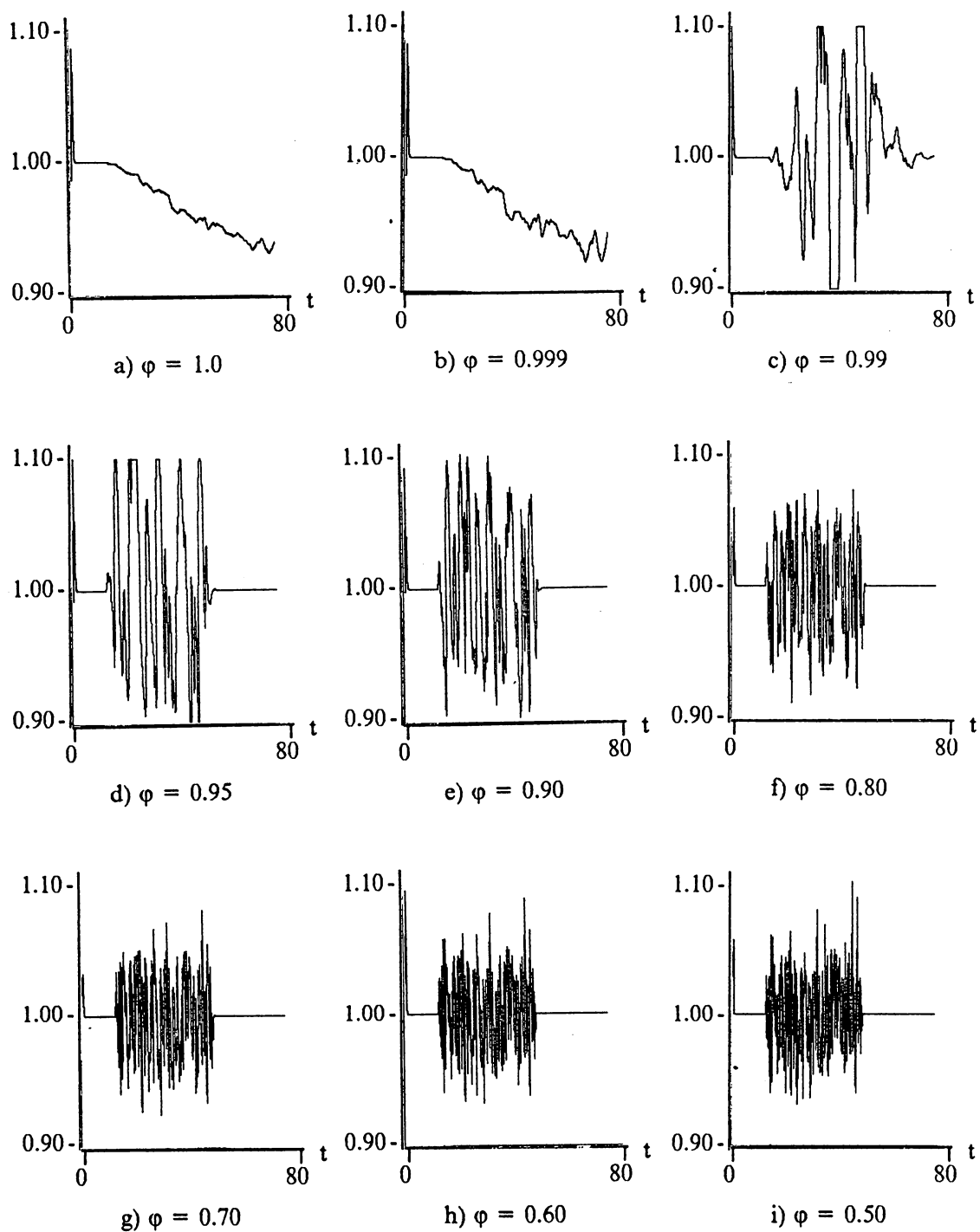


Figure 15. Normalized Estimates of d/m for Several Forgetting Factors.

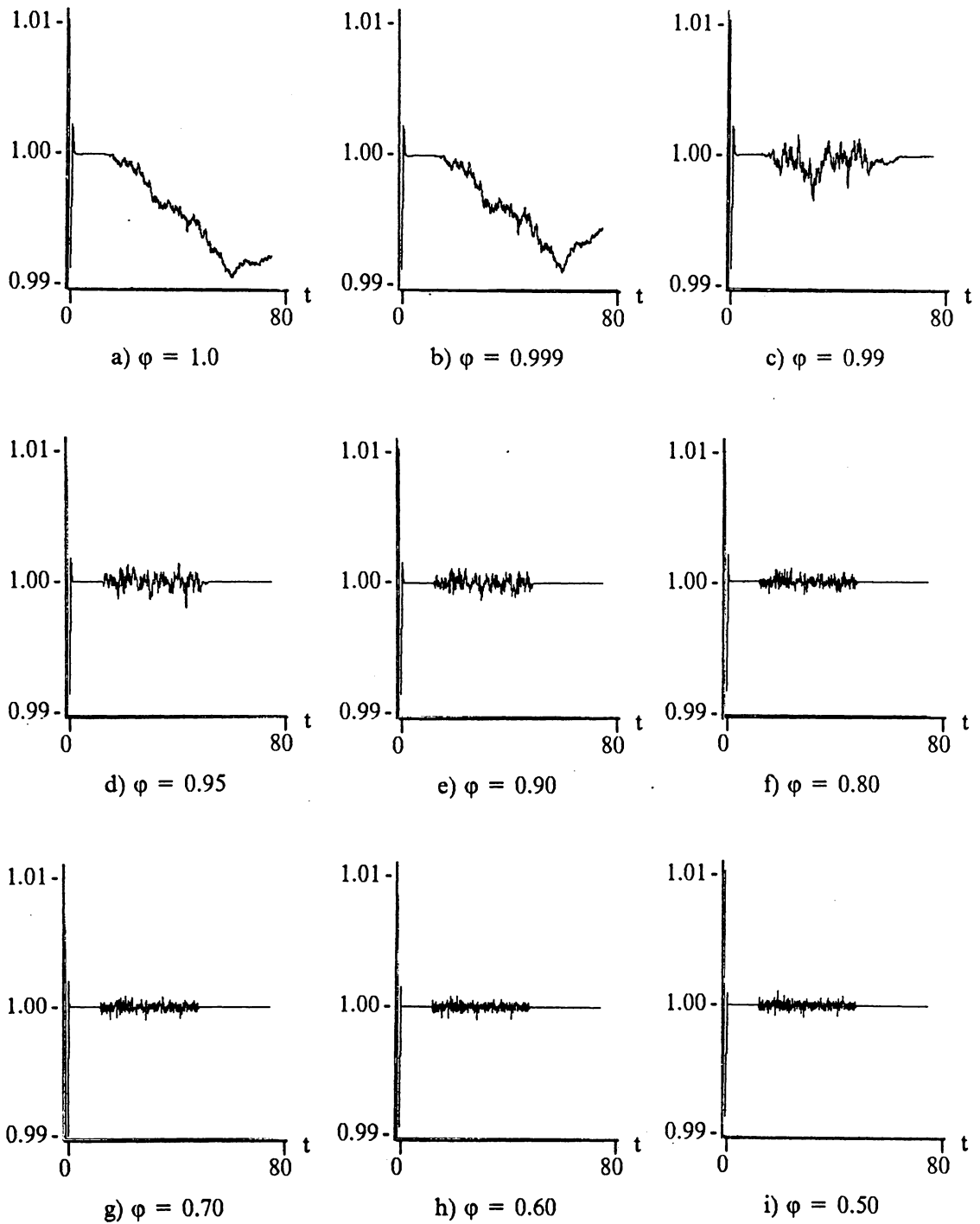


Figure 16. Normalized Estimates of $1/m$ for Several Forgetting Factors.

Table 2. Maximum Error of Parameter Estimates for Several Forgetting Factors.

Forgetting Factor ϕ	Parameter		
	$\frac{k}{m}$	$\frac{d}{m}$	$\frac{1}{m}$
1.0	6.0%	7.0%	0.95%
0.999	5.5%	8.0%	0.95%
0.99	2.0%	23.0%	0.35%
0.95	0.3%	18.5%	0.2%
0.90	0.2%	13.0%	0.1%
0.80	0.15%	9.0%	0.1%
0.70	0.15%	8.0%	0.1%
0.60	0.125%	9.0%	0.1%
0.50	0.125%	10.0%	0.1%

Forgetting factors less than the commonly used values described by Brogan produced superior estimates. The ability of the RWLS algorithm to track time varying parameters was improved to within 0.5% of the true value. Further, oscillations in estimates of time invariant parameters were reduced for $\phi < 0.95$, thus improving the accuracy of those estimates. Finally, for $\phi \leq 0.95$, estimates of all three parameters promptly returned to their correct values once the spring coefficient had reached its new, constant value.

3.4 Summary of Behavior

In the above study we have seen three major characteristics of recursive weighted least squares estimation:

- When using a forgetting factor less than 1, the plant must produce sufficient, useful information. This problem has been solved by De Keyser's addition of a probing component to his algorithm which forces the plant to produce useful information through its derivatives. This investigation did not duplicate this existing solution; instead, the forcing function was enriched with additive noise, causing the plant to produce non-zero derivatives.
- There is no clear delineation of a *slow* parameter change. It seems such a distinction must be made in the context of the plant in question and the accuracy desired in the estimates of time invariant parameters.
- Reducing the forgetting factor below commonly used values in the range $0.95 \leq \phi \leq 0.999$ can improve the accuracy of all estimates during a parameter variation. Such a reduction additionally promotes the rapid return of estimates to their correct values once the parameter change has ceased. However, any attempt to use an unusually low forgetting factor must take

into consideration the oscillations this may induce in the estimates of time invariant parameters.

Based on these characteristics we see that, if limited to RWLS estimation alone, a compromise forgetting factor based on user-supplied criteria of tracking ability and estimate accuracy will be necessary. Selection of a compromise forgetting factor would be plant dependant; it is desirable to spare the control engineer this choice. Using a single forgetting factor, our objective is then to track the most rapidly changing parameter while maintaining smooth estimates of other parameters. An added objective is, if more than one parameter is changing, to avoid total frustration of any ability the RWLS algorithm may have to track these multiple parameter variations. The next chapter presents such an improvement.

4. Estimate Conditioning

In the previous chapter, the behavior of RWLS estimation was examined. Forgetting factors near unity produce accurate estimates of constant parameters. But when a parameter changes, the estimates begin to wander from their correct values, recovering slowly only when such change stops. Forgetting factors less than 0.95 greatly improve the algorithm's ability to track time varying parameters, as well as speeding the return of all estimates to their correct values once the variation ceases. However, reducing the forgetting factor induces the time invariant parameter estimates to oscillate about their true values.

These opposing characteristics, reduced tracking error versus oscillatory estimates, seem to call for a compromise forgetting factor. Instead, this investigation chose to exploit the advantages of a reduced forgetting factor while further conditioning the estimates to smooth the oscillations which may be induced by a low forgetting factor.

Details of this modification and its behavior are contained in this chapter. The conditioning process itself is described first. The logic is outlined then broken into components and diagrammed in a flow-chart. Finally, the modification's behavior is studied with side-by-side comparisons of conditioned and unconditioned estimates for several plants.

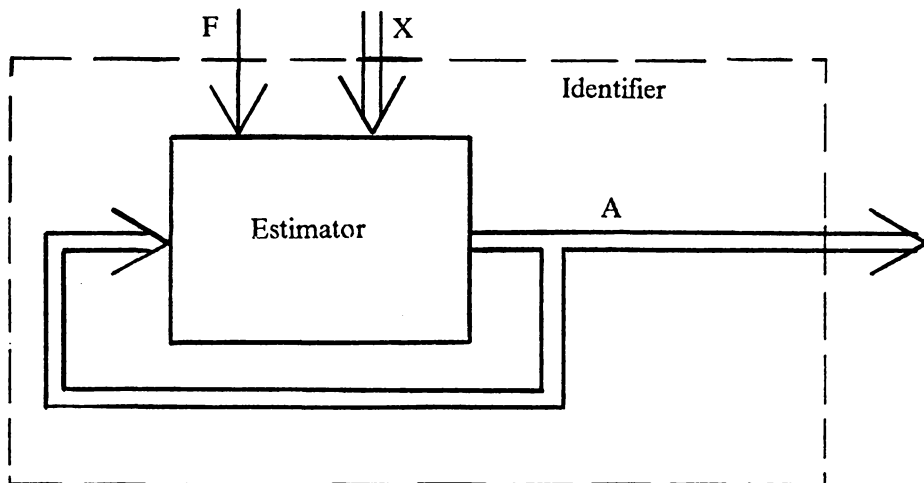
4.1 Conditioning Process Logic

Conditioning is a post-process to estimation. Signal flow in the RWLS algorithm may be diagrammed as in Figure 17a, while estimate conditioning is shown in Figure 17b. Based on observations of estimate behavior studied in the previous chapter, it was decided to condition RWLS estimates produced with $\phi = 0.8$ by, basically, averaging them. A forgetting factor of 0.8 was selected because it produced estimates which tracked time-varying parameters well, while somewhat reducing the errors in non-time-varying parameter estimates (see Table 2). Averaging was selected as the basic conditioning process because it smooths the estimates while still allowing them to vary with time.

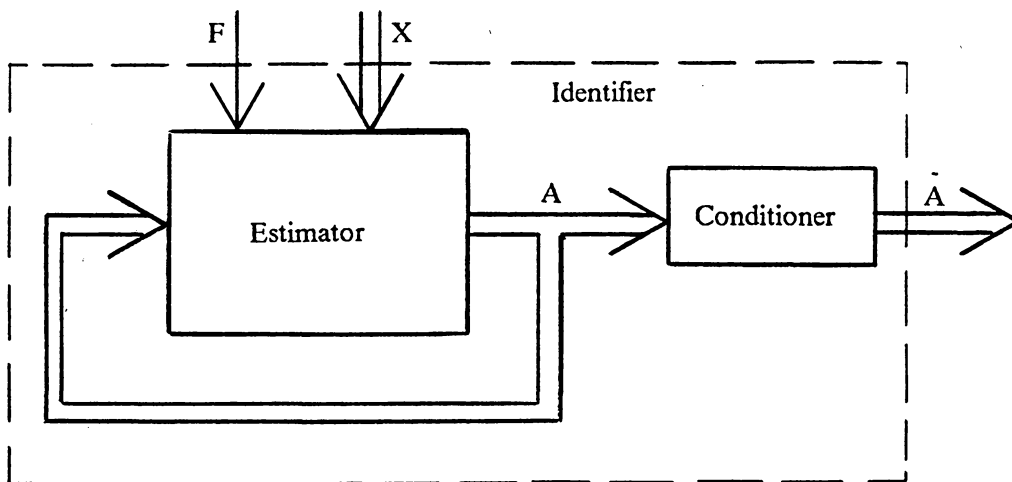
There are three major components of the conditioning algorithm. One to establish steady values, another to condition the estimates by averaging those which aren't currently changing and a third to detect the start and end of a parameter change. Steady values are established by averaging the initial RWLS estimates over a short time period. The user selects this time period, based on his knowledge of the plant, as one during which the parameters are relatively constant. The user can delay the start of this period relative to plant and estimator start up. This avoids any poor estimates which might be induced by start up characteristics. During the establishment of steady values, the outputs of the conditioner are the unaltered RWLS estimates.

Once steady parameter values are established, the algorithm begins to condition and monitor the estimates. A series of "flags" is used to aid this process logic. A parameter is flagged if the algorithm decides it is currently changing. The conditioner outputs the RWLS estimate of a flagged parameter, while unflagged parameter estimates are averaged before output.

Conditioned estimates are monitored for the start and end of a parameter change. This "watch-dog" mode begins by averaging a very brief sequence of the conditioner's output. This filters noisy estimates, especially those of changing parameters which are not averaged by the conditioner. The



a) RWLS Estimation



b) Conditioned RWLS Estimation

Figure 17. Flow Diagrams of RWLS Estimates and Conditioned RWLS Estimates.

user may set the number of estimates in this sequence, though ten estimates should be sufficient. This produces a time-local conditioned estimate. Each time-local estimate is then checked to be within a tolerance of both the previous time-local estimate and the parameter's previous steady value. These tolerances are set by the user for each parameter, again based on his knowledge of the plant. Thus, if a user anticipates change in one parameter, or if he determines one estimate to be more critical than others, he may set the corresponding tolerance tighter than those for other parameters. If the time-local estimate is within these bounds, the algorithm replaces the previous time-local estimates with current time-local values and begins to average the next sequence of conditioned estimates.

If a tolerance is exceeded, the flag indicating change is raised for that parameter. The algorithm then searches for a flagged parameter by order of decreasing tolerance. By this method, the first flag the algorithm encounters will correspond to the most rapidly varying parameter. The subscript of this estimate is noted and all other parameter flags are lowered to assure proper conditioning. At this point, the algorithm begins waiting for the flagged parameter to assume a new steady value. Typically, a parameter change in any real mechanical system will eventually end; for example, a damper will lose all its viscous fluid. A new, steady value is reached when the difference between successive time-local estimates is very nearly zero. From observation, such an end-of-change occurs when successive time-local estimates differ by less than 10^{-5} . When this is sensed, the algorithm lowers the flag on the changing parameter and begins to establish the new, steady parameter values.

A flow chart of the conditioner is shown in Figure 18. In this figure, the term (*repeat*) in a decision block indicates that this decision must be repeated for each parameter estimate. This algorithm requires only five subroutines:

- **WINDOW** a subroutine to "average in" a new data point given the previous average and the number of points included therein;

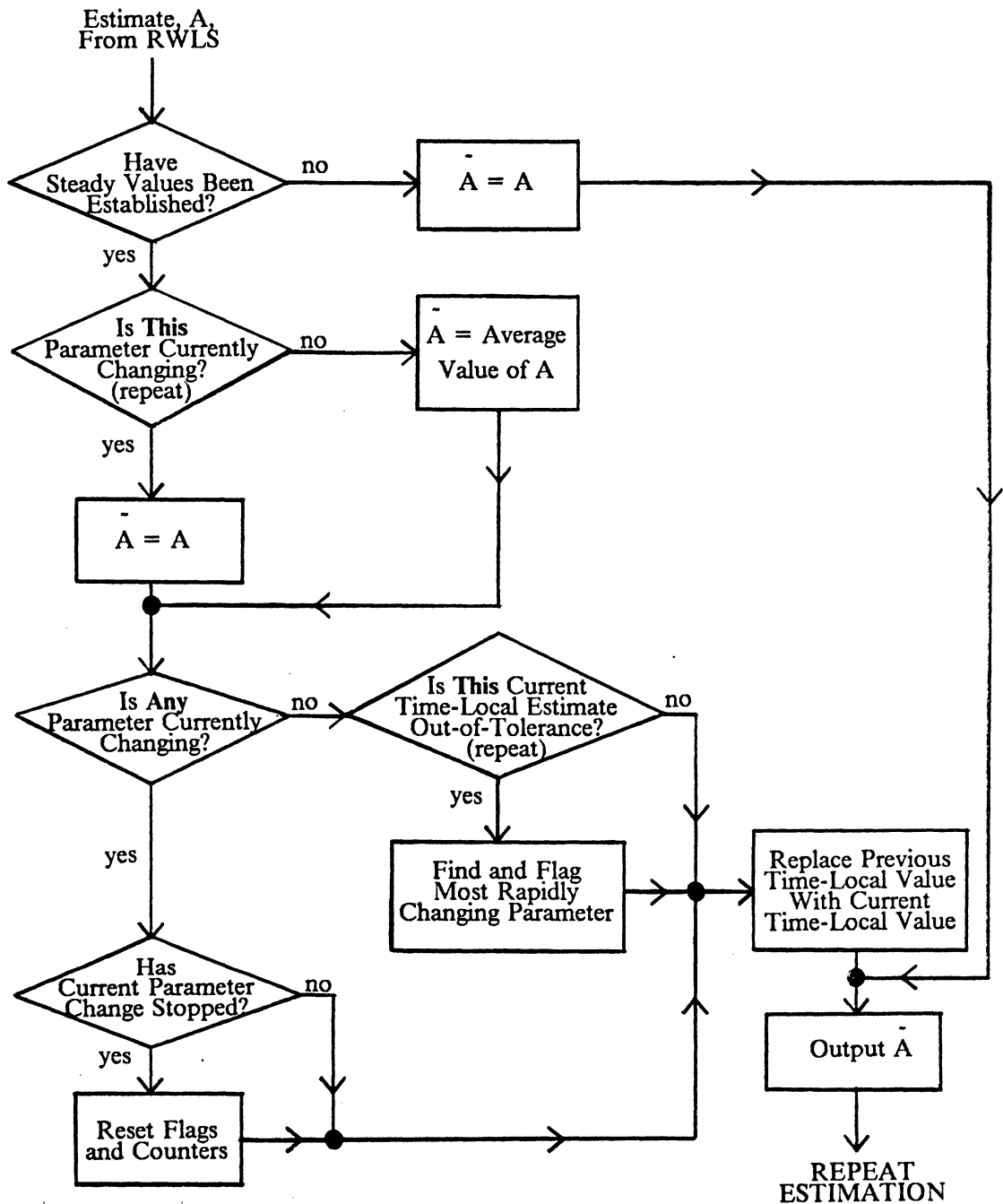


Figure 18. Flow Chart of the Estimate Conditioner.

- **FLAGUP** a subroutine to check that each time-local estimate is within the user specified tolerance of both its steady state parameter value and its previous time-local estimate;
- **DECIDE** a subroutine to search for flagged parameters by order of decreasing tolerance. Requires subroutine **ORDER** to establish the order of this search;
- **FLAGDN** a subroutine to check for the cessation of change of a flagged parameter;
- **ORDER** a subroutine which searches the user-set tolerances and notes them in descending order of magnitude.

The Waterloo FORTRAN/77 source code for these subroutines is contained in Appendix B. The source code for the complete identifier with conditioning is in Appendix C.

4.2 Behavior of Conditioned Estimates

This conditioning algorithm was tested with several plants. This was done to assure that the modification could be applied to plants other than the one from which it was designed. The plants used for this study were the plant studied in the previous chapter, a second order plant with randomly generated parameters and an overdamped second order plant. For each test the accuracies of conditioned and unconditioned estimates were compared.

The tests had several features in common. Time local estimates were obtained by averaging ten conditioned estimates, the plant and estimator were allowed to run 3s before data were first taken to establish initial, steady parameter values and, as before, a ramping parameter change over the period 12s to 48s was used. This was done to reduce the number of variables influencing the conditioning process.

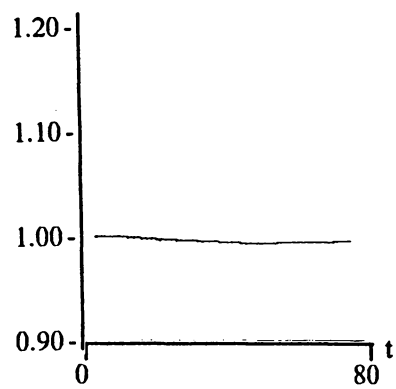
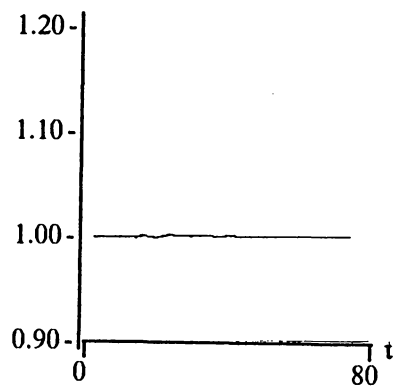
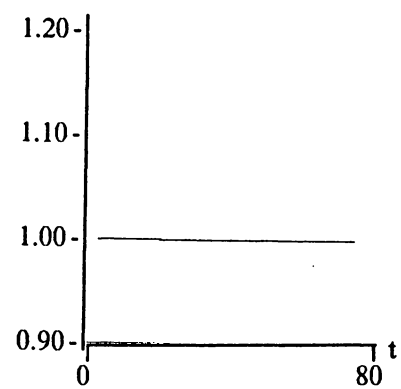
4.2.1 Test Plant of the Previous Chapter

The conditioning algorithm was first tested on the plant used in the previous chapter. Recall that it initially has a spring coefficient of 100 units, a mass of 100 units and a damping coefficient of 60 units. The forcing function applied to this plant was identical to that used in chapter 3: a mean of 20 units with a noise amplitude of 18 units. The spring coefficient was increased by 1%, 10%, 100% and 1000% over the period of change. With this parameter variation in mind, a tolerance of 1.0% was selected for the estimate of $\frac{k}{m}$. Tolerances on the time invariant parameters $\frac{d}{m}$ and $\frac{1}{m}$ were set at 10%.

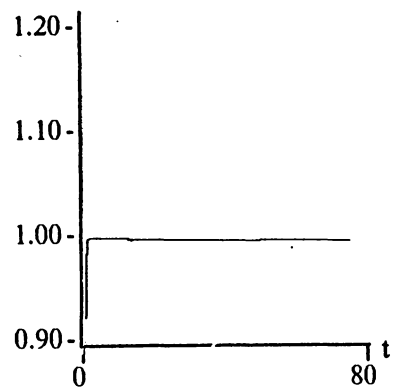
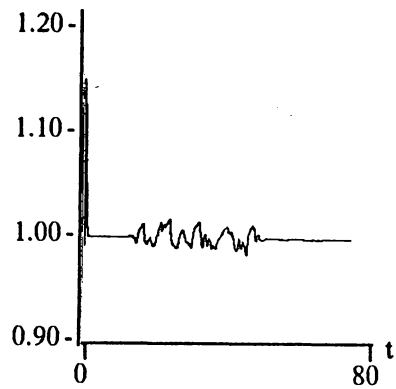
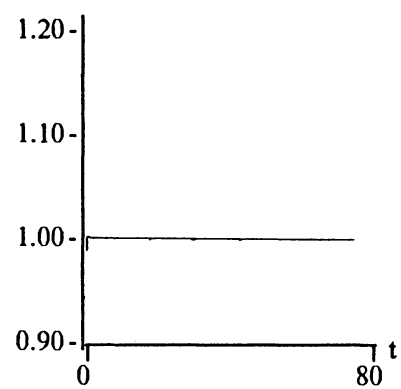
Figure 19 through Figure 22 contain normalized plots of the conditioned and unconditioned parameter estimates for the spring variations. The first of these figures, for the spring coefficient increase of 1%, shows the benefits, and the drawbacks of conditioning. Compare parts b and e of this figure. Conditioning removes much of the fluctuation in the estimate of $\frac{d}{m}$. We also see from this figure that the conditioned estimate of $\frac{k}{m}$ exhibits a slight loss in accuracy not present in the unconditioned estimate. This undesirable performance could be improved by reducing the variation-tolerance applied to the estimate.

Conditioning greatly improved the estimate of $\frac{d}{m}$ for a spring change of 10%. The maximum error for the estimate was reduced from 17%+ to about 2.5%, as shown in Figure 20. Observe also that the fuzz on the estimate of $\frac{1}{m}$ was virtually eliminated by conditioning. However, from this figure we note a phenomenon seen in the 1% spring change above: reduced accuracy in the conditioned estimate of the time varying parameter. The averaging involved in the conditioning process retards tracking of a time varying parameter. Again, this sluggishness could be reduced by decreasing the estimate's variation-tolerance.

Similar results were found in estimates produced during a 100% spring coefficient increase. These estimates are plotted in Figure 21. Conditioning reduced the maximum error in the estimate of

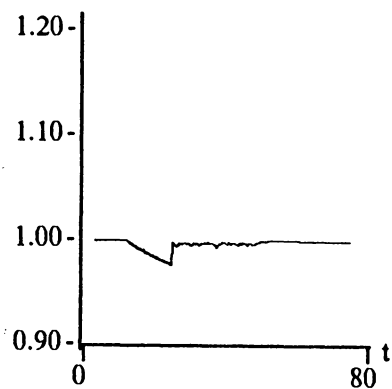
a) k/m b) d/m c) $1/m$

Conditioned

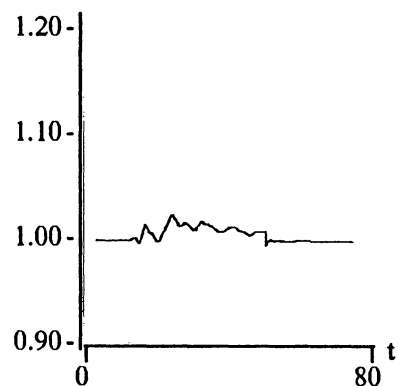
d) k/m e) d/m f) $1/m$

Unconditioned

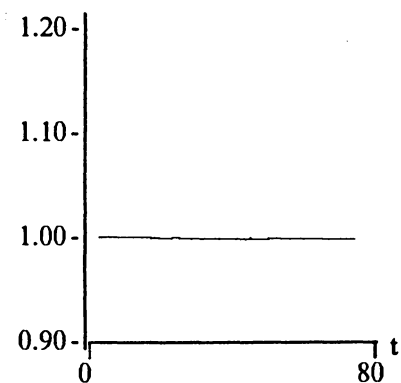
Figure 19. Estimates for a Spring Coefficient Increase of 1%.



a) k/m

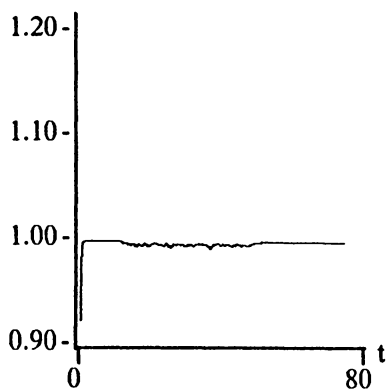


b) d/m

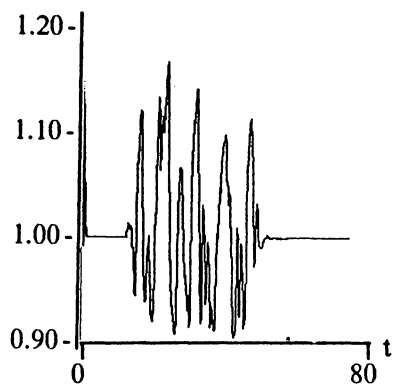


c) $1/m$

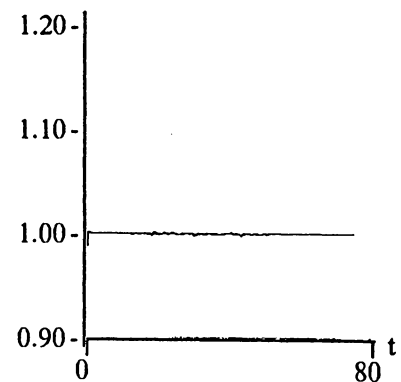
Conditioned



d) k/m



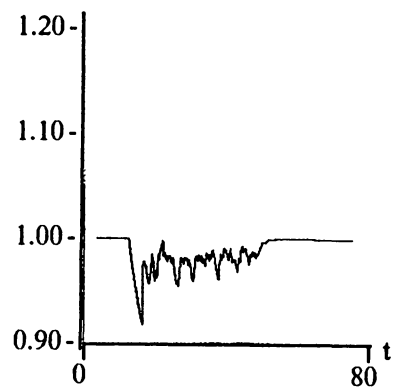
e) d/m



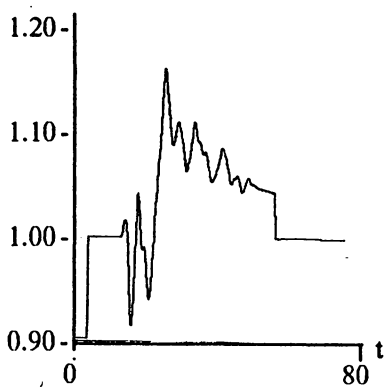
f) $1/m$

Unconditioned

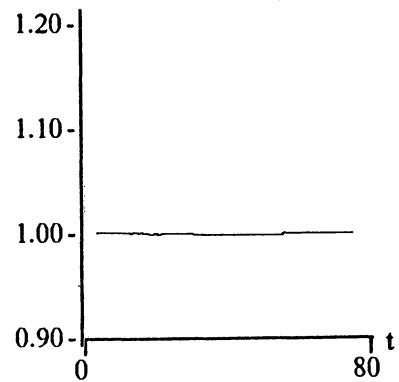
Figure 20. Estimates for a Spring Coefficient Increase of 10%.



a) k/m

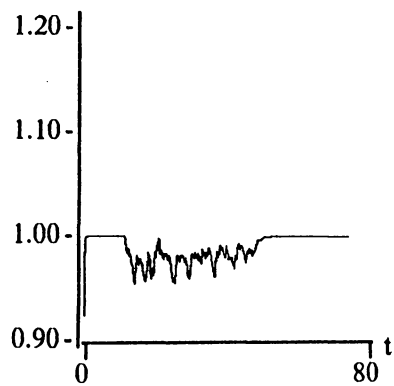


b) d/m

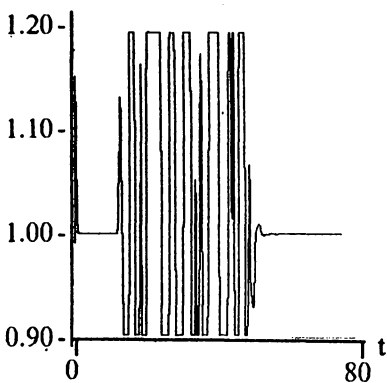


c) $1/m$

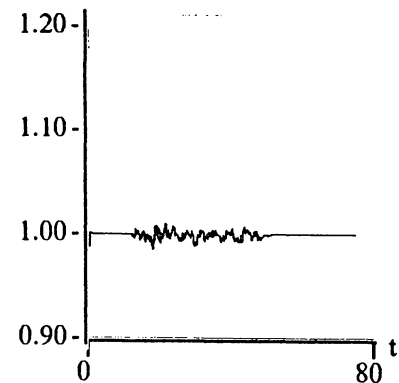
Conditioned



d) k/m



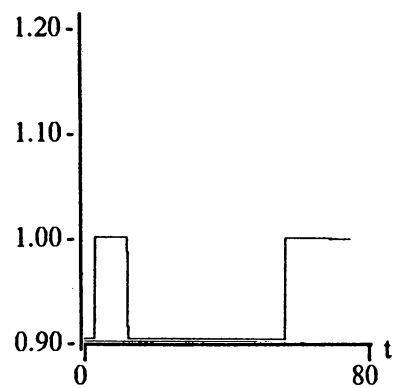
e) d/m



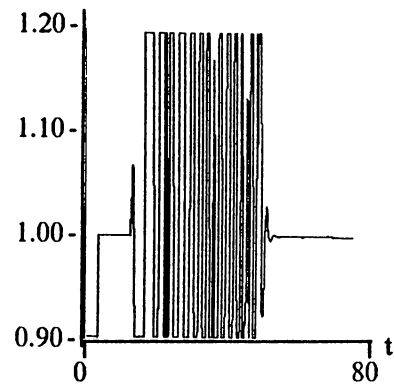
f) $1/m$

Unconditioned

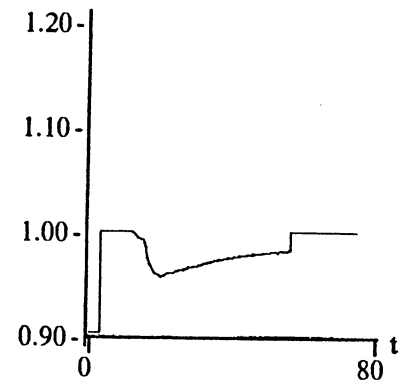
Figure 21. Estimates for a Spring Coefficient Increase of 100%.



a) k/m

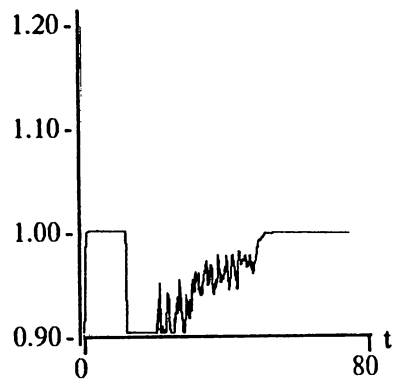


b) d/m

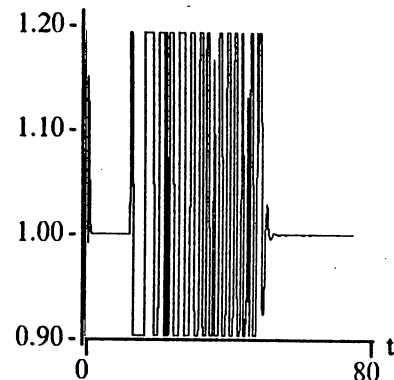


c) $1/m$

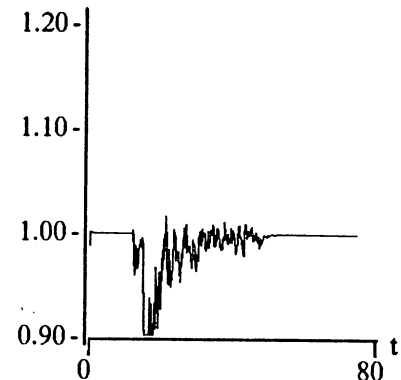
Conditioned



d) k/m



e) d/m



f) $1/m$

Unconditioned

Figure 22. Estimates for a Spring Coefficient Increase of 1000%.

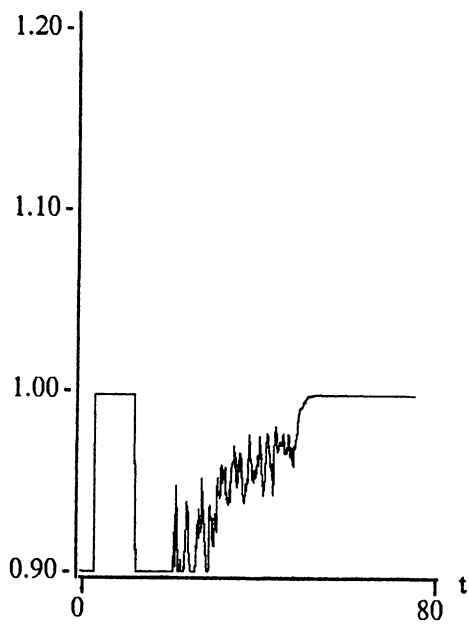
$\frac{d}{m}$ from 25% + to 16.5%; removed fuzz on the estimate of $\frac{1}{m}$; and slightly degraded the accuracy of the estimate of $\frac{k}{m}$.

This pattern did not repeat for the extremely rapid spring coefficient increase of 1000%. Instead, the conditioner was tricked by the RWLS estimates and incorrectly flagged $\frac{d}{m}$ as the most rapidly varying parameter. This caused the conditioner to track $\frac{d}{m}$ and smooth the estimates of $\frac{k}{m}$ and $\frac{1}{m}$. These features can be seen in Figure 22.

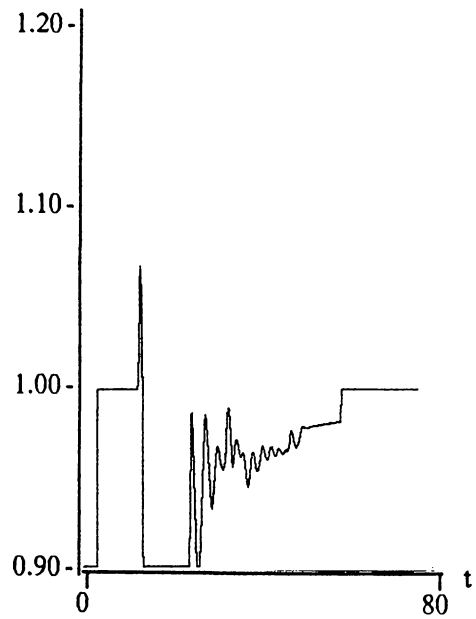
From Figure 22, notice that the estimates of $\frac{k}{m}$ and $\frac{d}{m}$ simultaneously exceed their respective tolerances of 1% and 10%. Both parameters were flagged but the conditioning algorithm chose $\frac{d}{m}$ as the most rapidly varying because it has the larger tolerance. If the user can anticipate that a rapid change in one parameter will cause large fluctuations in estimates of another, less rapidly varying parameter, he can compensate for this behavior by loosening the tolerance on the slowly changing parameter. Figure 23 shows the improved results of such an adjustment. The previous test of a 1000% increase in the spring coefficient was repeated with the variation-tolerance on estimates of $\frac{k}{m}$, $\frac{d}{m}$ and $\frac{1}{m}$ set at 0.5%, 25% and 20% respectively. Caution is urged in making such adjustments. Incorrectly anticipating which parameters change and which remain constant may lead to greatly reduced accuracies in the conditioned estimates.

4.2.2 Random Second Order Plant

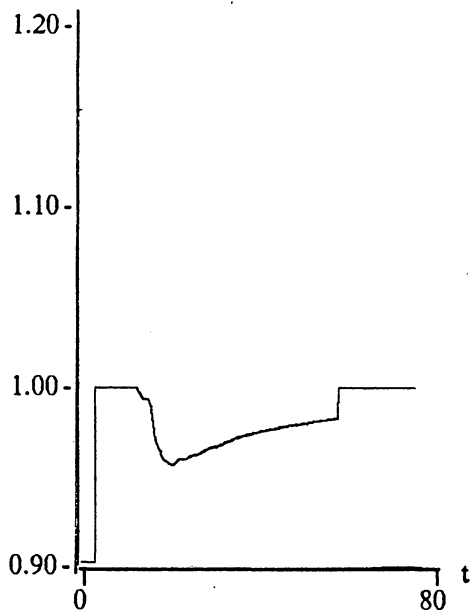
Next, a random number generator was used to create a "double-blind" test of the conditioner. The generator was used to determine parameter values in the range 0 to 500 units for a second order plant. Further random numbers were used to select which parameter varied, the sign of the variation (i. e. increase or decrease) as well as the amount of variation (limited to the range of 0% to 1000%). The average value of the forcing function was selected randomly in the range 0 to 50 units,



a) k/m



b) d/m



c) $1/m$

Figure 23. Conditioned Estimates Produced with Adjusted Tolerances: Spring Coefficient Increased by 1000%.

but additive noise was kept at 90% of this level. The variation-tolerances for $\frac{d}{m}$, $\frac{k}{m}$ and $\frac{1}{m}$ were 10%, 1% and 10%, respectively.

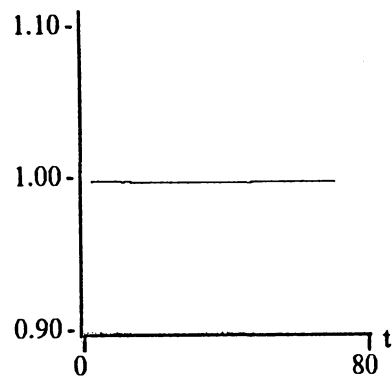
The results of random parameter generation were a spring coefficient of approximately 459 units, a damping coefficient near 409 units and a mass of about 403. This same generator also selected the damper to undergo a 531% increase while the average forcing function was about 42 units. The resulting plant has a natural frequency of $1.067 \frac{\text{rad}}{\text{s}}$ and it settles to within 10% of its final value after 5s when the forcing function is a unit step.

The normalized estimates of this random plant are plotted in Figure 24. As in the previous plant, the accuracy of the conditioned estimate of the changing parameter, $\frac{d}{m}$, has been slightly degraded. Estimates of the truly time-invariant parameters $\frac{k}{m}$ and $\frac{1}{m}$ have been nicely smoothed.

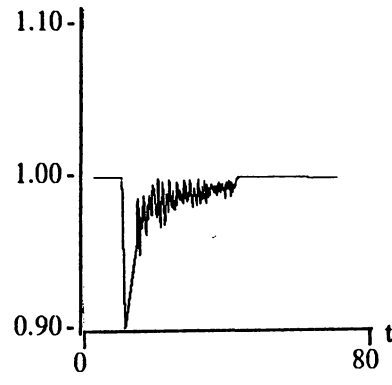
A final test was performed on this plant. The spring coefficient was forced to become the changing parameter; all other plant, conditioner and simulation variables remained the same. The conditioned and unconditioned estimates for this test are plotted in Figure 25. Again, conditioning produced superior results with fewer fluctuations in the non-varying parameter estimates.

4.2.3 Overdamped Second Order Plant

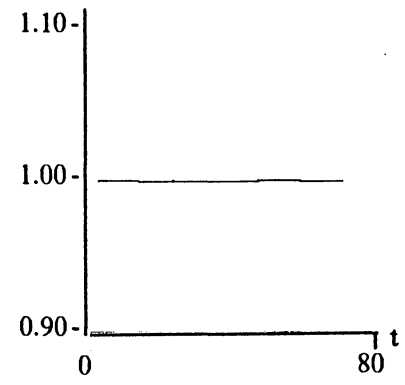
In the final test, an overdamped plant was simulated. Overdamped plants have well separated roots, whereas the previous test plants all had complex conjugate roots and therefore had at least the same real component. A damping ratio of $\zeta = 5$ will separate the roots of the characteristic equation by about two orders of magnitude. Keeping the spring coefficient and mass at their former values of 100 units each, a damping coefficient of 1000 units produces $\zeta = 5$. This places the roots of the characteristic equation at $s = \{-0.1010, -9.899\}$ in the Laplace domain.



a) k/m

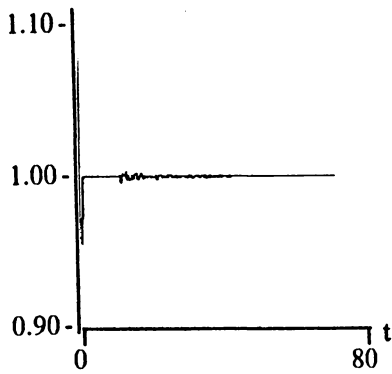


b) d/m

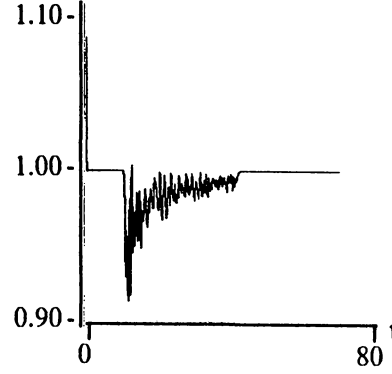


c) $1/m$

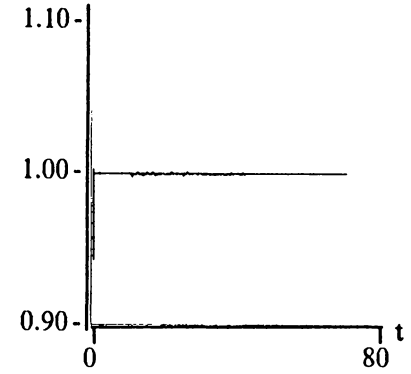
Conditioned



d) k/m



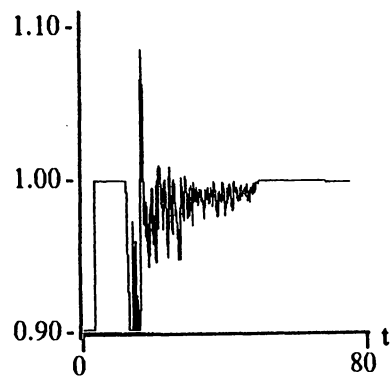
e) d/m



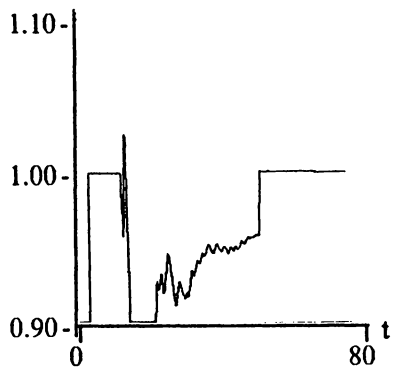
f) $1/m$

Unconditioned

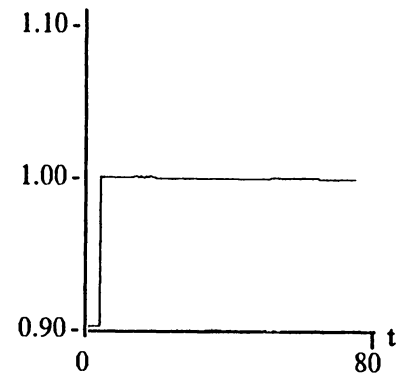
Figure 24. Estimates of Randomly Generated Plant Parameters: Damper Variation.



a) k/m

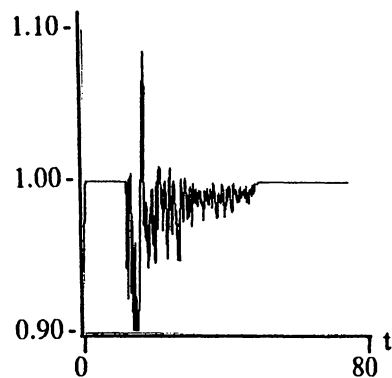


b) d/m

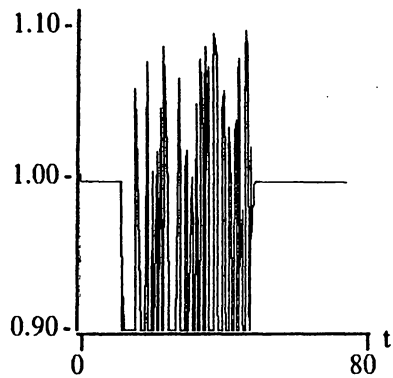


c) $1/m$

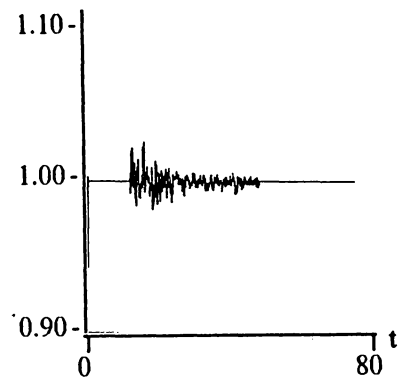
Conditioned



d) k/m



e) d/m



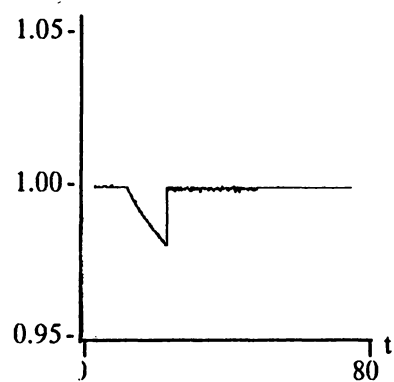
f) $1/m$

Unconditioned

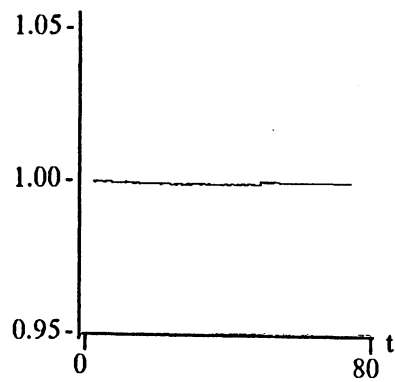
Figure 25. Estimates of Randomly Generated Plant Parameters: Spring Variation.

For this test, the forcing function once again had a mean value of 20 units and an additive noise amplitude of 18 units. The spring coefficient was varied by 10% over the period of change and the variation-tolerances were set at 1%, 10% and 10% for the parameters $\frac{k}{m}$, $\frac{d}{m}$ and $\frac{1}{m}$, respectively.

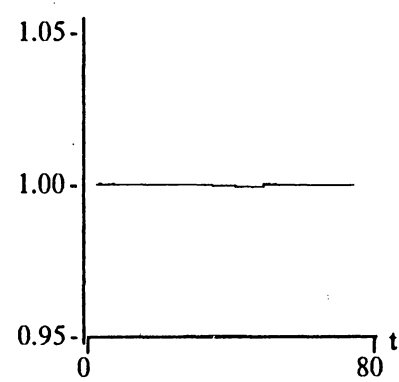
The normalized results of this test are plotted in Figure 26. This figure shows that the estimator had little trouble with this plant. The unconditioned estimates are quite good, with at most a 1% oscillation about their true values during the parameter change. Conditioning nicely removes this slight fuzz from the estimates of $\frac{d}{m}$ and $\frac{1}{m}$, but the accuracy of the conditioned estimate of $\frac{k}{m}$ is a bit worse than its unconditioned value. As noted above, this accuracy could be improved by reducing that parameter's variation-tolerance, causing the conditioner to recognize the change sooner.



a) k/m

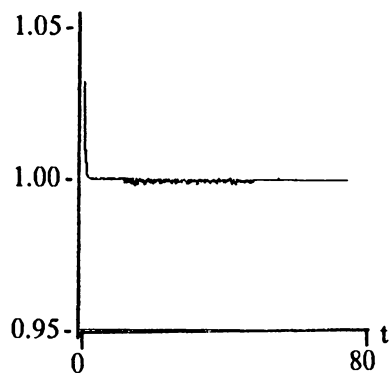


b) d/m

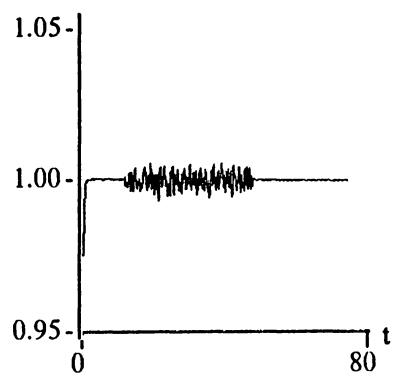


c) $1/m$

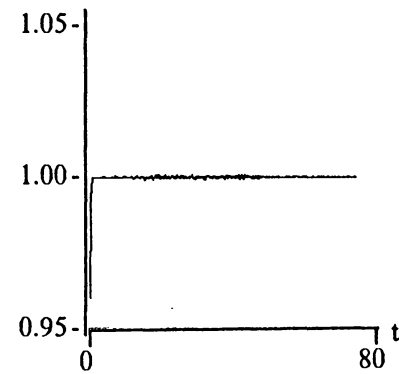
Conditioned



d) k/m



e) d/m



f) $1/m$

Unconditioned

Figure 26. Estimates of an Overdamped Second Order Plant: Spring Variation.

5. Conclusions

Digital methods of parameter identification are fast becoming an alternative to their analog counterparts. This is because of the versatility of digital processors and their rapidly decreasing prices. Modifications or changes to a digital method can be accomplished with re-coded software; the same switch in an analog method could require rewiring and retuning the circuit.

This investigation studied one of the most widely used digital algorithms, the recursive weighted least squares (RWLS) method of parameter estimation. The three main objectives addressed by this research were:

- the development, if possible, of a delineation between *slow* and *fast* parameter changes,
- an examination of the effect of forgetting factor values on the estimates of time varying parameters, and
- the development of a simple modification to the RWLS algorithm to improve its ability to track parameter changes.

The third chapter of this thesis, **Capabilities of Recursive Weighted Least Squares Estimation**, covered a study of the capabilities of the RWLS algorithm, taking in the first two objectives. Based on these findings, the previous chapter outlined a modification (called conditioning) to RWLS identification and compared conditioned and unconditioned estimates.

Though this investigation explored the first objective, it failed to develop an absolute measure of *slowness*. A detailed study of slowness was conducted by varying the the spring coefficient of a

second order plant, increasing it by 0.1% to 1000% over about four settling periods of the plant. These extremes provided rates of change which respectively are assuredly slow and fast. With a commonly used forgetting factor of $\phi = 0.95$, the estimates produced by the RWLS algorithm were normalized by their true values and their accuracies compared. The estimates lost accuracy gradually but, even in this simple plant, all accuracies were not degraded simultaneously. There was no clear division between *slow* and *fast* parameter change. Because of this, a definition of slow parameter change seems to depend on the plant in question and the desired accuracy of the parameter estimates.

An examination of forgetting factor effects provided more positive results. Two of the references cited in this thesis, Eykhoff [4] and Ljung [8], claim that the RWLS algorithm is capable of tracking only slowly varying parameters. Two other references, Andersson [1] and Brogan [2], suggest that forgetting factors are commonly in the range 0.95 to 0.99. No reason was found in the literature to limit the value of ϕ to this range, so several forgetting factors in the range 1.0 to 0.5 were used to produce estimates from identical plant simulations. The plant from the previous study was used and its spring coefficient was increased by 10% over four plant settling periods. This choice for a variation was made because, in the previous study, a forgetting factor of 0.95 had produced good estimates during this change. However, those estimates did exhibit room for improvement.

Forgetting factors less than the commonly used values produced superior estimates. The accuracy of the algorithm's estimate improved from 2% to 0.15% as the forgetting factor was decreased from 0.99 to 0.80. Forgetting factors of 0.99 and less, however, induced oscillations in the estimates of the time-invariant parameters about their true values. The amplitude of these oscillations was reduced, hence the estimate accuracy was improved, as the forgetting factor decreased to 0.80. Finally, for $\phi \leq 0.95$ all three parameter estimates promptly reached their correct values once the spring coefficient reached its new, steady value. The excellent tracking and reduced oscillations achieved argue for the use of forgetting factors less than the commonly used values. The oscillations induced by such low forgetting factors are undesirable and will require either a compromise value of the forgetting factor or compensatory measures.

The modification to RWLS identification proposed in the previous chapter is a compensatory measure. Estimates produced with a forgetting factor of 0.8 are post-processed to smooth the oscillations. This post processing is termed conditioning and basically consists of averaging the RWLS-produced estimates. When a time varying parameter is sensed, the averaging of that estimate is halted to permit the RWLS algorithm to track the change. The range over which a parameter is allowed to vary before it is determined to be changing is set by the user. Hence, by choosing appropriate variation-tolerances, the user can anticipate which parameters may change and select a desired accuracy in the estimates of those parameters.

In tests with three different, second order plants, the conditioning algorithm behaved quite well. Except for one very rapid parameter change, the conditioner successfully recognized which parameter was changing. The conditioned estimate of a changing parameter repeatedly exhibited a reduced accuracy from its unconditioned value. Both of these problems can be corrected by adjusting the user-set variation-tolerances.

One of the tests of the conditioner was conducted on a plant with randomly selected parameters and variations. This was probably the most demanding test of the conditioner since its designer, the author, could not fully anticipate the behavior of a random plant. The conditioner passed this test, correctly sensing which parameter varied and tracking that variation. As in other tests, the accuracy of the conditioned estimate of the changing parameter was slightly degraded. This phenomenon can be corrected by tightening the variation-tolerance established for the changing parameter.

In brief, this investigation found that the RWLS algorithm can track rapid parameter variations within a reasonable accuracy if a forgetting factor less than 0.95 is used. The results of these studies indicate that $\varphi = 0.8$ is a good selection for such purposes. To smooth the oscillations which may be induced by this low forgetting factor, this investigation proposed an estimate post-processor termed a conditioner. Using three different plants including one whose parameter values and variations were selected at random, the conditioning process was tested successfully.

5.1 Recommendations for Future Studies

In any investigation points are bound to arise which merit study but were not addressed due to limitations such as time, facilities or focus. Of these, one of the first areas for development of the conditioning process is the optimization of the identification code. The codes contained in the appendices were written to be easily followed, not quickly executed. In the past few weeks a FORTRAN compiler with a callable routine to return current time-of-day has been added to the computer facilities here at VPI&SU. This compiler could be used to compare execution speeds of an optimized conditioning code and other, similarly optimized methods such as De Keyser's [3] or Andersson's [1]. Such comparisons would be especially interesting if the order of the plant models was much greater than two.

In the more general topic of digital identification, few studies consider simultaneous, rapid parameter variations. Such occurrences may be able to quickly destabilize current techniques and could require the development of a new approach. This investigation leaves the topic of simultaneous parameter changes open for future study. Too, the conditioning process could be extended to handle identification of multiple-input, multiple-output plants.

One of the more interesting ideas for future study is the problem of reduced order identification. Any plant is truly of infinite order -- conventional automatic control design techniques are based on low order approximations of real plants. The question in reduced order identification is, *Is it better to use a model whose order is greater or lesser than that of the plant?* During the course of this investigation some preliminary studies of this matter were conducted. In these tests, a first order system was added between the forcing function and a second order plant, as shown in Figure 27. The damping coefficient, mass and spring coefficient of the second order plant were initially held constant at 60 units, 100 units and 100 units, respectively. The forcing function was constant at 20 units. The value of the time constant of the first order plant, τ , was varied to move the lo-

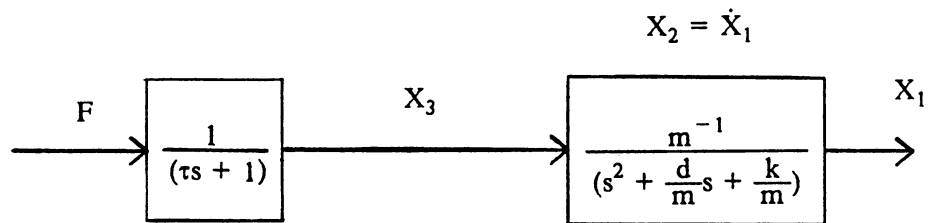


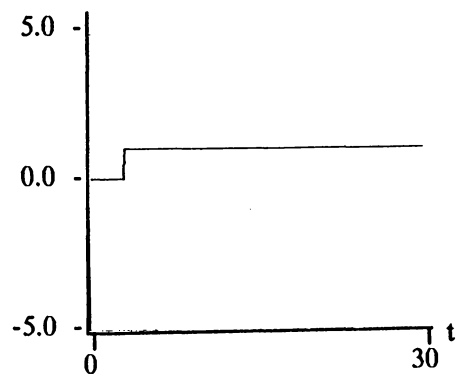
Figure 27. Plant for Reduced Order Identification.

cation of the root due to the first order system. Identification was accomplished by conditioned RWLS using the states (shown in Figure 27) X_1 and X_2 , the forcing function, F , and $\dot{X}_2$. This scheme ignored the state marked X_3 in Figure 27 and forced a second order identification of the third order plant.

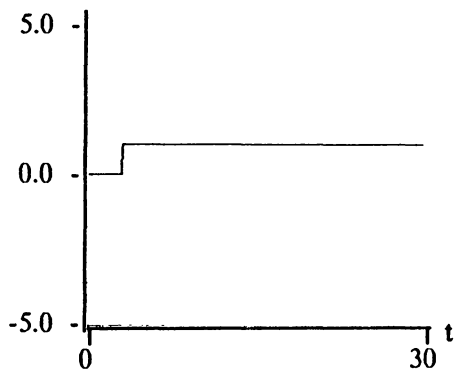
Starting with a value of 0.02 to place the root far away at $s = -50$ in the Laplace domain, the value of τ was increased. This moved the first order root closer to the roots of the second order system which were at $s = \{-0.3 \pm 1.98j\}$. The conditioned and unconditioned estimates produced for $\tau = 0.02$, $\tau = 0.1$, $\tau = 0.5$ and $\tau = 2.0$ are respectively plotted in Figure 28 through Figure 31, normalized by their true values. From these figures note that the second order model produces excellent estimates even when $\tau = 0.5$ places the final root as close as $s = -2.0$. Conditioning smooths some of the spikes and fuzz from these estimates. Figure 31 shows that the second order approximation produces very poor estimates as the three roots of the plant get close to one another. Even worse estimates were produced when the first order root was between the second order roots and the origin.

With such results, the next (and final) step taken in this side-study was to vary a parameter. The spring coefficient in the second order plant was varied while the first order root was at $s = -5$, or $\tau = 0.2$. The conditioned and unconditioned estimates of the second order model are plotted in Figure 32 on page 70, normalized by their true values. This figure shows that the reduced order identification was unsuccessful during a parameter variation. However, the success of reduced order identification for a plant with constant parameters suggests that this topic merits further study.

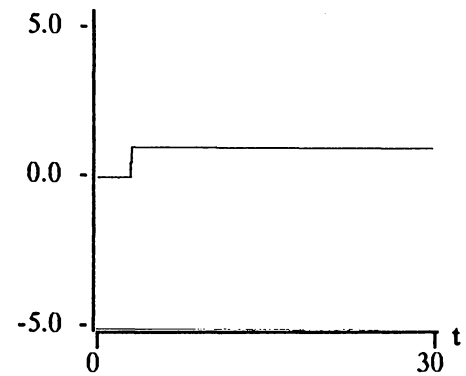
Perhaps the area with the greatest potential for successful future work is the construction and testing of a real-time digital identifier. This identifier could be coupled to an adaptive controller or used solely for plant diagnosis such as in monitoring tool bit wear. From his search of the literature, the author cannot recall any report of the physical testing of real-time digital identification. The availability of microcomputers able to handle high level languages like FORTRAN, plus the fact that many of these machines have addressable ports, should make the initial construction of test



a) k/m

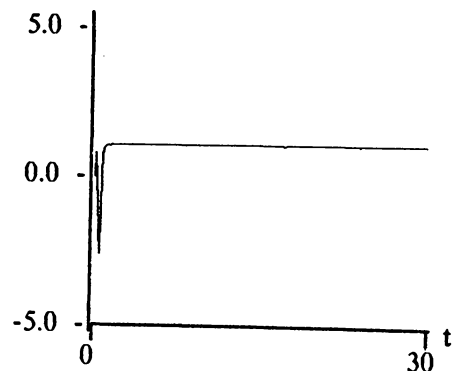


b) d/m

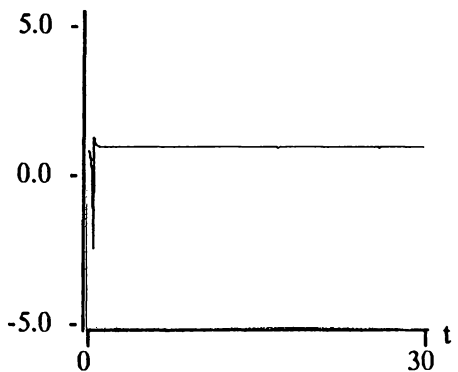


c) $1/m$

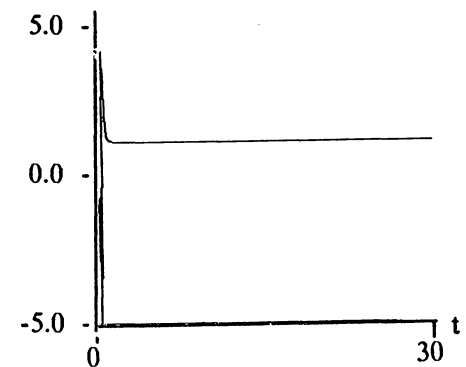
Conditioned



d) k/m



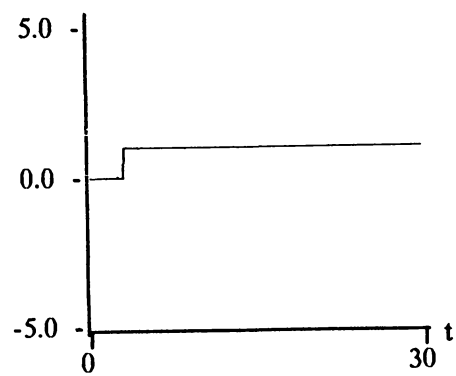
e) d/m



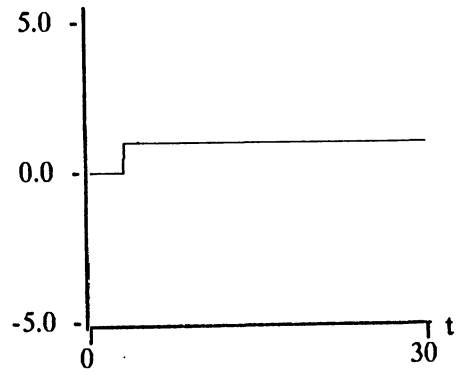
f) $1/m$

Unconditioned

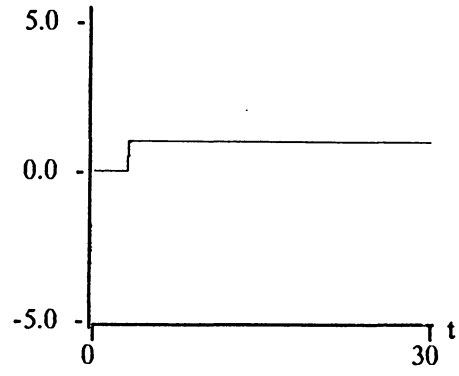
Figure 28. Reduced Order Estimates for Added Root at $s = -50$.



a) k/m

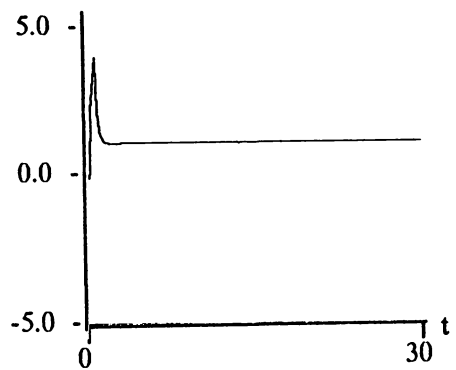


b) d/m

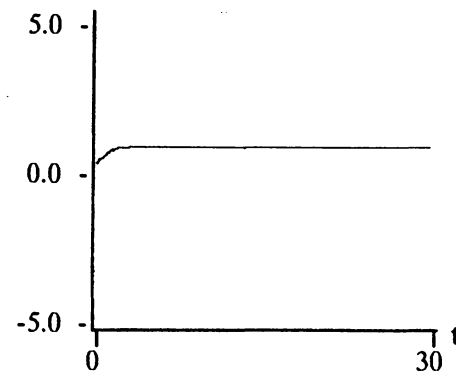


c) $1/m$

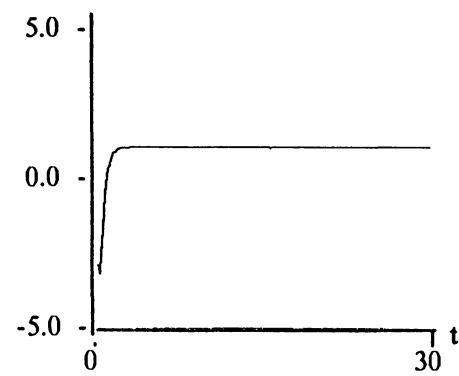
Conditioned



d) k/m



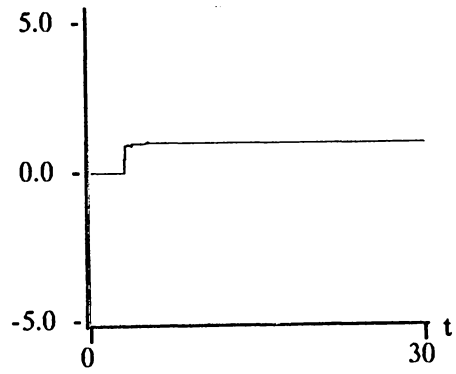
e) d/m



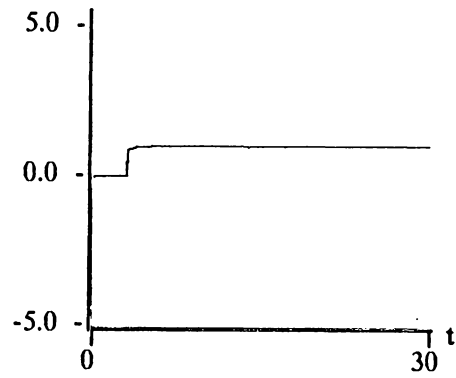
f) $1/m$

Unconditioned

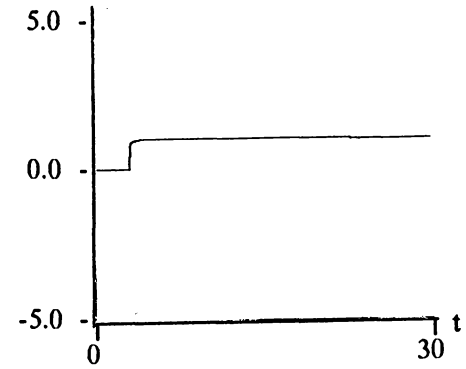
Figure 29. Reduced Order Estimates for Added Root at $s = -10$.



a) k/m

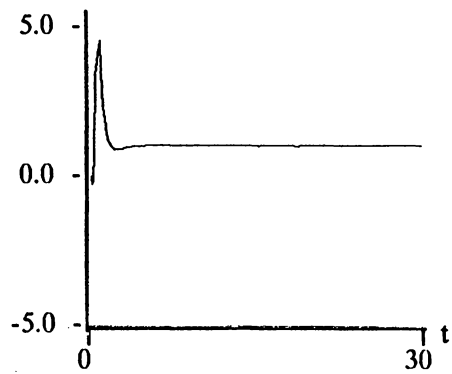


b) d/m

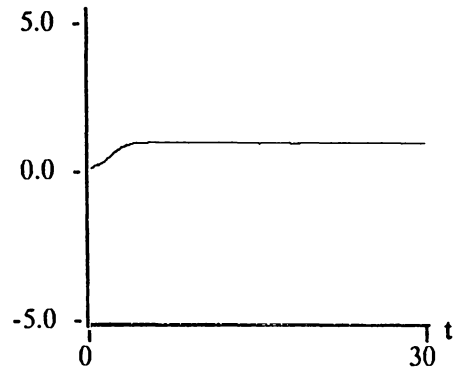


c) $1/m$

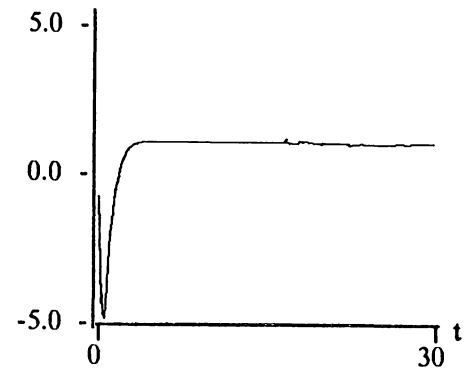
Conditioned



d) k/m



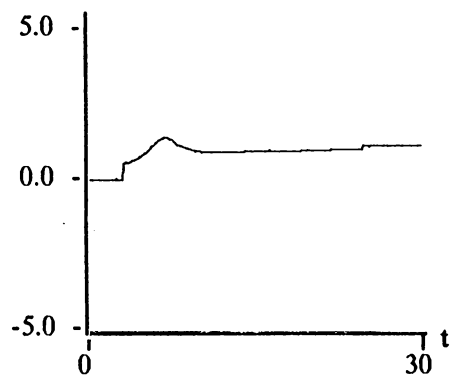
e) d/m



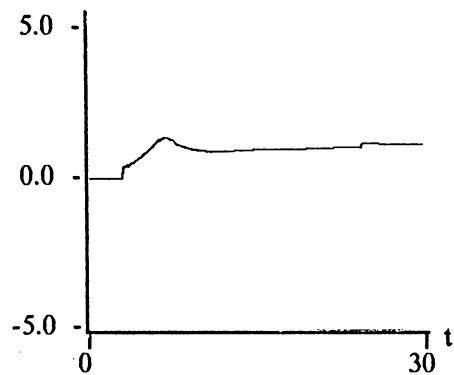
f) $1/m$

Unconditioned

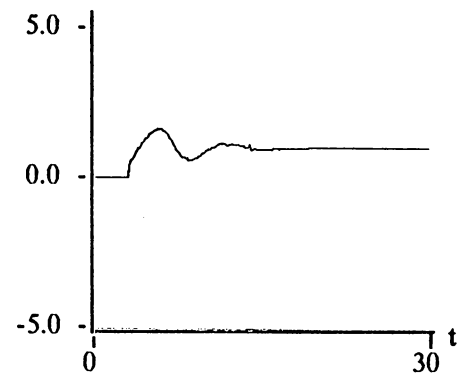
Figure 30. Reduced Order Estimates for Added Root at $s = -2$.



a) k/m

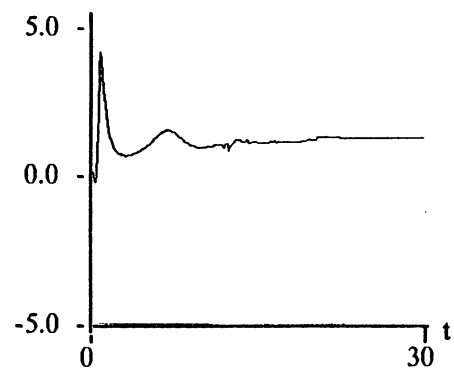


b) d/m

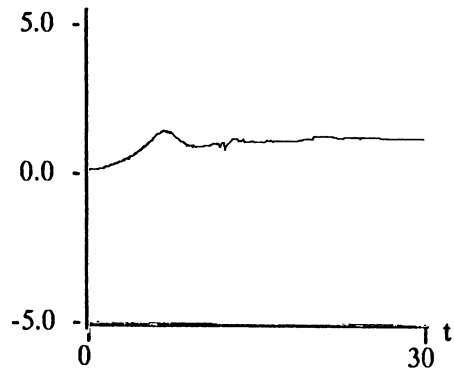


c) $1/m$

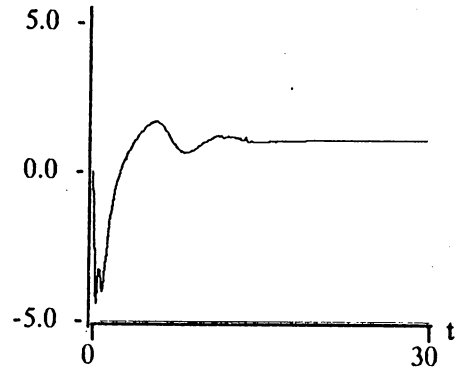
Conditioned



d) k/m



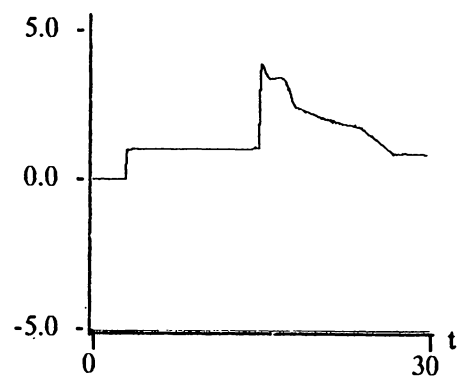
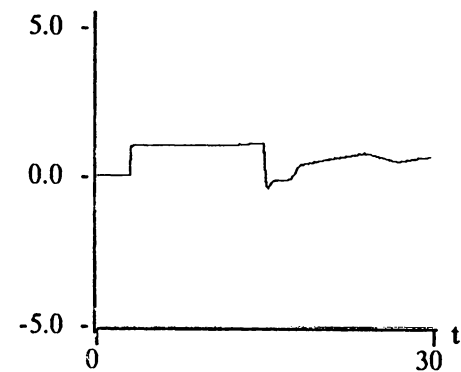
e) d/m



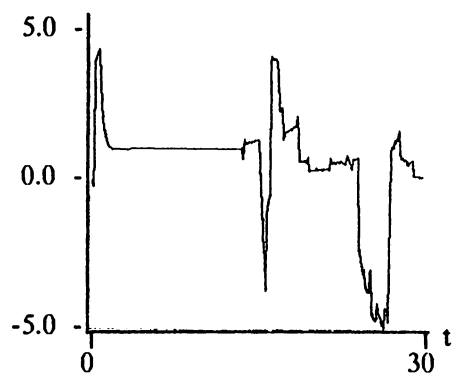
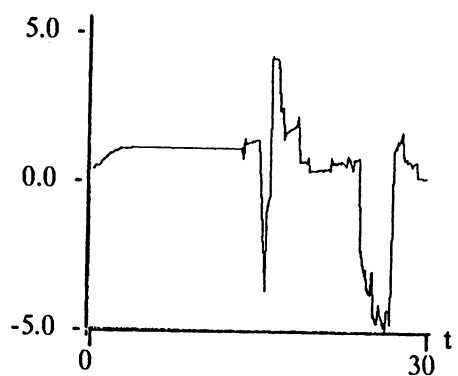
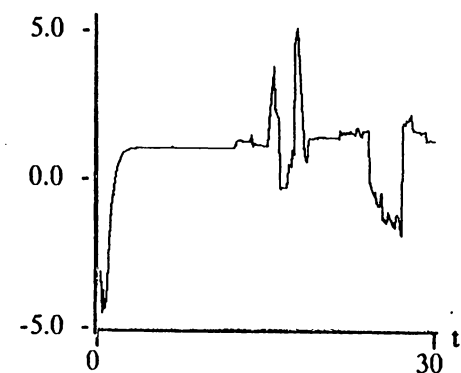
f) $1/m$

Unconditioned

Figure 31. Reduced Order Estimates for Added Root at $s = -0.5$.

a) k/m b) d/m c) l/m

Conditioned

d) k/m e) d/m f) l/m

Unconditioned

Figure 32. Reduced Order Estimates with Parameter Variation.

apparatus relatively easy. Judging from the scarcity of papers on the subject of digital identification of time varying parameters, such equipment could confirm the practicality of extant algorithms as well as serve as a test-bed for future developments.

References

1. Andersson, Peter., *Adaptive Forgetting in Recursive Identification through Multiple Models*, **International Journal of Control**, Vol. 42, pp. 1175-1193, Nov. 1985.
2. Brogan, William L., **Modern Control Theory**, Prentice-Hall Inc., 1985.
3. DeKeyser, R. M. C., *Simple Adaptive Controller for Systems with Fast Parameter Variations*, **Control and Computers**, Vol. 13, No. 2, pp. 72-77, 1985.
4. Eykhoff, Pieter, **System Identification: Parameter and State Estimation**, John Wiley & Sons, 1974.
5. Graupe, Daniel, **Time Series Analysis, Identification and Adaptive Filtering**, Robert E. Krieger Publishing Co., Inc., 1984.
6. James, M. L., G. M. Smith and J. C. Wolford, **Applied Numerical Methods for Digital Computation with FORTRAN and CSMP**, Second Edition, Harper & Row, Publishers, Inc., 1977.
7. Liu, P. T., and P. L. Bongiovanni, *Optimum Inputs for State Identification*, **Journal of Optimization Theory and Applications**, Vol. 45, No. 2, pp. 257-264, Feb. 1985.
8. Ljung, Lennart, and Torsten Soderstrom, **Theory and Practice of Recursive Identification**, The MIT Press, 1983.
9. Mounfield, William Pratt, Jr., *Adaptive Positioning Control by Signal Synthesis*, Thesis, Virginia Polytechnic Institute and State University, 1980.
10. Raisinghani, S. C., and A. K. Adak, *Aircraft Parameter Identification by Gauss-Newton Minimization Technique using Flight Test Data*, **International Journal of Systems Science**, Vol. 14, No. 12, pp. 1395-1409, Dec. 1983.

Appendix A.

RWLS Algorithm

XXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXX

AAAAAAAAAAAAAAAAAAAAAAAA

[illegible]

n n n n n n n n n n

nnnnnnnnnn

CCCCC

٧

[illegible]

CCCCCCCCCCCCCCCC

n n n n n n n n n n n n n

```

C      RAMP1 value of first ramp (T-TRAMP1)
C      RAMP2 value of second ramp (T-TRAMP2)
C      RAN random number generated from NRAN by SUBROUTINE RANDOM
C      RRIS1 amount of ramping step change in parameter
C      SPRING current value of spring coefficient
C      T time
C      TPULZ time pulse begins
C      TPULD duration time of pulse
C      TPULC cycle time of pulse (time between beginning of pulses)
C      TRAMP1 time first ramp begins (start time of ramping step)
C      TRAMP2 time second ramp begins (end time of ramping step)
C      TSTOP end time of simulation
C      X position variable
C      XD velocity variable
C      XDD acceleration variable
C      Y the state vector
C
C      estimation variables
C
C      A the vector of estimated coefficients
C      EPS seed value for the initial P-matrix
C      KESTIM count of the number of estimations performed in time T
C      P "gain" matrix used in the construction of estimates
C      PHI forgetting factor used in recursive weighted
C      least squares
C      PREDUC constructed to update the P matrix
C      PU equals P*U
C      TEOLD the last time estimates were updated
C      TESTIM the time lapse between estimate updates
C      U the estimation input vector
C      UTA equals (U)transpose*A
C      UTP equals (U)transpose*P
C
C      output variables
C
C      KPLOT count of the number of lines of data output for plotting
C      KPRINT count of the number of lines of data output for printing
C      TPLOT time interval between data output for plotting
C      TPLTLD most recent time plotting data was output
C      TPOLD time interval between data output for printing
C      TPRINT most recent time printing data was output
C
C      SUBROUTINE CONTRL (N,NEQ,U,A,P,PREDUC,PU,UTP,C,D,Y,DY,Q,
1      STDST,OLDAVG,AVGTIL,MARK,RANGE,IRANK,AVGA,ATILDE)
C      IMPLICIT REAL (A-H,O-Z)
C      DIMENSION U(N),A(N),P(N,N),PREDUC(N,N),PU(N),UTP(N),C(NEQ),
1      D(NEQ),Y(NEQ),DY(NEQ),Q(NEQ),STDST(N),OLDAVG(N),AVGTIL(N),
2      MARK(N),RANGE(N),IRANK(N),AVGA(N),ATILDE(N)
C
C      initialize control variables and print headings
C
C      PI=2.0*ASIN(1.0)
C
C      simulation
C
C      T=0.0
C      DT=0.01000
C      TSTOP=75.00
C
C      AVSPR=100.

```

```

    AVDAMP=60.
    AVMASS=100.
    AVFOR=20.
    AMP=0.0
    TPULZ=0.000
    TPULD=7.0000
    TPULC=14.000
    AMN=0.9*AVFOR
    NRAN=127317910
    RRIS1=0.10
    TRAMP1=12.0
    TRAMP2=48.0
    RAMP1=0.0
    RAMP2=0.0
C
C    estimation
C
    PHI=0.50
    EPS=5.E4
    TEOLD=0.000
    TESTIM=0.0500
    KESTIM=TEOLD/TESTIM
C
C    printing
C
    KPRINT=0
    TPOLD=0.000
    TPRINT=0.5000
C
C    plotting
C
    OPEN (3,FILE='IDENT SAS')
    TPLOT=0.10
    TPLTLD=0.0000
    KPLOT=0
C
C    Initialize matrix variables
C
    DO 30 I=1,N
      DO 20 J=1,N
        P(I,J)=0.0
        PREDOC(I,J)=0.0
20    CONTINUE
      A(I)=0.0
      PU(I)=0.0
      U(I)=0.0
      UTP(I)=0.0
      P(I,I)=EPS
30    CONTINUE
    DO 40 I=1,NEQ
      C(I)=0.0
      D(I)=0.0
      Y(I)=0.0
      Q(I)=0.0
40    CONTINUE
*****
*
*           Begin Simulation Loop
*
*****
C
C    recalculate system variables and repeat identification
C

```

```

50  CALL RAMP (T,TRAMP1,RAMP1)
    CALL RAMP (T,TRAMP2,RAMP2)
    FACTOR=RRIS1*(RAMP1-RAMP2)/(TRAMP2-TRAMP1)
54  CALL PULSE (T,TPULZ,TPULD,TPULC,IPUL)
    CALL RANDOM (NRAN,RAN)
    SPRING=AVSPR*(1+FACTOR)
    MASS=AVMASS
    DAMP=AVDAMP
    FORCE=AVFOR+AMN*(1.0-(RAN*2.0))+AMP*IPUL
    C(1)=SPRING/MASS
    C(2)=DAMP/MASS
    D(1)=0.0
    D(2)=FORCE/MASS

C
C  simulate plant variables using a Runge-Kutta integration
C
    CALL RKG (NEQ,DT,T,C,D,Y,DY,Q)
=====
C=
C=  Begin identification algorithm
C=
C=
=====
    IF((T-TEOLD).GE.TESTIM) THEN
        KESTIM=KESTIM+1
        TEOLD=TESTIM*KESTIM

C
C  load estimation input (i.e. U) vector
C
        X=Y(1)
        XD=Y(2)
        XDD=DY(2)

C
        U(1)=XD
        U(2)=X
        U(3)=FORCE

C
C
C          Perform Estimation
C
        CALL ESTIM (N,XDD,PHI,U,A,P,PREDUC,PU,UTP)
        DO 17 I=1,N
            ATILDE(I)=A(I)
17    CONTINUE
        END IF
=====
C=
C=  End identification algorithm
C=
C=
=====
C
C  output plotting data
C
    IF((T-TPLTLD).GE.TPLOT) THEN
        KPLOT=KPLOT+1
        TPLTLD=TPLOT*KPLOT
        DAMN=ATILDE(1)/(-C(2))
        SPRN=ATILDE(2)/(-C(1))
        RMAS=ATILDE(3)*MASS
        WRITE(3,2000) T,SPRN,RMAS,DAMN
    END IF

C

```

```

C      check for end of simulation
C
      IF(T.LE.TSTOP) GOTO 50
*****
*
*                      End Simulation Loop
*
*****
2000 FORMAT(1X,F7.3,6(1X,E11.5))
      RETURN
      END

SUBROUTINE ESTIM (N,DERIV,PHI,U,A,P,PREDUC,PU,UTP)
C
C      subroutine ESTIM
C      estimates the paramater vector and updates the P matrix
C      by a recursive weighted least squares method
C
      IMPLICIT REAL (A-H,O-Z)
      DIMENSION U(N),A(N),P(N,N),PREDUC(N,N),PU(N),UTP(N)
C
C      form gain "matrix"
C
      CALL ROWMAT (N,N,U,P,UTP)
      CALL ROWCOL (N,UTP,U,UNDER)
      GAIN=1.DO/(PHI+UNDER)
C
C      update parameter estimate
C
      CALL ROWCOL (N,U,A,UTA)
      CALL MATCOL (N,N,P,U,PU)
      DIFF=DERIV-UTA
      DO 10 I=1,N
        A(I)=A(I)+GAIN*PU(I)*DIFF
10      CONTINUE
C
C      update P matrix
C
      CALL COLROW (N,N,PU,UTP,PREDUC)
      DO 20 I=1,N
        DO 20 J=1,N
          P(I,J)=(P(I,J)-GAIN*PREDUC(I,J))/PHI
20      CONTINUE
      RETURN
      END

SUBROUTINE RKG (NEQ,H,X,C,D,Y,DY,Q)
C
C      subroutine RKG
C      a RUNGE-KUTTA routine for solving differential equations
C      (requires subroutine DERIV)
C
      DIMENSION A(2),C(NEQ),D(NEQ),Y(NEQ),Q(NEQ),DY(NEQ)
      A(1)=0.292893218813452475
      A(2)=1.70710678118654752
      H2=0.5*H
      CALL DERIV (NEQ,C,D,Y,DY,X)
      DO 10 I=1,NEQ

```

```

      B=H2*DY(I)-Q(I)
      Y(I)=Y(I)+B
      Q(I)=Q(I)+3.0*B-H2*DY(I)
10  CONTINUE
      X=X+H2
      DO 20 J=1,2
        CALL DERIV (NEQ,C,D,Y,DY,X)
        DO 20 I=1,NEQ
          B=A(J)*(H*DY(I)-Q(I))
          Y(I)=Y(I)+B
          Q(I)=Q(I)+3.0*B-A(J)*H*DY(I)
20  CONTINUE
      X=X+H2
      CALL DERIV (NEQ,C,D,Y,DY,X)
      DO 30 I=1,NEQ
        B=(H*DY(I)-2.0*Q(I))/6.0
        Y(I)=Y(I)+B
        Q(I)=Q(I)+3.0*B-H2*DY(I)
30  CONTINUE
      RETURN
      END

```

```

SUBROUTINE DERIV (NEQ,C,D,Y,DY,X)
C
C  subroutine DERIV
C  a subroutine containing the differential equations to be
C  solved (differential equations are in phase variable form)
C
      DIMENSION C(NEQ),D(NEQ),Y(NEQ),DY(NEQ)
      DO 10 I=1,NEQ-1
        DY(I)=Y(I+1)
10  CONTINUE
      SUM=0.0
      DO 20 I=1,NEQ
        SUM=SUM-C(I)*Y(I)
20  CONTINUE
C
C  Form the last derivative and include the input
C
      DY(NEQ)=SUM+D(NEQ)
      RETURN
      END

```

```

SUBROUTINE RAMP (T,TON,R)
C
C  subroutine RAMP
C  generates a ramp function (R) starting at time TON
C
      REAL T,R
      IF(T.GE.TON) THEN
        R=T-TON
      ELSE
        R=0.0
      END IF
      RETURN
      END

```

```

C
C
C
C
C
C
C
SUBROUTINE PULSE (T,Z,D,C,I)
subroutine PULSE
generates a pulse function (I):
    beginning at time Z
    with a duration of D
    with a cycle time of C

```

```

REAL T,Z,D,C
IF(T.GE.Z) THEN
    I=1
    IF(T.GE.(Z+D)) THEN
        I=0
    END IF
    IF(T.GE.(Z+C)) THEN
        Z=Z+C
        I=1
    END IF
ELSE
    I=0
END IF
RETURN
END

```

```

C
C
C
C
C
C
C
C
SUBROUTINE RANDOM (NRAN,RAN)
subroutine RANDOM
a pseudo-random number generator
from:  FORTRAN/77
        An Introduction to Structured Problem Solving
        Dyck, V. A., J. D. Lawson and J. A. Smith
        Reston Publishing Company, Inc., 1984
        pp. 466-7

```

```

REAL RAN
INTEGER NRAN
NRAN=NRAN*843314861+453816693
IF(NRAN.LT.0) THEN
    NRAN=NRAN+2147483647+1
END IF
RAN=NRAN*0.4656612E-9
RETURN
END

```

```

C
C
C
C
C
SUBROUTINE ROWMAT (N,M,RIN,ARY,ROUT)
subroutine ROWMAT
calculates:  row vector * matrix = row vector
            rin(n) * ary(n,m) = rout(n)
DIMENSION RIN(N),ARY(N,M),ROUT(M)
REAL SUM
DO 20 I=1,M
    SUM=0
    DO 10 J=1,N

```

```

        SUM=SUM+RIN(J)*ARY(J,I)
10      CONTINUE
        ROUT(I)=SUM
20      CONTINUE
        RETURN
        END

```

```

SUBROUTINE MATCOL (N,M,ARY,CIN,COUT)
C
C  subroutine MATCOL
C  calculates:  matrix * column vector = column vector
C               ary(n,m) * cin(m) = cout(n)
C
        DIMENSION ARY(N,M),CIN(M),COUT(N)
        REAL SUM
        DO 20 I=1,N
            SUM=0
            DO 10 J=1,M
                SUM=SUM+ARY(I,J)*CIN(J)
10          CONTINUE
            COUT(I)=SUM
20        CONTINUE
        RETURN
        END

```

```

SUBROUTINE ROWCOL (N,A,B,ALPHA)
C
C  subroutine ROWCOL
C  calculates:  row vector *dot* column vector = scalar
C               a(n) * b(n) = alpha
C
        DIMENSION A(N),B(N)
        REAL ALPHA
        ALPHA=0
        DO 10 I=1,N
            ALPHA=ALPHA+A(I)*B(I)
10        CONTINUE
        RETURN
        END

```

```

SUBROUTINE COLROW (N,M,C,R,ARY)
C
C  subroutine COLROW
C  calculates:  column vector *cross* row vector = matrix
C               c(n) x r(m) = ary(n,m)
C
        DIMENSION C(N),R(M),ARY(N,M)
        DO 10 I=1,N
            DO 10 J=1,M
                ARY(I,J)=C(I)*R(J)
10        CONTINUE
        RETURN
        END

```

Appendix B.

Conditioning Algorithm Subroutines

```

SUBROUTINE FLAGUP (N,RANGE,STDST,OLDAVG,AVGTIL,MARK)
C
C      subroutine FLAGUP
C      checks for beginning of parameter change
C
REAL RANGE(N),STDST(N),OLDAVG(N),AVGTIL(N)
DIMENSION MARK(N)
DO 10 I=1,N

C
C      previous average comparison
C
      CHNG=ABS(1.0-(OLDAVG(I)/AVGTIL(I)))
      IF(CHNG.GE.RANGE(I)) MARK(I)=1
C
C      steady-state comparison
C
      CHNG=ABS(1.0-(STDST(I)/AVGTIL(I)))
      IF(CHNG.GE.RANGE(I)) MARK(I)=MARK(I)+2
10  CONTINUE
    RETURN
    END

SUBROUTINE DECIDE (N,MARK,IRANK,MOVE)
C
C      subroutine DECIDE
C      establishes which parameter is changing
C
DIMENSION MARK(N),IRANK(N)
I=0
10  I=I+1
    IF(MARK(IRANK(I)).GT.0) MOVE=IRANK(I)
    IF(MOVE.EQ.0) GOTO 10
C
C      set MARKs so less rapidly changing parameters will be averaged
C
DO 20 I=(MOVE+1),N
  MARK(I)=0
20  CONTINUE
    IF(MOVE.GT.1) THEN
      DO 30 I=1,(MOVE-1)
        MARK(I)=0
30  CONTINUE
    END IF
    RETURN
    END

SUBROUTINE FLAGDN (N,MAG,OLDAVG,AVGTIL,MOVE)
C
C      subroutine FLAGDN
C      checks for end of parameter change
C
REAL OLDAVG(N),AVGTIL(N)
MAG=ALOG10(ABS(AVGTIL(MOVE)-OLDAVG(MOVE)+1.0E-9))
IF(MAG.LE.-5) MOVE=0
RETURN
END

```

```

SUBROUTINE ORDER (N,RANGE,IRANK)
C
C      subroutine ORDER
C      establishes an order of the elements of RANGE, entering this
C      order in IRANK.  RANGE is searched for the element with the
C      largest value.  this element-number/subscript is then entered
C      in the first element of IRANK.  the search and assignment are
C      repeated for the next largest value, etc.
C
REAL RANGE(N),BIG
DIMENSION IRANK(N)
C
C      search for largest entry of RANGE
C
      BIG=0
      DO 10 I=1,N
        IF(RANGE(I).GT.BIG) THEN
          BIG=RANGE(I)
          IRANK(1)=I
        END IF
10    CONTINUE
C
C      search for next largest entry of RANGE
C
      DO 30 J=2,N
        BIG=0
        DO 20 I=1,N
          IF(RANGE(I).GT.BIG .AND. RANGE(I).LE.RANGE(IRANK(J-1))) THEN
            MARKER=0
            DO 20 K=1,(J-1)
              IF(I.EQ.IRANK(K)) MARKER=1
10          CONTINUE
              IF(MARKER.EQ.1) GOTO 30
              BIG=RANGE(I)
              IRANK(J)=I
            END IF
30    CONTINUE
      RETURN
      END

SUBROUTINE WINDOW (N,K,SIGNAL,AVRG)
C
C      subroutine WINDOW
C      averages SIGNAL (in vector form).  length of averaging window is
C      set outside this subroutine and is used to control the calls
C      placed to the subroutine
C
REAL SIGNAL(N),AVRG(N)
DO 10 I=1,N
  AVRG(I)=(AVRG(I)*K+SIGNAL(I))/(K+1)
10 CONTINUE
  K=K+1
  RETURN
  END

```

Appendix C.

Conditioning Algorithm

00000000000000000000000000000000

XXXXXXXXXXXXXXXXXXXX

AAAAAAAAAAAAAAAAAAAA

n n n n n n n n n n

n n n n n n n n

৯৯৯৯৯৯৯৯৯৯

ccccc

6

CCCCCCCCCCCCCCCCCCCC

XXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXX


```

LSTDST=5.00/TBERN
LWNDW=0.5/TESTIM
RANGE(1)=0.10
RANGE(2)=0.005
RANGE(3)=0.10
KSMTH=0
KSTDST=0
KWNDW=0
MAG=0
MOVE=0
DO 10 I=1,N
    STDST(I)=0.0
    OLDAVG(I)=0.0
    AVGTIL(I)=0.0
    AVGA(I)=0.0
    ATILDE(I)=0.0
    MARK(I)=0
10  CONTINUE
C
C  printing
C
    KPRINT=0
    TPOLD=0.000
    TPRINT=0.5000
C
C  plotting
C
    OPEN (3,FILE='IDENT SAS')
    TPLOT=0.10
    TPLTLD=0.0000
    KPLOT=0
C
C  Initialize matrix variables
C
    DO 30 I=1,N
        DO 20 J=1,N
            P(I,J)=0.0
            PREDUC(I,J)=0.0
20  CONTINUE
        A(I)=0.0
        PU(I)=0.0
        U(I)=0.0
        UTP(I)=0.0
        P(I,I)=EPS
30  CONTINUE
        DO 40 I=1,NEQ
            C(I)=0.0
            D(I)=0.0
            Y(I)=0.0
            Q(I)=0.0
40  CONTINUE
*****
*
*           Begin Simulation Loop
*
*****
C
C  establish decending order entries of RANGE
C
    CALL ORDER (N,RANGE,IRANK)
C
C  recalculate system variables and repeat identification
C

```

```

50  CALL RAMP (T,TRAMP1,RAMP1)
    CALL RAMP (T,TRAMP2,RAMP2)
    FACTOR=RRIS1*(RAMP1-RAMP2)/(TRAMP2-TRAMP1)
54  CALL PULSE (T,TPULZ,TPULD,TPULC,IPUL)
    CALL RANDOM (NRAN,RAN)
    SPRING=AVSPR*(1+FACTOR)
    MASS=AVMASS
    DAMP=AVDAMP
    FORCE=AVFOR+AMN*(1.0-(RAN*2.0))+AMP*IPUL
    C(1)=SPRING/MASS
    C(2)=DAMP/MASS
    D(1)=0.0
    D(2)=FORCE/MASS

C
C   simulate plant variables using a Runge-Kutta integration
C
CALL RKG (NEQ,DT,T,C,D,Y,DY,Q)
=====
C=
C=   Begin identification algorithm
C=
C=====

    IF((T-TEOLD).GE.TESTIM) THEN
        KESTIM=KESTIM+1
        TEOLD=TESTIM*KESTIM

C
C   load estimation input (i.e. U) vector
C
        X=Y(1)
        XD=Y(2)
        XDD=DY(2)

C
        U(1)=XD
        U(2)=X
        U(3)=FORCE

C
C
C   Perform Estimation
C
        CALL ESTIM (N,XDD,PHI,U,A,P,PREDUC,PU,UTP)
    END IF
Cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
Cc      Perform Maloney's Conditioning
Cc
Cc
Cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
    IF((T-TBOLD).GE.TBERN) THEN
        KBERN=KBERN+1
        TBOLD=TBERN*KBERN

C
C   establish steady state of parameters
C
        IF(KSTDST.LT.LSTDST) THEN
            CALL WINDOW (N,KSTDST,A,STDST)
            DO 60 I=1,N
                ATILDE(I)=A(I)
60        CONTINUE
            END IF
            IF(KSTDST.EQ.LSTDST) THEN
                DO 70 I=1,N
                    OLDAVG(I)=STDST(I)
70        CONTINUE

```

```

        KSTDST=KSTDST+1
    END IF
C
C
C    monitor parameters
        IF(KSTDST.GE.LSTDST) THEN
            DO 80 I=1,N
                CALL WINDOW (N,KSMTH,A,AVGA)
                IF(MARK(I).EQ.0) THEN
                    ATILDE(I)=AVGA(I)
                ELSE
                    ATILDE(I)=A(I)
                END IF
            END IF
80        CONTINUE
C
C    average ATILDE to smooth parameter estimate
        CALL WINDOW (N,KWNDW,ATILDE,AVGTIL)
        IF(KWNDW.GE.LWNDW) THEN
            KWNDW=0
            IF(MOVE.EQ.0) THEN
C
C
C    check for beginning of parameter change
                CALL FLAGUP (N,RANGE,STDST,OLDAVG,AVGTIL,MARK)
                ISUM=MARK(1)
                DO 90 I=1,N
                    ISUM=ISUM+MARK(I)
20        CONTINUE
C
C    decide which parameter is changing (most quickly)
        IF(ISUM.GT.0) CALL DECIDE (N,MARK,IRANK,MOVE)
        ELSE
C
C
C    check for end of parameter change
            CALL FLAGDN (N,MAG,OLDAVG,AVGTIL,MOVE)
            IF(MOVE.EQ.0) THEN
C
C
C    reset counters and flags if parameter change has stopped
                KSMTH=0
                KSTDST=0
                MAG=0
                DO 100 I=1,N
                    MARK(I)=0
30        CONTINUE
100       CONTINUE
            END IF
        END IF
C
C
C    "push-down" current parameter average for next iteration
        DO 110 I=1,N
            OLDAVG(I)=AVGTIL(I)
110       CONTINUE
        END IF
    END IF
END IF
C=====
C=
C=    End identification algorithm
C=

```

```

C
C
C      output plotting data
C      IF((T-TPLTLD).GE.TPLOT) THEN
          KPLOT=KPLOT+1
          TPLTLD=TPLOT*KPLOT
          CDAMN=ATILDE(1)/(-C(2))
          CSPRN=ATILDE(2)/(-C(1))
          CMAS=ATILDE(3)*MASS
          UDAMN=A(1)/(-C(2))
          USPRN=A(2)/(-C(1))
          UMAS=A(3)*MASS
          WRITE(3,2000) T,CSPRN,CMAS,CDAMN,USPRN,UMAS,UDAMN
      END IF
C
C      check for end of simulation
C      IF(T.LE.TSTOP) GOTO 50
*****
*
*                      End Simulation Loop
*
*****
2000 FORMAT(1X,F7.3,6(1X,E11.5))
      RETURN
      END

```

**The vita has been removed from
the scanned document**