A FAULT-ADAPTIVE POLE-PLACEMENT ALGORITHM SUITED FOR COMPUTER IMPLEMENTATION

By

MARK ALAN BREWER

Bachelor of Science in Electrical Engineering
Oklahoma State University
Stillwater, Oklahoma
1983

Master of Science in Electrical Engineering
Oklahoma State University
Stillwater, Oklahoma
1984

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
July, 1988

A FAULT-ADAPTIVE POLE-PLACEMENT ALGORITHM SUITED FOR COMPUTER IMPLEMENTATION

Thesis Approved:

[illegible]
Thesis Advisor

[illegible]

[illegible]

[illegible]

[illegible]
Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to numerous individuals who contributed time, assistance, and encouragement throughout my graduate program. In particular, I would like to thank Dr. Charles Bacon for tremendous support and encouragement throughout my years at Oklahoma State University. His assistance was invaluable. Thanks also goes to Dr. Rao Yarlagadda, Dr. Martin Hagan and Dr. Marvin Keener for their time and contributions. Their many suggestions and ideas were very beneficial.

Let me also acknowledge the generous research fellowship of Dow Chemical. Their fellowship was extremely influential in providing the opportunity and time for this research program. In addition to Dow Chemical, I would like to thank numerous individuals at AT&T Technologies, Inc. in Oklahoma City for their support and flexibility in setting working hours. In particular, let me single out Bill Dickerson, Bob Langmacher, Charlie Stapp, Sam Kysar, Connie Moore, and Bob Hering as supervisors who allowed me the opportunity to complete my studies.

Many thanks are due my parents for making this time at school possible. Their encouragement and assistance was extremely important.

Finally, let me thank my wife Beth who was willing to wait for a new house. She gave up many hours while I worked on homework and research. Her wonderful support held everything together.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION TO THE FAULT-ADAPTIVE CONTROL PROBLEM

Introduction

This research presents a new, fault-adaptive control strategy for linear systems suitable for fast, on-line computer control schemes. The new strategy is an extension of the well-known pole-placement method initially developed by Brogan[1]. Details of the computer implementation are discussed and numerous examples are shown.

This chapter introduces the preliminary definitions, the various failure modes that occur in linear systems, the associated ideas of controllability and some previous work on fault-adaptive schemes.

Chapter II develops the problem of adapting to actuator failures in control systems. Several approaches are explored and discussed. Brogan's method is introduced and explained in detail. The advantages and suitability of Brogan's approach for the fault-adaptive problem are explained. Finally, Brogan's method is cast into a new form for purposes of this new adaptive strategy.

Chapter III develops the adaptive strategy in detail for the problem of finding a feedback solution as quickly as possible. Several possible control approaches are introduced and considered. Their relative advantages and disadvantages are explored, and examples of their use are presented. Finally, a four-step method which gives excellent results at minimal computer cost is chosen. The four-step test is compared against brute force computer solutions through extensive computer simulations. Chapter III closes by comparing the two methods and showing how the four-step method results in significantly superior performance over a brute force computer solution.

Chapter IV extends the results of Chapter III for the problem of choosing a 'best' solution out of a solution space. In Chapter IV, two different approachs are presented. The first approach uses the four-step test developed in Chapter III to reduce the problem solution space by approximately forty-four percent. A linear search through the remaining space is then performed to determine the 'best' solution. The second approach redefines the feedback matrix calculation procedure and then uses a non-linear least squares algorithm to find a solution. It is shown that the least-squares approach can generally find better solutions than the linear search method.

Chapter V covers several details required for a successful computer implementation of the fault-adaptive method outlined in this research. Specific details for implementing the four-step test are covered. Chapter V also explains the use of the four-step test in the overall fault-adaptive solution.

Finally, Chapter VI summarizes the work and discusses possible directions for future work. The last chapter also covers a few details that were not addressed in earlier chapters.

Preliminary Definitions

This research deals with linear time-invariant systems described by systems of differential equations of the form:

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} \tag{1}$$

$$\underline{y} = C\underline{x} \tag{2}$$

where $\underline{x}$ is the state vector, $\underline{y}$ is the output vector, $\underline{u}$ is the input vector, A is the system matrix, B is the input mapping matrix, and C is the output mapping matrix. For most discussions, C = I, where I is an identity matrix.

An important observation from equation (1) is that the columns of B describe how each input signal is mapped into the state trajectory. Each column represents a different actuator that can influence the system.

The word *controller*, or sometimes *actuator*, will be used to refer to individual system inputs $(\underline{u})_i$.

Throughout this research, reference will be made to linear feedback of the form:

$$\underline{u} = F\,\underline{x} + \underline{w} \tag{3}$$

where F is the feedback matrix and $\underline{w}$ is the input vector to the closed-loop system.

Failures in Control Systems

In a system with state or output feedback, the system failures that can occur fall into three distinct types. A first type of failure that can occur is in the plant dynamics. The open-loop model used to describe the system is no longer valid. If a wing fails on an airplane, plant dynamics will dramatically change and the original control laws are no longer valid. A second type of fault occurs when a controller that influences a system fails, but the original open-loop dynamics of the plant are not affected. An example of this type of failure is the loss of a jet engine on an airliner. The plant dynamics governing the plane's motion have not changed, but one controller that influences the system dynamics has failed. A third type of failure occurs when sensors used to determine plant states fail. An example of this type of failure would be a failed altimeter in an airplane.

This research deals with the second type of failure. Specifically, failure modes are considered when one or more controllers in a multiple controller environment fail.

It will be assumed that if a controller fails it can be taken out of service so that it exerts no influence on the system. This assumption corresponds to zeroing out a column of B. Throughout the rest of this research, changes in the B matrix will correspond to failures in controllers. It should be stressed, however, that plant dynamics are *not* changing.

In other cases, consideration will be given to the problem where the performance of the actuators is slowly varying over a long time period. This problem can be handled by slowly varying the parameters of the B matrix and then adapting to the new conditions.

Consider the following example of a system before and after an actuator failure occurs. Given:

$$\dot{\underset{\sim}{x}} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \underset{\sim}{x} + \begin{bmatrix} 1 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \underset{\sim}{u} \qquad \underset{\sim}{y} = \underset{\sim}{x} \tag{4}$$

If the goal is to move the closed-loop pole positions to -2, -3, and -4, the feedback matrix:

$$F = \begin{bmatrix} -0.3660 & -1.1156 & 1.3944 \\ -3.5119 & 1.4874 & -1.8593 \\ -0.0807 & -1.4364 & -4.4545 \end{bmatrix} \tag{5}$$

can be used to move the poles to the desired locations[11]. The resulting closed-loop system matrix is given by:

$$A + BF = \begin{bmatrix} -2.8779 & 0.37184 & -0.4648 \\ 0.18721 & -2.66755 & -1.6655 \\ -0.08069 & -0.4364 & -3.4545 \end{bmatrix} \tag{6}$$

If the characteristic equation is calculated, the resulting pole positions meet the design goal.

$$f(\lambda) = | \lambda I + A + BF | = \lambda^3 + 9\lambda^2 + 26\lambda + 24 \tag{7}$$

$$f(\lambda) = (\lambda + 2)(\lambda + 3)(\lambda + 4) \tag{8}$$

Under normal conditions, the calculated control law F will work exactly as required. The resulting pole positions will be exactly where they were intended to be positioned. However, consider what happens if the second actuator fails and can no longer influence the system. Without any adaptation to this failure, the system will be described by:

$$A + B_{-2}F = \begin{bmatrix} 0.634 & -1.1156 & 1.3944 \\ 0.1873 & -2.6676 & -1.6657 \\ -0.0807 & -0.4363 & -3.4545 \end{bmatrix} \tag{9}$$

where B_{-2} is obtained from B by making all the entries in the second column of B zero. The corresponding characteristic equation is:

$$f_{new}(\lambda) = \lambda^3 + 5.488 \lambda^2 + 4.928 \lambda - 4.0957 \tag{10}$$

when the characteristic equation is factored, the new pole positions are clearly seen to be wrong:

$$f_{new}(\lambda) = (\lambda + 2) (\lambda - 0.512) (\lambda + 4) \tag{11}$$

Unless the system adapts to the loss of the actuator, the system will be unstable.

If the control law calculation is repeated on this 'new' system, the required pole positions can still be achieved by the following feedback matrix F:

$$F = \begin{bmatrix} -39.756 & 15.567 & -19.459 \\ 0 & 0 & 0 \\ 23.648 & -11.4866 & 8.108 \end{bmatrix} \tag{12}$$

Fault-Tolerance Definitions

Two separate types of fault-tolerance can be considered. First, a system could be defined as fault-tolerant if it remains controllable after the failure of any single controller $(\underline{u})_i$. In other words, if a controller should fail, the system will remain controllable in the reduced order (one column in B will have zero entries) configuration. As shown later, this is equivalent to having continued freedom to arbitrarily assign the closed-loop poles to any desired location. The previous example represents this type of fault-tolerance.

A second type of fault-tolerance could require only stability of the system after the failure of any single controller $(\underline{u})_i$. For this type, the close-loop poles can only be placed in the stable region of the complex plane but not placed at specific locations.

Clearly, the first type of fault-tolerance is more difficult to achieve than the second type. Recall that a system can be stable yet not be fully controllable. This research will concentrate on the first fault-tolerant definition where the system remains controllable.

On the next page, Figure I represents a fault-adaptive configuration where an ordinary system containing closed-loop feedback is augmented by additional systems to achieve fault adaption. The additional details include a fault detection/identification system followed by an fault-adaptive system. The work of this research concentrates on the fault-adaptive processor which calculates a new control law in the event of a system fault or change. The new control law replaces the original feedback matrix F.

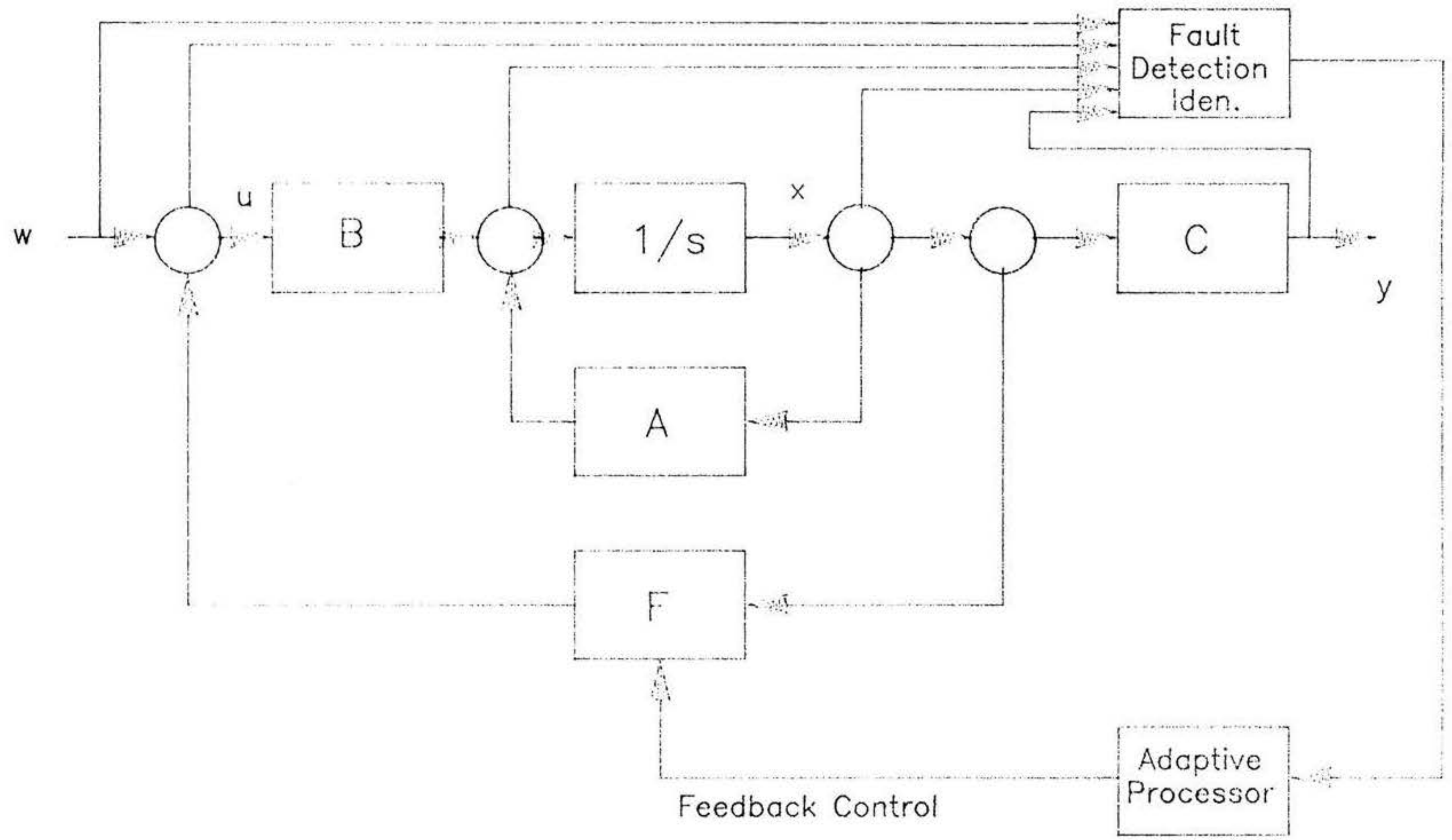

Figure 1. Adaptive Feedback Configuration

Classical Controllability

Recall that a system is controllable if the system state can be moved from any arbitrary state to any other state in a finite amount of time. Gilbert's Criterion offers the best insight into the physics of controllability[2]. His criterion states that if a system of the form (1) is transformed by the linear transformation:

$$\underset{\sim}{x} = M \underset{\sim}{z} \qquad (13)$$

where M is the modal matrix of the A matrix and:

$$A^* = M^{-1}AM \qquad (14)$$

$$B^* = M^{-1}B \qquad (15)$$

Then, if A* is diagonal and if each row of B* has a non-zero element, the system is controllable. For example:

$$A^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \qquad B^* = \begin{bmatrix} 1 \\ 8 \\ 2 \end{bmatrix} \qquad (16)$$

Clearly, each state variable can be controlled because there are inputs to each row of the A matrix.

While Gilbert's Criterion is a simple explanation of controllability, it turns out that it is not the most efficient means of determining the controllability of the system. It is not efficient to evaluate the modal matrix because it requires solving for eigenvalues, eigenvectors, and eventually the inverse of M.

A more efficient means of determining the controllability of a system is Kalman's test for controllability[3] [4]. This test states simply that if a system is described by the form (1), where rank(A) is n and the controllability matrix P is formed as:

$$P = \left[B \,|\, AB \,|\, AAB \,|\, \cdots \,|\, A^{\left(n-1\right)}B \right] \tag{17}$$

Then, the system is controllable if, and only if, the rank of P is n. If the rank is less than n, then there are not enough linearly-independent inputs to influence the future states of the system.

If the dimension of P is small, a quick test to determine if rank(P) is equal to n is given by:

$$\det (P\, P^T) \neq 0 \tag{18}$$

A further point about controllability was stated by Wonham[5]. He proved that the pair (A,B) as in equation (1) is controllable if, and only if, for every choice of the desired eigenvalues, there is a matrix F, in equation (3), such that the closed-loop system defined by an A matrix, a B matrix and a feedback matrix F (A,B,F), has the desired eigenvalues. In other words, controllability is equivalent to the property that the closed-loop transfer matrix:

$$T(s) = C\left[sI - (A + BF)\right]^{-1} B \tag{19}$$

can be assigned an arbitrary set of poles by a suitable choice of the feedback matrix F.

Non-Minimal Control System

At this point, the earlier example problem and the previous discussion on controllability can be tied together to make an important observation about a system that is fault-adaptive. Any system that can adapt to an actuator fault must alternatively be described as being controllable when any single actuator is removed from the system. In the example problem, when one actuator was removed there was still another means to control the system using the remaining actuators. A further point is that a system can be described as fault-adaptive when any single actuator is removed, or any two actuators, or any three, etc. Such a means can be used to describe the degree to which a system is fault-adaptive.

If a system has the characteristics described above and the required pole-placement control specifications do not offer any criteria for selecting between the available feedback alternatives, the control problem will be under-determined because there will be more than one control means to achieve the required pole positions.

Previous Work on Fault-Tolerance

There are several papers in the literature exploring various issues of the fault-adaptive problem. The following section will summarize some relevant works.

Raza and Silverthorn address the fault-tolerant control problem using an L_2 scheme and achieve good results on a flight control system[6]. Their work differs from this research in two regards. First, they vary the B matrix by applying different combinations of controllers as opposed to changing the feedback matrix F. Second, the optimization problem that they address is to minimize change in control signal variation as opposed to minimizing pole movement. They change combinations of controllers as needed. As a result of their approach, they are not able to maintain exact pole locations.

Alos[7] addresses the problem of designing the system to be stable such that if an actuator fault occurs, the system will maintain stability without any changes in feedback. He argues effectively that even if a fault-adaptive scheme is available, the system should remain stable during the time it takes an adaption mechanism to detect and react to the fault. This work does not address the question of how to adapt to a failure after one has occurred.

A third relevant paper is by Pearson and Staats[8]. Their analysis is similar to Alos in that they define classes of plants that will be robust to arbitrary perturbations in problem data or controller parameters. Like the previous paper, this one does not address adaption after the detection of a fault.

Another significant paper is by Ackermann[9]. This paper presents a design method that determines state or output feedback which will result in stability despite variations in plant parameters, sensor failures, and quantization effects in the controller. Ackermann defines a parameter space P that consists of the allowable feedback parameters that will result in a closed-loop system with eigenvalues in a specified region of the eigenvalue plane. He then shows how to calculate P and how to design a system that is robust to the above failures. He states, however, that his technique does not apply well to the multi-input problem due to too many design parameters. Further, he states that the problem of actuator faults is still open.

The problem studied in this research could also be classified under the broad heading of adaptive control. However, most of the adaptive control literature concerns deterministic autoregressive moving average (DARMA) models and not linear multivariable models. Of the works on the multivariable problem, Elliot and Wolovich have done extensive study[10] [11]. Their full multivariable adaptive approach provides for variations in the plant dynamics as well as controller response and, as such, they are addressing a different problem than the one presented in this research. Their approach also has a restricted control structure and is better geared towards small variations in system parameters as opposed to catastrophic events such as controller failures.

A final reference that addresses the relevance of this research is a report found in the April 1987 issue of IEEE Transactions on Automatic Control[12]. This report was compiled by approximately fifty prominent individuals in the controls field and summarizes the current status of controls research, future directions for research, and issues still to be resolved. This report states:

> A more general class of control systems which adapt to significant changes in their environment is the class of fault-tolerant control systems. In this class of problems we admit that one or more key components of the physical feedback system will fail and that this failure can have significant impact on stability or performance. At the simplest level we can think of sensor and/or actuator failures, while at a more complex level we can think of other system failures, e.g., partial structural damage to an aircraft due to a mid-air collision or weapon damage. The idea is to design the control system so as to retain stability and lose performance in a gracefully degraded manner. It may be necessary to reconfigure the control system following the detection of such failures. Such reconfiguration may be as simple as reading a new set of control gains from a precomputed table or as

complex as complete redesigning of the control system in real-time. A challenging problem for control theory is to take into account advances in computer technology and to stimulate the development of real-time and concurrent systems which allow the implementation of such control strategies in hardware form.

What we lack at present is a set of prescriptive methodologies that can be used to design fault-tolerant feedback control systems.

Research Goal and Plan

The goals of this research are to achieve the following:

1. Develop and test an adaptive pole-placement algorithm specifically designed to adapt to actuator failures.
    - i. The algorithm will provide a quick means to recalculate a control law in situations where one or more actuators are changing.
    - ii. The algorithm will maintain the same pole locations after adaption.
    - iii. The algorithm can be used to allow for penalties associated with the use of various actuators and adapt to minimize the use of 'expensive' actuators.
    - iv. The algorithm is suitable for rapid on-line computer calculations.
2. Design principles governing use and implementation of the adaptive pole-placement algorithm will be specified. These principles will enable the user to implement a fault-adaptive system as described.

CHAPTER II

ADAPTION TO FAILURES

Adaptive Pole-Placement: A First Approach

Once the various controllability issues are examined, the question of how to adapt to controller failures needs to be explored. For the sake of the following discussion, assume that a given control structure has been developed through any of several strategies. Examples include optimal control, pole-placement, and frequency response techniques. The result of the control design is a linear feedback law of the form given in equation (3).

Once F has been calculated, the completely specified system has the form:

$$\dot{\underline{x}} = A\underline{x} + B\underline{u} \tag{20}$$

$$\underline{y} = C\underline{x} \tag{21}$$

$$\underline{u} = F\,\underline{x} + \underline{w} \tag{22}$$

with each matrix known. At this point, recall that the use of the feedback can be viewed as a vehicle to move the open-loop poles of system (A, B) to new locations under the new system (A, B, F). One possible reason for moving the poles is to move them from the right half to the left half of the complex plane in order to stabilize system response. However, no matter what reason is used, whether pole-placement or not, the problem can be viewed as having moved the poles from one location to another.

Based on the above discussion, assume that new pole positions were selected to be optimal in some sense, and that the goal is to maintain those pole positions in the presence of controller failures.

The problem can be stated as needing to develop an adaptive pole-placement algorithm that will hold the poles of a system at certain locations despite variations in the B matrix.

Consider the matrices:

$S \equiv$ (n x n) transformation matrix

$G \equiv$ (n x n) eigenvalue goal matrix

where G is a diagonal matrix with target or goal eigenvalues on the main diagonal and zeros elsewhere. The pole-placement problem can now be stated as follows:

Find the feedback matrix F which satisfies:

$$G = S^{-1} (A + BF) S \qquad (23)$$

where the transformation matrix S diagonalizes the closed-loop system matrix $A+BF$. The required feedback matrix can be solved in the least-squares sense as:

$$F = (B^t B)^{-1} B^t (SGS^{-1} - A) \qquad (24)$$

providing $(B^t B)^{-1}$ exists. Unfortunately, this development is flawed since equation (24) has two unknowns (S and F). The matrix S depends on the value of F and hence F is not unique. Furthermore, the solution in (24) is a least-squares solution and other constraints may have to be satisfied before we can say the eigenvalues are located correctly.

The pole-placement problem has received considerable attention through the years and yet, it has remained a surprisingly difficult issue. Numerous papers have been written discussing the calculation of a feedback matrix F to move the poles to new locations [13] [1] [14] [15].

Adaptive Pole-Placement: A Second Approach

From a pole-placement perspective, consider what happens when a failure occurs. The B matrix changes from B to some new input mapping matrix B*. Unless the feedback matrix F changes or adapts to the new B* the poles will move to new, incorrect locations. Clearly, if (A + BF) and (A + B*F*) have the same characteristic polynomial, the system after the failure with F* as a feedback matrix will have identical closed-loop poles as the original system. These results will be obtained if:

$$BF = B^* F^* \tag{25}$$

or if we solve the problem in a least-squares sense with the same remarks as earlier:

$$F^* = (B^{*t}B^*)^{-1} B^{*t}BF \tag{26}$$

However, if $(B^{*t}B^*)^{-1}$ does not exist F^* cannot be found from (26). In fact, the requirement that BF before the failure equals B*F* after the failure is too stringent. (A + BF) can be 'similar' to (A + B*F*) and not exactly equal. Similar matrices have the same characteristic polynomials and thus the same roots which is our goal.

In the case where (26) cannot be used, a different method to obtain F^*

is needed. The goal of this research is to develop a new adaptive pole-placement algorithm to solve this problem.

Brogan's Pole-Placement Method

The general problem of pole-placement can be considered from several different perspectives ranging from transfer function techniques to state-variable techniques. The most promising approach for the problem considered in this research appears to be Brogan's transfer matrix approach[1]. His work results in a series of linear equations which can be solved for the feedback parameters in the F matrix. Also, his approach has the added advantage of allowing the B matrix to be removed from a crucial matrix inversion. This last feature is perfect for an on-line routine because the inversion can be performed off-line in advance. His approach is presented below with a slight modification in the notation used in his paper.

Given a system defined by equations (20-22), the closed-loop characteristic equation can be written as:

$$\Delta'(\lambda) = \left| \lambda I_n - A - BF \right| = 0 \qquad (27)$$

which can be rewritten as:

$$\Delta'(\lambda) = \left| (\lambda I_n - A) \left[I_n - (\lambda I_n - A)^{-1} BF \right] \right| = 0 \tag{28}$$

$$= \Delta(\lambda) \left| \left[I_n - \Phi(\lambda) BF \right] \right| = 0 \tag{29}$$

where I_n is an identity matrix of dimension n and

$$\Delta(\lambda) \equiv \left| \lambda I - A \right|$$

and

$$\Phi(\lambda) \equiv (\lambda I - A)^{-1} \tag{30}$$

The overall problem is to choose F in (29) such that the n specified eigenvalues λ_i corresponding to the closed-loop poles are the roots of (29). Reexpress (29) as:

$$\Delta'(\lambda) = \Delta(\lambda) \left| \left[I_r - F\Phi(\lambda) B \right] \right| = 0 \tag{31}$$

by using the matrix identity:

$$\left| I_n \pm AB \right| = \left| I_r \pm BA \right| \tag{32}$$

In Brogan's approach, the required equality in (31) is achieved by setting the determinant equal to zero for each target eigenvalue. Recall from linear algebra that a sufficient condition for the determinant to be zero is for any row or column to be zero. For Brogan's approach, set a column equal to zero for each target eigenvalue.

Given the j^{th} column of I_r as e_j where e_j is a column vector with in the j^{th} entry equal to one and zeros elsewhere:

$$\Psi(\lambda_i) = \Phi(\lambda_i)B \tag{33}$$

with the j^{th} column being Ψ_j. Then λ_i is a root of $\Delta'(\lambda)$ if F is selected to satisfy:

$$e_j - F\Psi_j(\lambda_i) = 0 \tag{34}$$

since this forces column j of the determinant in (31) to zero. Therefore:

$$F\Psi_j(\lambda_i) = e_j \tag{35}$$

Now find an equation of the form (35) for each target eigenvalue. Together, the n distinct equations will allow the determination of F.

Consider the example problem from Chapter I repeated below for convenience:

$$\dot{\underline{x}} = \begin{bmatrix}1 & 0 & 0\\1 & 1 & 0\\0 & 1 & 1\end{bmatrix}\underline{x} + \begin{bmatrix}1 & 1 & 0\\2 & 0 & 1\\0 & 0 & 1\end{bmatrix}\underline{u} \tag{36}$$

calculate $(\lambda I - A)^{-1}$ as:

$$\Phi(\lambda) = \begin{bmatrix}\frac{1}{\lambda-1} & 0 & 0\\ \frac{1}{(\lambda-1)^2} & \frac{1}{\lambda-1} & 0\\ \frac{1}{(\lambda-1)^3} & \frac{1}{(\lambda-1)^2} & \frac{1}{\lambda-1}\end{bmatrix} \tag{37}$$

now to place the poles at -2, -3 and -4:

$$\Psi(-2) = \begin{bmatrix}-.333333 & 0 & 0\\ .111111 & -.333333 & 0\\ .037037 & .111111 & -.333333\end{bmatrix}\begin{bmatrix}1 & 1 & 0\\2 & 0 & 1\\0 & 0 & 1\end{bmatrix} \tag{38}$$

$$= \begin{bmatrix}-.333333 & -.333333 & 0\\ -.555555 & .111111 & -.333333\\ .185185 & -.037037 & -.222222\end{bmatrix} \tag{39}$$

likewise:

$$\Psi(-3) = \begin{bmatrix} -.25 & -.25 & 0 \\ -.4375 & -.0625 & -.25 \\ .109375 & -.015625 & -.1875 \end{bmatrix} \tag{40}$$

$$\Psi(-4) = \begin{bmatrix} -.2 & -.2 & 0 \\ -.36 & .04 & -.2 \\ .072 & -.008 & -.16 \end{bmatrix} \tag{41}$$

According to Brogan's Method set a column in the determinant equal to zero for each λ. Therefore, try column one for the -2 eigenvector, column two for -3 and column three for -4. Evaluate (35) for each eigenvector:

$$F \begin{bmatrix} -.333333 \\ -.555555 \\ .185185 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \text{for } \lambda = -2 \tag{42}$$

$$F \begin{bmatrix} -.25 \\ .0625 \\ -.015625 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \text{for } \lambda = -3 \tag{43}$$

$$F \begin{bmatrix} 0 \\ -.2 \\ -.16 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \text{for } \lambda = -4 \tag{44}$$

Solve these three equations (42-44) simultaneously and find:

$$F = \begin{bmatrix} -.366 & -1.1156 & 1.3944 \\ -3.5119 & 1.4874 & -1.8593 \\ -.0807 & -1.4364 & -4.4545 \end{bmatrix} \tag{45}$$

This value of F was shown in Chapter I.

Brogan goes on to show that if the λ_i are distinct, there will always be n linearly independent vectors $\Psi_j(\lambda_i)$ to solve for F. Refer to Brogan[1] and Brogan[16] for additional details.

Let us make one more observation before continuing. Since the three equations (42-44) must be solved simultaneously, write one equation to solve as:

$$F \begin{bmatrix} -.333333 & -.25 & 0 \\ -.555555 & .0625 & -.2 \\ .185185 & -.15625 & -.16 \end{bmatrix} = I \tag{46}$$

or

$$F \left[\Psi_{j1}(\lambda_1) \Psi_{j2}(\lambda_2) \ldots \Psi_{jn}(\lambda_n) \right] = \left[e_{j1} e_{j2} \ldots e_{jn} \right] \tag{47}$$

or

$$F = \begin{bmatrix} e_{j1} e_{j2} \ldots e_{jn} \end{bmatrix} \begin{bmatrix} \Psi_{j1}(\lambda_1) \Psi_{j2}(\lambda_2) \ \ldots \ \Psi_{jn}(\lambda_n) \end{bmatrix}^{-1} \tag{48}$$

Since an implied inversion is required, the columns chosen in $\Psi(\lambda)$ must be linearly independent for F to exist.

For notational purposes, define:

$$G = \begin{bmatrix} \Psi_{j1}(\lambda_1) \Psi_{j2}(\lambda_2) \ \ldots \ \Psi_{jn}(\lambda_n) \end{bmatrix} \tag{49}$$

and

$$g = \begin{bmatrix} e_{j1} e_{j2} \ldots e_{jn} \end{bmatrix} \tag{50}$$

therefore solve an equation of the form:

$$F = g \ G^{-1} \tag{51}$$

to find the feedback matrix F.

It is possible for the $\Phi(\lambda)$ equation to result in nonfinite values when goal eigenvalues are substituted into the equation. Brogan's original paper addresses this problem and shows how it can be handled. The infinite value problem does not affect this research so it will not be addressed further here.

The presence of more feedback parameters than states provides for the possibility of choosing feedback to satisfy some criterion. Brogan's work does not address any form of optimization. Clearly, in our example problem, different columns could have been chosen to solve for F.

The freedom in choosing vectors in this problem highlights the under-determined nature of the stated problem.

Brogan's Method for Computer Implementation

By considering Brogan's algorithm from a computer implementation point of view, recognize that the algorithm offers several key advantages that make it ideal for a computer implementation. First, the algorithm offers the possibility of solving the characteristic equation off-line prior to run-time. This ability to take as much computation as possible out of a time-sensitive iteration loop is very important. In fact, any work that can be pushed forward into the off-line, preexecution time is clearly advantageous.

The second computer implementation point, which is really an expansion of the first point, is that off-line work can be done one time as opposed to every time the B matrix changes. This clearly is a desirable attribute of a fast, fault-adaptive computer algorithm.

A third point concerns the final matrix equation that appears in Brogan's algorithm. In that final equation, Brogan states that a user can choose any combination of columns that results in a nonsingular matrix to solve for the feedback matrix F. The key advantage, from a computer perspective, is that Brogan's algorithm guarantees that as long as the resulting matrix is nonsingular, the matrix equation can be solved for F. No additional testing or analysis is necessary and a solution is *guaranteed.*

A fourth point concerns the use of Brogan's method for our stated problem of adapting to actuator faults. If actuator faults are modeled by zeroing columns in B, then when the fault occurs the corresponding columns in $\Phi(\lambda)B$ can be removed. This has the effect of reducing our solution space to a smaller problem, but the same algorithm can be reapplied to calculate a new feedback matrix F if the failed actuator affects system performance.

The last point affecting this research is that Brogan's algorithm states that any linearly-independent combination of columns will result in a solution. This suggests it might be possible to use some additional criterion to determine which of several possible solutions would be advantageous under certain conditions. For example, if the use of one actuator in a given system was overly expensive, and if there were four combinations of columns that lead to feasible solutions, then perhaps one combination might use the expensive actuator much less than the other three combinations. If the choosing of an optimal combination could be included in a computer implementation, then there is a clear improvement over the original algorithm in addition to the adaption improvement.

Rephrasing for Computer Implementation

Consider the following example problem:

$$\dot{\underset{\sim}{x}} = \begin{bmatrix} -2 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 4 \end{bmatrix} \underset{\sim}{x} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \underset{\sim}{u} \tag{52}$$

then

$$\Phi(\lambda) = \begin{bmatrix} \frac{1}{\lambda+2} & \frac{1}{(\lambda+2)^2} & 0 \\ 0 & \frac{1}{\lambda+2} & 0 \\ 0 & 0 & \frac{1}{\lambda-4} \end{bmatrix} \tag{53}$$

Examination of Brogan's algorithm shows that for each target pole position it is possible to have more than one column to choose from for the needed nonsingular matrix G. Let us take advantage of this multi-column possibility by including all possibilities in a format suited for this problem.

Let us create a new matrix called *Z*. This new *Z* matrix will be a partitioned matrix with the number of partitions equaling the system dimension n. Each partition will be populated with columns that are taken by substituting the desired closed-loop pole positions into the $\Phi(\lambda)$ matrix above. For example, if the closed-loop pole positions are to be moved to -5, -4 and -3; the first partition will be the entries where -5 is substituted, the second -4, etc. The resulting Z matrix is given as:

$$Z = \left[\begin{array}{ccc|ccc|ccc} \frac{-1}{3} & \frac{1}{9} & 0 & \frac{-1}{2} & \frac{1}{4} & 0 & -1 & 1 & 0 \\ 0 & \frac{-1}{3} & 0 & 0 & \frac{-1}{2} & 0 & 0 & -1 & 0 \\ 0 & 0 & \frac{-1}{9} & 0 & 0 & \frac{-1}{8} & 0 & 0 & \frac{-1}{7} \end{array}\right] \tag{54}$$

Now post-multiply each partition of the Z matrix by B and form a new matrix X.

$$X = ZB = \left[\begin{array}{cc|cc|cc} \frac{1}{9} & 0 & \frac{1}{4} & 0 & 1 & 0 \\ \frac{-1}{3} & 0 & \frac{-1}{2} & 0 & -1 & 0 \\ 0 & \frac{-1}{9} & 0 & \frac{-1}{8} & 0 & \frac{-1}{7} \end{array}\right] \tag{55}$$

At this point, it should be pointed out that if an actuator (column of B), should fail and need to be zeroed, then the corresponding column in each partition of X can be zeroed. It is not necessary to re-invert a matrix, or re-multiply matrices or make any other calculations. It is this key point that makes Brogan's approach, when cast into this new form, ideal for an adaptive computer implementation. This final X matrix has been our goal to solve the system on a computer.

Recall from Brogan's method Equation (51):

$$F = g\ G^{-1} \tag{56}$$

where:

g = *r x n matrix with columns from I*

G = *n x n nonsingular matrix chosen from X*

F = *Feedback matrix to be evaluated*

The G matrix in (56) is a matrix formed by taking one column from each partition in X. To calculate a feedback matrix F, choose columns from X to create G. The only requirement is that the resulting G matrix must be nonsingular.

As mentioned earlier, Brogan's method does not address the selection of columns. In his approach, he simply states that one should pick columns. For this research, the selection of columns to achieve a feasible solution and finally an optimal solution according to some selected criterion will be considered.

The freedom to pick columns from X increases the complexity of the pole-placement problem. There might be more than one combination of columns which would work for a given set of desired pole locations.

Let us consider the number of combinations of columns that might exist. If a system is n^{th} dimensional, then there will be n partitions in the X and Z matricies. If the B matrix has a column rank of r, then each partition in the X matrix could have as many as r columns. Therefore, since we need to choose once column from each partition, there are r^n possible combinations. Let us define the number of possible combinations as:

$$w = r^n \tag{57}$$

Brogan's algorithm guarantees that if the G matrix is invertible, the resulting F will produce the desired pole positions.

Potential Solution Criteria

Any solution of (56) for the feedback matrix F will be called a 'feasible solution.' It is important now to determine what is going to be called a 'valid' solution. Two criterion for choosing a solution come to mind. The first criterion is to minimize the time to find *any* feasible solution. For an adaptive control scheme, our goal might be to find a feasible solution as quickly as possible. The second solution criterion is to minimize the time to find an optimal solution, in some sense, out of the entire set of feasible solutions. This type of solution might be necessary if the use of some actuators in the system was expensive and minimization of the overall cost of controlling the system was needed.

For this research, both types of solutions will be considered valid and will be examined. The following two chapters address each of these two solution criteria.

CHAPTER III

OPTIMIZING BROGAN'S METHOD FOR FOR FAST COMPUTER SOLUTIONS

Introduction

In a highly failure-sensitive system where it is essential to minimize system downtime, a legitimate control criteria could be to minimize the time necessary to find an alternative feasible control scheme. An example of such a system would be an aircraft flight control system. In this case, there might be several redundant means to reconfigure the control system and maintain acceptable performance. An algorithm to quickly identify an alternative control scheme could be applied.

Given that our goal in a real-time control system is to simply find any feasible solution in minimum time, the following list of alternatives will be considered:

1. Brute Force Selection: Random Selection
2. Brute Force Selection: Modified Random Selection
3. Ranked Selection: Condition Evaluation
4. Phase One Singularity Test
5. Phases One and Two Singularity Tests

Recall, from the last chapter, that the G matrix must be nonsingular to solve for F. As a result, a nonsingular matrix must be *built* from X to have a feasible solution. The first two approaches toward finding a feasible solution involve a form of random trial and error. In each case, a combination of columns from the X matrix is tried until a combination which will result in a nonsingular matrix for inversion is found. The third approach requires the condition number of a possible matrix to be evaluated. If the condition number is below a certain user definable level, then that combination of columns is inverted. The fourth approach involves a test to determine if the columns of the possible matrix are of rank n. This fourth test has the advantage of requiring no multiplies or divides; however, it can not guarantee one hundred percent accuracy. The fifth approach continues where the fourth test ends and can guarantee one hundred percent accuracy. However, the fifth test requires on the order of n^3 multiplies/divides to complete per matrix.

Each of these five solution approaches are presented in detail in the following pages. It will be found that the fourth approach offers the best solution of the five candidates. It can provide extremely good results with a minimum of computer work.

Throughout the rest of this thesis, the work requirements of an algorithm will be measured by the number of multiplies and divides in the algorithm. This measure of an algorithm's requirements is the traditional measure used throughout the literature. It is also based on the practical consideration that multiplies and divides on a computer generally require many more computer operations than additions or indexing operations.

Brute Force Selection: Random Selection

It was shown in the previous chapter that there are $w = r^n$ possible sets of solutions that might yield a feasible solution in the problem as we have formulated it. The simplest solution approach which comes to mind is to throw infinite computer resources at the problem and solve each possible solution set for a feedback matrix F which will result in the correct pole positions. If during solution of the problem the G matrix is shown to be singular, then continue to the next combination of columns from the X matrix.

Advantages

This approach has several advantages. First, it is a conceptually simple approach to finding a solution as it is necessary only to solve equation (56) for the feedback matrix F using a row reduction method. The second advantage is that it requires no additional work or preprocessing to determine if the solution is feasible.

<u>Disadvantages</u>

There are, unfortunately, several disadvantages to the Brute Force approach. As is well known, inverting a matrix is an expensive operation numerically. Recall from elementary numerical analysis that as many as $\frac{1}{3}n^3$ multiplies/divides can occur before it is known whether or not the matrix is singular. In this case it might be necessary to invert w matrices once each. Furthermore, since w is a function of n, the worst case cost of the Brute Force approach in the problem as defined could run as high as $\approx\frac{1}{3}n^3r^n$.

It is apparent that the search for a feasible solution will probably end before every candidate is examined. However, if it is necessary to guarantee a fast solution, then the worst-case cost identified above must be considered.

A second problem with this approach is the random nature the w matrix equations are being solved. Clearly, it might be advantageous to analyze the equation for feasibility before solving for F. Some form of ranking might then suggest an order to attempt to solve the equations.

In general, a random solution of equation (56) should be avoided whenever possible. A better solution than the Brute Force method is necessary.

Brute Force Selection: Modified Random Selection

Due to the inherent structure of our X matrix, recognize that corresponding columns in each partition will tend to be correlated. This is due to the numerical values in those columns having been calculated from the same $\Phi(\lambda)$ column. As a result, perhaps the likelihood of forming a matrix with dependent vectors can be reduced if combinations using corresponding columns are avoided.

For example, consider a third-order system that has three separate actuators and thus three columns in B. When the X matrix is formed, the result will be a three partition matrix with three columns in each partition. The following figure lists a possible binary search strategy that would choose the first column in each partition on the first attempt with succeeding candidates chosen in a binary manner.

1	1	1
1	1	2
1	1	3
1	2	1
1	2	2
1	2	3
1	3	1
1	3	2
1	3	3
2	1	1
2	1	2
2	1	3
2	2	1
2	2	2
2	2	3
2	3	1
2	3	2
2	3	3
3	1	1
3	1	2
3	1	3
3	2	1
3	2	2
3	2	3
3	3	1
3	3	2
3	3	3

However, if the modified approach of attempting 'off column' entries is used, then the following search pattern, one of several possible, might be used:

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1
1	1	2
1	1	3
1	2	1
2	1	1
3	1	1
1	3	1
1	2	2
2	1	2
2	2	1
2	2	3
2	3	2
3	2	2
1	3	3
2	3	3
3	1	3
3	2	3
3	3	1
3	3	2
1	1	1
2	2	2
3	3	3

Clearly, the last three choices are the most likely to produce a singular G and are tested only if all others fail.

Advantages

If the binary search strategy and the modified search strategy are implemented on a computer under simulated conditions, the modified search strategy does generally find a feasible solution faster. In numerous simulations, the modified search strategy resulted in a feasible solution on the first attempt approximately eighty percent of the time. Unfortunately, the performance improvement is not significant enough to end our search for a better approach. However, the modified approach requires little additional overhead over the straight Brute Force approach and might be suitable for some situations.

An analysis of the modified approach does point out that the search strategy used to find a feasible candidate G matrix can impact on the algorithm's performance. This important point will be revisited later.

Disadvantages

The disadvantages of this approach are the same as for the Brute Force approach.

Ranked Selection: Condition Evaluation

If the type of solution desired is 'any feasible solution,' then a preprocessing step might be able to order the candidates according to their likelihood of resulting in a feasible solution. If such a ranking process were effective, only one equation would need to be solved. An approach that comes to mind involves estimating the condition number of the given candidate G matrix for each equation. Based on the calculated condition numbers, the inversion process could begin by starting with the equation that is most likely to yield a satisfactory solution, or the first matrix with an acceptable condition number.

The following condition number material is included because the condition of a matrix is an often used measure of how close a matrix is to being singular. As such, the feasibility of using this measure should be evaluated to see if it has potential in our adapting problem. In the end, it will be found that a better approach is available.

Numerous sources develop the theory behind evaluating the condition number of a matrix[17] [18]. Therefore, this work will not present the theory behind condition numbers except where shown to be useful in the following pages.

Recall that the condition number of a matrix is defined as:

$$cond(A) = ||\ A\ ||\ ||\ A^{-1}\ || \tag{58}$$

where $||\ A\ ||$ represents a norm of the matrix A. Generally, the following norm is used because of its simplicity:

$$||\ A\ ||_{\infty} = \max \frac{||\ A\,x\ ||_{\infty}}{||\ x\ ||_{\infty}} \tag{59}$$

$$= \max_{1 \le i \le n} \sum_{j=1}^{n} \left| a_{ij} \right| \tag{60}$$

$$= Maximum \quad Row \quad Sum$$

Throughout the rest of this development, $|| A ||$ will mean $|| A ||_\infty$.

Unfortunately, to calculate the cond(A) exactly, A^{-1} must first be calculated. As a result, approximation techniques are generally used instead of the definition directly.

<u>Random Vector Approach</u>

The condition number approach considered here begins with the $A\underline{x} = \underline{b}$ and A being invertible, then:

$$\underline{x} = A^{-1} \underline{b} \tag{61}$$

and

$$|| \underline{x} || \le || A^{-1} || \; || \underline{b} || \tag{62}$$

or

$$\frac{||\ \underline{x}\ ||}{||\ \underline{b}\ ||} \leq ||\ A^{-1}\ || \tag{63}$$

Therefore, let us choose k arbitrary $\underline{b}$ vectors, solve for $\underline{x}$, and then evaluate:

$$||\ A^{-1}\ || \geq \max_{1 \leq p \leq k} \frac{||\ x(p)\ ||}{||\ b(p)\ ||} \tag{64}$$

from our set of arbitrary $\underline{b}$. The overall condition number can then be approximated as:

$$cond(A) \geq ||\ A\ || * \max_{1 \leq p \leq k} \frac{||\ x(p)\ ||}{||\ b(p)\ ||} \tag{65}$$

In review, this approximation approach requires the following steps:

1. Calculate $||\ A\ ||$.
2. Choose k arbitrary $\underline{b}$ vectors.
3. Solve for k $\underline{x}$ vectors.
4. Calculate $\max_{1 \leq p \leq k} \frac{||\ x(p)\ ||}{||\ b(p)\ ||}$.
5. Approximate cond(A) as: $cond(A) \geq ||\ A\ || * \max_{1 \leq p \leq k} \frac{||\ x(p)\ ||}{||\ b(p)\ ||}$.

This approach can be easily implemented as a computer algorithm. However, this technique requires many multiplies/divides per candidate G matrix. Specifically, to solve the first system composed of an arbitrary $\underline{b}$ vector and the A matrix using a triangular factorization approach such as LU Decomposition it is necessary to perform:

$$O(n^3) \text{ multiplies/divides}$$

in the initial factorization phase followed by:

$$kO(n^2) \text{ multiplies/divides}$$

in the solving phase or $O(n^2)$ for each of the k arbitrary $\underline{b}$ vectors. The amount of work necessary for this condition number evaluation seems a bit excessive for our requirement of a fast, on-line, adapting algorithm. As a result, the condition number material will be put aside for now to determine if another technique might result in less work.

Phase One Singularity Test

At this point, a completely different approach will be considered in determining a combination of vectors (columns) from X that will result in a feasible solution to our matrix equation. In general, there are two approaches that can be followed in the analysis of a problem. One approach would be to determine an exact deterministic answer to the given question by using an algorithm that yields conclusive results. Our Brute Force method discussed in prior sections is such an approach because it involves attempting solutions to determine if the candidate matrix was singular. A second approach that can be taken is to analyze the problem to yield an answer in a probabilistic sense. Such an approach might involve the use of heuristics or simplier algorithms that are less expensive to use. For example, if a simple test could be applied to determine that a given candidate matrix was nonsingular with ninety-five percent probability, then that information might be useful for our fault-adaptive problem. For this Phase One Test, the second approach will be adopted.

In the following pages, several elementary linear algebra properties will be examined to find a series of tests for computer implementation. The tests being sought must be simple and well suited for fast implementation on a digital computer. In the end, a five step test will be developed to evaluate whether a given candidate G matrix is singular. The first four tests will prove to be inexpensive to implement and will achieve excellent results.

Two Vector Dependency Test

Recall from elementary matrix theory that a nonsingular matrix A of dimension n has exactly n linearly independent rows and n linearly independent columns. That is:

$$n = \text{Rank}(A)$$

Thus, if an inexpensive means can be found to determine the rank of a candidate G matrix, the singularity of the matrix can be determined without an inversion.

Let us begin this approach by developing sufficiency tests to determine if two vectors are linearly dependent. This two vector test will be applied against combinations of column vectors in our candidate G matrix. Recall from elementary linear algebra that if:

$$S = \left\{ \underline{v}_1, \underline{v}_2, \cdots \underline{v}_n \right\} \tag{66}$$

is a set of vectors and if a vector equation is created as:

$$k_1 \underline{v}_1 + k_2 \underline{v}_2 + \cdots k_n \underline{v}_n = \underline{0} \tag{67}$$

the vectors $\underline{v}_1$, $\underline{v}_2$, ... $\underline{v}_n$ are linearly independent if, and only if, the only solution to equation (67) is:

$$k_1 = k_2 = \cdots k_n = 0 \tag{68}$$

The vectors are linearly dependent otherwise.

Consider the two vectors $\underline{v}_1$ and $\underline{v}_2$ then:

$$k_1 \underline{v}_1 + k_2 \underline{v}_2 = \underline{0} \tag{69}$$

is the required vector equation as in (67). First, recognize that the vectors $\underline{v}_1$ and $\underline{v}_2$ cannot be zero vectors because the corresponding k_i could then assume any value which violates the required condition for independence. Therefore:

$$\underline{v}_1 \neq \underline{0} \tag{70}$$

$$\underline{v}_2 \neq \underline{0}$$

The first step in our two vector test for dependency is a test to determine if either vector is a zero vector. Such a test is extremely simple to implement on a computer and thus it meets the computer criteria.

For the second step, the vector equation (69) can be rewritten without loss of generality as:

$$b\,\underline{v}_1 + \underline{v}_2 = \underline{0} \tag{71}$$

$$b\,\underline{v}_1 = -\,\underline{v}_2 \tag{72}$$

Now consider each entry (row) in the vectors separately. In this case:

$$b_i = -\frac{v_2[\,i\,]}{v_1[\,i\,]} \qquad i = 1,\ 2,\ \cdots\ n \tag{73}$$

Now for equation (73) to be true with $b \neq 0$ then all the b_i must be equal.

$$b_1 = b_2 = \ldots\ b_n = b \tag{74}$$

If any two b_i are not equal, then the vectors are independent.

If after making and passing the tests for zero vectors, and if one entry of either vector is zero while the corresponding entry in the other vector is not zero, equation (73) will have no finite solution. There is no finite value by which to multiply the first vector to equal the second vector for that given entry. Therefore, the two vectors are independent. This test will be used to stop dependency testing because it is known that the vectors are independent. This test to examine corresponding entries in vectors for their zero content will be step two of our testing.

The second step also meets the criteria of being easy to implement on a computer.

So far in our development, an algorithm has been created that looks at zero content of the candidate G matrix to determine if given pairs of vectors in the matrix are linearly dependent. The specified approach made no attempt to determine if two vectors were linear multiples of each other beyond the examination of zero placement. Let us begin to determine if the two vectors are multiples of each other. Consider the following two vectors:

$$\underset{\sim}{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ a \\ 0 \\ 0 \end{bmatrix} \qquad \underset{\sim}{y} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ b \\ 0 \\ 0 \end{bmatrix} \tag{75}$$

In this case, if a and b are non-zero, the two vectors will be linearly dependent because they are multiples of each other. The dependency can be identified without performing any multiplies/divides by counting the number of times a pair of nonzero entries appear in the same row. If the count is one, then the two vectors are linear multiples of each other. Realize that this testing method must follow the first two tests in order to be valid.

This step of counting the entries that have nonzero elements will be a third step in the test being developed.

The first three steps are based on pure linear algebra principles and they can easily be implemented on a digital computer. By applying these three steps against a large sample of candidate matrices approximately twenty-one percent of the time the tests would suggest that a candidate G matrix is nonsingular, when in fact it is singular. These twenty-one percent false positives still result in excessive waste. Another step is needed.

Upon examination of the false positives that are occurring, a noticeable characteristic can be identified. In many cases, there are two or more column vectors that are exactly equal. If this were a pure linear algebra problem with candidate matrices that were purely random, such duplication would not be expected. However, this is a specific control problem that is being based on the platform established in Brogan's Method. Therefore, something in that method is resulting in an increase in duplicate column vectors. It is this additional characteristic specific to our problem that will provide us with the vital fourth step. Later in this chapter, the cause of the duplicate vectors will be discussed.

The fourth step will check to see if two vectors that have passed the previous three steps are equal. In this case, the first vector can be multiplied times one to equal the second vector. This simple test can be executed by subtracting corresponding entries and determining if the difference is zero for each pair of entries in the vectors. If the difference is zero for each entry, then the vectors are equal and the G matrix will be singular. The application of this last step effectively reduced the level of false positives to three percent as shown in numerous simulation studies shown later.

The four-part test gives us a simple sufficiency test to determine if two vectors are linearly dependent. While the test does examine the necessary conditions for the vectors to be linearly independent, it is not sufficient to guarantee that the vectors are linearly independent. An additional generalized test to determine if the vectors are multiplies of each other is required to determine if the vectors are linearly independent. This further test will be discussed later.

Phase One Test

The two vector test can be used for evaluating vector dependencies in our problem by applying it against each combination of column vectors in our candidate G matrix. If, at any point in our application of this test, two vectors are found to be linearly dependent it will be known that the candidate G matrix is singular because the column rank is not equal to the dimension of the matrix G. The application of this two vector test against all the column vectors in our matrix will be called the Phase One Test throughout the rest of this discussion.

Advantages of Phase One Test

The principle advantage of the Phase One Test is that it requires no multiplies/divides to determine its answer. The only operations that are necessary are compares, indexing and additions/subtractions. This lack of multiplies and divides can result in a very fast approach to determining if a given G matrix might be feasible. Since our goal in this section is to minimize the amount of time necessary to find a feasible solution, then this test looks promising.

Disadvantage of Phase One Test

The only disadvantage to this test is that if a given matrix should pass the Phase One Test it is still not guaranteed to be nonsingular although the percentage of false positives is less than three percent. It is still necessary to execute the inversion, or another test, to achieve certain results. However, from a probabilistic point of view, this test will rule out many non-feasible solutions very quickly. Later, under simulated conditions the test will be shown to yield extremely good predictions of candidate feasibility. On the following page, Figure 2 outlines the Phase One Test as described in the previous pages.

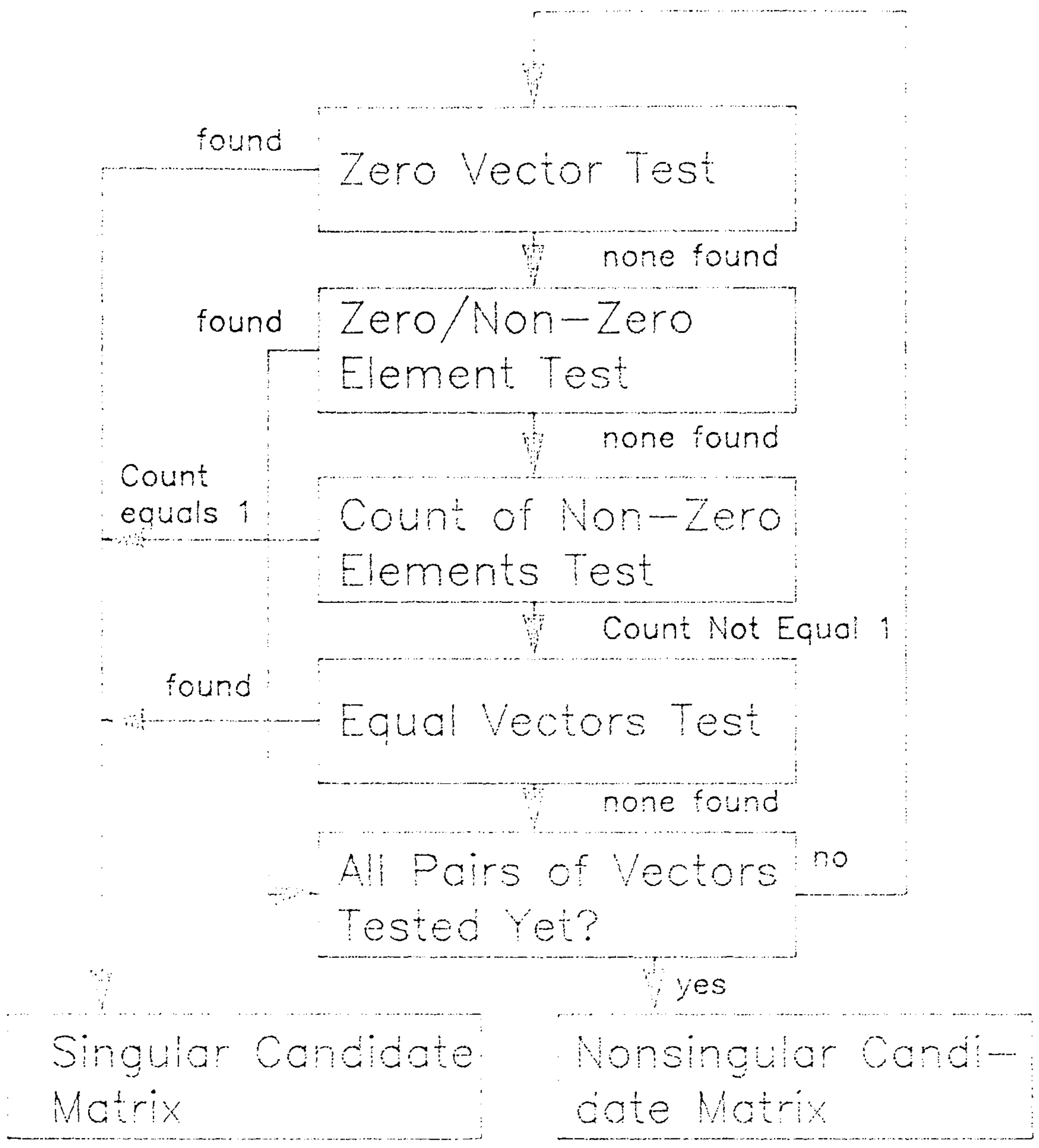
found
Zero Vector Test
none found
found
Zero/Non-Zero Element Test
none found
Count equals 1
Count of Non-Zero Elements Test
Count Not Equal 1
found
Equal Vectors Test
none found
All Pairs of Vectors Tested Yet?
no
yes
Singular Candidate Matrix
Nonsingular Candidate Matrix

Figure 2. Phase One Test Outline

Phase One and Two Singularity Tests

This last test to determine if a given candidate G matrix is nonsingular will guarantee an accurate answer. Unfortunately, the price of this guaranteed answer is considerable computer work.

Recall from the end of the development on the Phase One Test that to provide necessary and sufficient proof that the given vectors are linearly independent, it was necessary to test the vectors to determine if they were linear multiples of each other. To verify this last condition, begin with equation (73) which is repeated here for convenience:

$$b_i = -\frac{v_2[\,i\,]}{v_1[\,i\,]} \qquad i = 1,\ 2,\ \cdots\ n \tag{76}$$

Recall from above that all the b_i must be equal in order for the two vectors to be linearly dependent. This last test will simply solve equation (76) for each value of b_i. If the b_i are all equal, then the two vectors are linear multiples of each other and thus linearly dependent. If any of the b_i differ, then the two vectors are linearly independent.

This last test completes the necessary and sufficient conditions to state that the two vectors are linearly independent. This last step in the testing procedure will be called the Phase Two Test.

Let us analyze the Phase Two Test to determine how many multiply/divide operations are required to completely analyze a given candidate G matrix. For each combination of two vectors there will be:

n Divides Per Two Vectors

To examine all the combinations of columns will require test runs against:

$$\frac{n!}{2!\,(\,n\,-2\,)!} \tag{77}$$

combinations of vectors. Therefore, the number of required divides is:

$$\frac{n}{2} * \frac{n!}{(\,n\,-2\,)!} \tag{78}$$

This calculated number of divides assumes that the matrix is not shown to be singular before all the divides are done. However, as a worst case, consider the number identified here.

Advantages and Disadvantages

This application of the Phase Two Test is a very expensive test to assure that there are no false positives. The advantage of this approach, when coupled with the Phase One Test, is that it guarantees the resulting matrix will be nonsingular and yield a feasible solution. The disadvantage is that many expensive computer operations will be spent to reach that 'guaranteed' answer. In fact, the work spent in the Phase Two Test is comperable to solving the problem directly. Recall that the front-end of a matrix inversion is approximately a $\frac{1}{3}n^3$ operation. The Phase Two test is approximately a $\frac{1}{2}n^3$ operation. Therefore, the ratio of multiplies/divides between the two alternatives is:

$$work \quad ratio = \frac{testing}{solving} \approx \frac{\frac{1}{2}n^3}{\frac{1}{3}n^3} = \frac{3}{2}\frac{n^3}{n^3} \tag{79}$$

Since the testing process is comparable to the solving process, it does not make sense to use the testing procedure.

Implementations, Comparisons and Recommendations

In previous sections, several different means to find a feasible solution to the control problem have been examined. In this section, the Brute Force approach and the Phase One Test will be implemented under simulated conditions. Such simulations will provide us insight as to when the various approaches will or will not work. Furthermore, these results might suggest additional approaches or combinations of approaches which would yield greater improvement.

Example Problem

Let us begin by considering an example problem. Later, generalization will be considered. Let A be:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Let us populate a 3x3 B matrix in a random nature as follows:

1. Let each entry in the B matrix be independent of the other entries.
2. Let each entry equal zero with probability p.
3. Let each entry equal a random number between -20 and 20 with probability 1-p.

These assumptions on the entries of the B matrix are reasonable in that each column of the B matrix represents the input from a separate actuator. Consequently, there are generally no predeterminable correlations between these separate actuators. Exceptions to these assumptions can occur when redundant actuators are present and in some network problems where order in the B matrix can appear. The redundant actuator situation will be discussed later.

For the rest of this development, p will be referred to as the zero density of the matrix. Let us then calculate the X matrix for each successive B matrix and also select columns from X for our candidate G matrix in the random nature described in the Brute Force section.

If three thousand different B matrices, three columns wide, are used in a simulation run where p = 0.3 the results are shown in Table I.

TABLE I

CANDIDATES ATTEMPTED BY
BRUTE FORCE SELECTION

Candidates Attempted	**Occurrences**	**Percentage**
1	2135	71.2
2	570	19.0
3	80	2.7
5	121	4.0
14	22	0.7
15	30	1.0
18	15	0.5
27	27	0.9
total	3000	100.0

Table I shows that of the 3000 different B matrices, 2135 times the Brute Force method yields a feasible solution with the first candidate. In 570 of the 3000 cases, two candidates were examined to produce a single feasible solution, etc.

Since this is a third-order system, it is known that approximately $\frac{1}{3}n^3$ or 9 multiplies/divides are needed for the first reduction. This number is based on row reduction and the number of operations necessary to evaluate singularity. Consequently, 2135 attempts required 9 multiplies, 570 attempts required 18 multiplies, etc.

By calculating the average multiplies that were necessary for each new B matrix the Brute Force Method required:

17.6 Multiplies per Successful Solution

By repeating this analysis for different values of p, the results shown in Table II are obtained. In Table II, the '% Wasted' column refers to the number of times an row reduction is attempted that does not lead to a feasible solution.

TABLE II

BRUTE FORCE RESULTS FOR VARYING ZERO DENSITY

Zero Density	Average Multiplies	% Wasted
0.3	17.6	48.9
0.4	21.3	57.7
0.7	77.9	88.4

The results in Table II are not surprising. As the zero density increases, more of the columns will tend to be dependent, and the Brute Force approach must look at more and more candidate matrices to find a feasible solution.

Given that the previous results are based on a Brute Force approach, consider what would happen if the Phase One Test was applied before attempting the row reductions. Table III presents the results when the zero density equals 0.3.

TABLE III

PHASE ONE TEST RESULTS
EXAMPLE PROBLEM

Occurrences	**Predicted Result**	**Actual Result**	**Percentage**
1078	Not Feasible	Not Feasible	35.9
127	Feasible	Not Feasible	4.2
1795	Feasible	Feasible	59.8
0	Not Feasible	Feasible	0
3000			

For this example problem, the Phase One Test correctly ruled out 36% of the candidate matrices. Furthermore, the test resulted in no false negatives and only 4% false positives. Overall, the test was correct 95.7% of the time. Let us apply this predictive accuracy back against the Brute Force approach. Consider another test run where the row reductions will not be executed unless the candidate matrix passes the Phase One Test. Again, this test is for a zero density of 0.3 and the results are shown in Table IV.

The number of first time successes has increased from 71% to 98%. Previously, almost 50% of our CPU time was wasted on candidate matrices

TABLE IV

CANDIDATES ATTEMPTED AFTER
PHASE ONE TEST

Candidates Attempted	**Occurrences**	**Percentage**
0	26	0.9
1	2930	97.7
2	44	1.5

that were not going to work, now the number has been cut back to 0.6%. Note that Table IV is the 'after' table associated with Table I.

Let us summarize the comparison between the Brute Force Approach and the Phase One Test followed by row reductions in Table V.

TABLE V

COMPARISON OF BRUTE FORCE VS.
PHASE ONE TEST

	BF	**POT**	**ideal**
Avg. Multiplies	17.6	9.05	9
% Wasted Inversions	48.9	0.6	0
Total Runs	3000	3000	-

Clearly the addition of the simple Phase One Test before attempting a solution can save a significant amount of CPU time in this example problem.

Generalization of Phase One Test Effectiveness

At this point, it must be asked if it is possible to generalize the results of the example problem. The question centers on the ability to derive a closed-form expression for the effectiveness of the Phase One Test versus a Brute Force selection approach. Let us now consider what factors influence the effectiveness of the test. The first factor to consider is duplicate target eigenvalues. Recall that the X matrix is a partitioned matrix formed by concatenating $\Phi(\lambda_i)\,B$ matrices from each target eigenvalue λ_i. As a result, the same column in each partition is formed from the same column in $\Phi(\lambda)$. If a matrix has partitions that are the same, then the fourth step of the Phase One Test will detect the duplicate vectors. Consider the following problem where A is defined as:

$$A = \begin{bmatrix} -2 & -2 & 0 \\ 0 & 0 & -3 \\ 0 & -3 & -4 \end{bmatrix}$$

and the target eigenvalues of -4, -4 and -10 are chosen. Then

$$Z = \left[\begin{array}{ccc|ccc|ccc} -.5 & 0 & .333 & -.5 & 0 & .333 & -.125 & -.0222 & .00396 \\ 0 & 0 & .333 & 0 & 0 & .333 & 0 & -.0952 & .01587 \\ 0 & -1 & -1.333 & 0 & -1 & -1.333 & 0 & -.4762 & -.1587 \end{array}\right]$$

Clearly the first two partitions are correlated because they are exact duplicates. If our algorithm is building a candidate G matrix, it would find that all the combinations which have the corresponding columns in the first two partitions will result in non-feasible solutions. Therefore, the algorithm effectiveness is influenced by the presence of duplicate target eigenvalues.

A second factor influencing the performance of the Phase One Test is the structure of the $\Phi(\lambda)$ matrix itself. As another example, consider the following $\Phi(\lambda)$ matrix:

$$\Phi(\lambda) = \begin{bmatrix} \frac{1}{\lambda + 1} & 0 \\ \frac{1}{(\lambda + 1)^2} & \frac{1}{\lambda + 1} \end{bmatrix}$$

No matter what target eigenvalues are chosen, the second columns of each partition will be multiples of each other and thus the vectors will be dependent. In this case, step three of the Phase One Test will detect the dependency and eliminate the corresponding candidate G matrix. Thus, the effectiveness of the algorithm is a function of the structure of the $\Phi(\lambda)$ matrix which is a function of the structure and the values in the system matrix A.

Unfortunately, the structural mapping between the A matrix and the $\Phi(\lambda)$ matrix is through an inversion that complicates the derivation of any closed-form expressions.

A third influencing factor is the strategy used to select vectors from each partition. In most of this work, a simple binary search strategy has been used; however, as discussed earlier, the modified random search might be more efficient. In any case, different strategies will influence the performance of the Brute Force approach as well as the Phase One Test. How to account for variations in strategies in a closed-form solution is a difficult question.

The entries in the B matrix provide a fourth influencing factor that further clouds the effectiveness issue because strategically-placed zero entries can influence candidate feasibilities. As mentioned earlier, the Phase One Test greatly depends on zeros and zero placements. Accurate and meaningful accounting of zero placement in the B matrix and its influence on algorithm performance appears to be a difficult goal.

Due to these complicating factors, it does not appear feasible to evaluate the effectiveness of our algorithm against the Brute Force approach in a closed-form, deterministic fashion.

However, three arguments for the general use of this approach and for its effectiveness can be made. First, it makes intuitive sense that such a preprocessing step should be effective in reducing the amount of work otherwise necessary. It is known that the solution process is an expensive operation to run on a computer in that it requires $O(n^3)$ multiplies/divides to perform. If a preprocessing step that requires *no* multiplies/divides to eliminate some candidate matrices that would otherwise be inverted can be used, then such a step should be advantageous.

A second argument for use of this test is that the preprocessing step is based on fundamental linear algebraic principles. There is no guesswork involved. By application of these principles, candidate matrices are ruled out in a mathematical manner.

The third argument supporting use of the Phase One Test can be found in the results of extensive Monte Carlo simulations that evaluate effectiveness of the algorithm against thousands of potential systems. Let us run the Brute Force approach against 100,000 separate A and B pairs to determine the performance of the algorithm. By randomly populating an A matrix, a B matrix and a vector containing the target eigenvalues the test can be repeated many times. The results of such a simulation run on a third order system are shown in Table VI.

TABLE VI

LARGE SCALE SIMULATION RUN
100,000 SYSTEMS

	Brute Force	**Phase One Test**
# Multiplies	3,137,616	1,169,190
Mults/Matrix	31.4	11.7
Unnecessary Inversions	248,624	29,910
% Feasible First Time	66.6	94.6

The key observation from this simulation run is that the Brute Force approach required 8.31 times more unnecessary inversions per matrix. This large number is attributable to the Brute Force approach searching the entire solution space looking for a solution when there were no feasible solutions at all. The Phase One Test in those cases might try zero, one or a few candidates and avoid testing the entire solution space. The reduction in unnecessary inversions should prove significant if higher order systems are considered.

Before leaving this section, let us determine how often the Phase One Test is giving us the correct answer. Once again, let us simulate many example problems and calculate the experimental success rate. The results are shown in Table VII.

TABLE VII

PHASE ONE TEST RESULTS: 100,000 SYSTEMS

Occurrences	Predicted Result	Actual Result	Percentage
44,676	Not Feasible	Not Feasible	44.7
2,815	Feasible	Not Feasible	2.8
52,509	Feasible	Feasible	52.5
0	Not Feasible	Feasible	0

Several points should be discussed in the closing of this section. First, from our large-scale simulation runs, 97% accuracy is achieved with the implementation of a simple four-step algorithm. This result is surprisingly good for such simple tests. The key to this success is the fourth test which takes advantage of a particular characteristic in Brogan's algorithm that was discussed earlier.

A second point concerns the elimination of candidates which would yield singular solutions. Our simulations reveal that use of the Phase One Test eliminates 44.7% of the candidate matrices that otherwise would be considered.

The third important point is that there are no false negatives. This factor is extremely important because no feasible candidates should be ruled out accidentally. This point will be further developed in the next chapter.

From our simulations, a fourth significant point is found. When the Phase One Test is used, the first candidate matrix is feasible 94.6% of the time as opposed 66.6% of the time in the random approach. This is important when trying to find any feasible solution in a minimum amount of time which was the goal of this chapter. The fifth chapter will draw the Phase One Test together with Brogan's algorithm and apply the results to the fault-adaptive problem. The sixth chapter will summarize the performance of our new approach.

There is one final point worth mentioning. As the simulations reveal, there are approximately three percent of the candidates that pass the Phase One Test yet still are singular. Upon examination of those candidates, it is found there are no easily identifiable *visual* clues left in those matrices to suggest that they are singular. In other words, our four steps manage to extract all the easy clues which suggest that a matrix is singular. The clues of zero vectors, duplicate vectors, or single non-zero vectors are all detected by our tests. This result is appealing.

Time Evaluation of the Phase One Test versus the Brute Force Method

In the previous sections, the comparisons between the Phase One Test and the Brute Force Method have been by an analysis of the multiply/divide operations necessary to arrive at an answer. As mentioned earlier, the multiply count is the generally used way of measuring the work in an algorithm. However, with faster and faster computers and the trend to using math coprocessors, it is also necessary to evaluate the algorithms in a time sense. Clearly, if the Phase One Test used many, many more indexing and comparison operations to eliminate the multiply operations, then it might actually take longer to run on a computer. Such a characteristic would make the Phase One Test unacceptable.

The chart shown on the following page is the result of computer simulations of example problems of system dimension three through ten. For each different system dimension, two hundred example problems were tested using the Phase One Test and the Brute Force Method. The simulations were run on an AT&T 3B15^{TM} computer with a math coprocessor. The compiler was the standard UNIX^{TM} System V C compiler.

While two hundred simulation runs each is not sufficient to draw any concrete conclusions, it is enough to see the resulting trend. As can be seen, the Phase One Test ran in less time in each of the simulation runs. In fact, when the Phase One Test is used prior to attempting the required row reductions, the solution time is less than half of the time required for the Brute Force Approach. This results agrees well with the results of Table V where the Phase One Test is shown to reduce the required multiplies in half. Careful inspection of the results also suggests that use of the Phase One Test allows a problem of one dimension greater to be solved with the same computer resources.

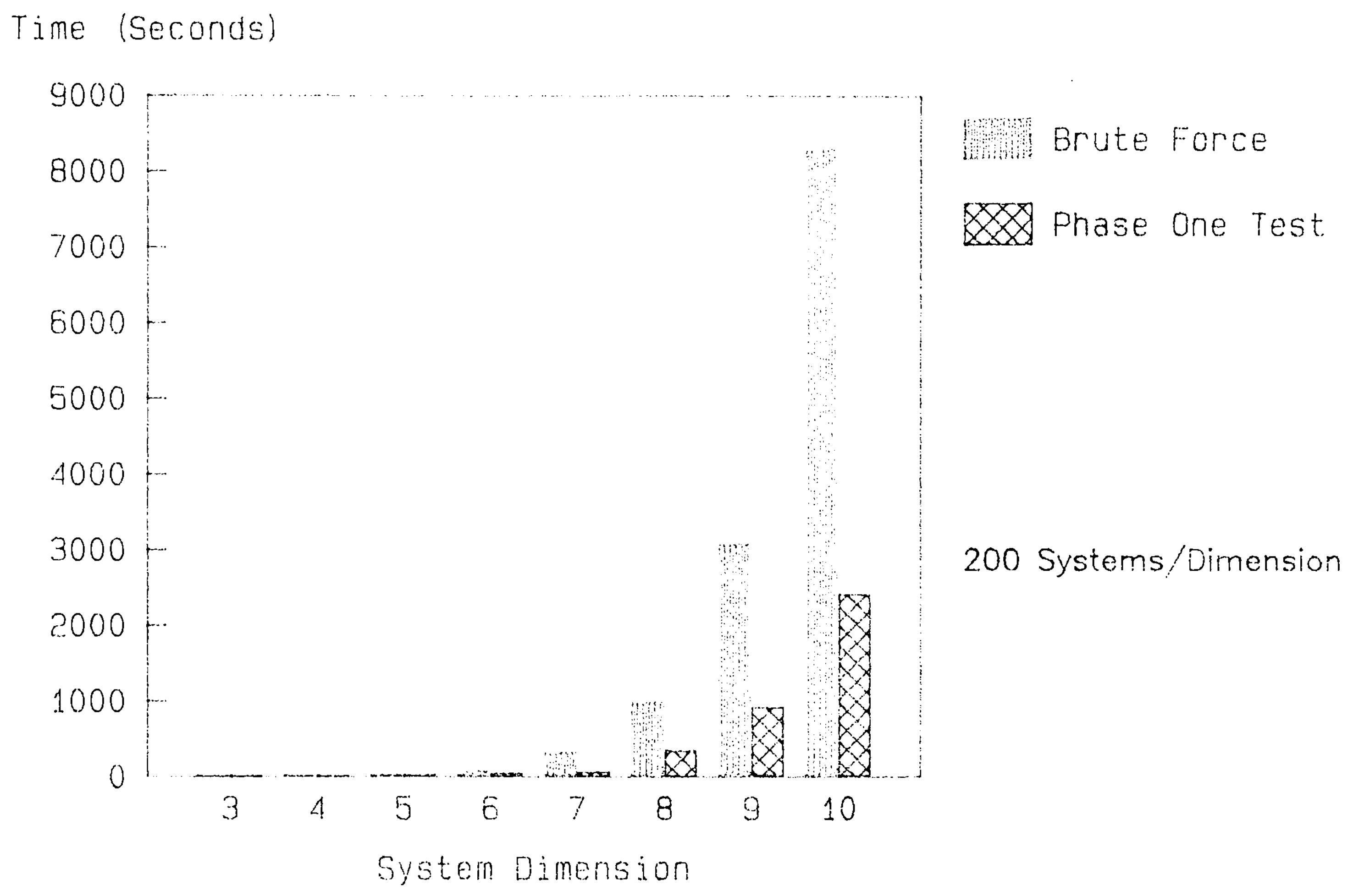

Figure 3. Calculation Time vs. Dimension

Therefore, the various simulations used in this research suggest that the Phase One Test can greatly reduce the number of multiplies needed to solve for the feedback matrix F and the Phase One Test can solve the problem in less time.

Summarizing the Phase One Test

To close this chapter, the Phase One Test is summarized in the following table:

TABLE VIII

PHASE ONE TEST SUMMARY

Step	Description	Add/Sub	Mult/Div
One	Zero Vector Tests	no	no
Two	Zero/Non-Zero Element Test	no	no
Three	Count of Non-Zero Elements Test	no	no
Four	Equal Vectors Test	yes	no

The Add/Sub column indicates the need for floating point additions or multipys and the Mult/Div column indicates the need for floating point multiplies and divides.

CHAPTER IV

FINDING A BETTER SOLUTION

Introduction

The previous chapter addressed the question of trying to find any feasible solution as quickly as possible. This chapter addresses the question of finding a 'best' solution out of many feasible solutions in a minimum amount of time.

Two approachs will be considered. The first approach is to search through the feasible solution set of Brogan's formulation using the Phase One Test followed by cost evaluation of feasible candidates and finally solution selection. The second approach is to use a more intelligent search strategy to find a solution using a non-linear least squares algorithm on a different form of the feedback problem. While the first approach is a simple solution to the problem for some cases, the second approach will generally find better solutions.

Full Search Using the Phase One Test

When finding an optimal solution to a given control problem, it is necessary to be able to prove that any other possible combination of actuators in the given problem leads either to a non-feasible solution or a solution costing more than the calculated optimal solution. In that case, it becomes necessary for us to examine, in some sense, every possible combination of actuators. If certain combinations of actuators can be identified as leading to non-feasible solutions, then the expense of finding an optimal solution can be reduced by avoiding the cost calculation for those combinations.

In previous discussions, it was determined that there were w possible combinations of matrices that could yield feasible solutions to equation (51). Let us consider those w possible solutions as our entire solution space.

$$W = \text{Solution Space}$$

Let us define a subset of W called Q that consists of those matrices in W that will actually yield feasible solutions to the matrix equation (51).

$$\text{Feasible Set} \equiv Q \subset W$$

If the set Q can be distinquished from the set W where Q is smaller than W then our problem has been reduced from the entire solution space to a smaller feasible solution space. Furthermore, if it can be guaranteed that the set of solutions in the space W - Q are not feasible and that the i^{th} combination in Q is the optimal solution over the space Q, then it can be stated that the i^{th} solution is the optimal solution overall.

Keep in mind that the work necessary to identify the feasible solution space Q must be less than the work necessary to rule out the solutions in W - Q as non-feasible by other methods available.

The effectiveness of the Phase One Test at detecting the candidates in the W - Q space will be evaluated in the following pages.

False Negatives

In the preceding section, it was stated it was essential to guarantee that all the candidates in the W - Q space are singular. In this section, the four steps in the Phase One Test will be examined for such a requirement. Table IX summarizes the four steps taken in the Phase One Test.

TABLE IX

FOUR STEPS OF THE PHASE ONE TEST

Step	**Description**
One	Zero Vector Tests
Two	Zero/Non-Zero Element Test
Three	Count of Non-Zero Elements Test
Four	Equal Vectors Test

The first step in the test checks the vectors to determine if either vector is a zero vector. In the previous chapter, it was shown that if either vector was a zero vector, the two vectors are not linearly independent. The zero vector test provides a sufficient test to state that the corresponding candidate G matrix is singular. As a result, there can be no false negatives from this test.

The zero/non-zero element test checks whether an entry in one vector is zero while the corresponding entry in the other vector is non-zero. In the previous chapter, it was shown that if such a condition was spotted, then the vectors must be linearly independent. This test is sufficient to guarantee that the two vectors are linearly independent. It does not rule out any pairs of vectors as being dependent and, as such, will generate no false negatives.

The third test counts that number of non-zero element in the vectors and provides a sufficiency test to state that the vectors are linearly dependent. The previous chapter showed that if only two non-zero entries were in the vectors and that if they were at the same locations, the vectors were linear multiples of each other. This test cannot generate a false negative because if it detects the condition it is seeking, then that condition is sufficient to guarantee dependency.

The equal vectors test checks to see if the two vectors are exactly equal. If it detects this condition, then that is sufficient information to declare the vectors linearly dependent because they are multiples of each other. Since this test will succeed only when the vectors are equal, and that equality guarantees dependency, there can be no false negatives from the fourth step of the Phase One Test.

In summary, the four-step Phase One Test can generate no false negatives. It will never rule out a candidate matrix that is nonsingular. This characteristic was demonstrated by the simulation runs in Chapter III. Therefore, it can be stated that any candidate G matrix that fails the Phase One Test is not in the feasible solution space for the control problem. Such non-feasible candidates are in the non-feasible solution space W - Q and do not need to be considered in any optimality test.

Application of Phase One Test

From the previous chapter, it was demonstrated in numerous simulation tests that an application of the Phase One Test before a solution attempt would accurately rule out about forty-four percent of the possible candidate matrices.

$$\text{Ruled Out} \approx .44r^n \text{ candidates}$$

Consequently, before any optimality evaluation begins, nearly half of the solution space can be eliminated. It is guaranteed that none of the candidates that have been eliminated are actually feasible. In effect, an application of the Phase One Test can accurately and quickly identify most of the candidates in the W - Q space.

Since most cost functions will involve a multiply of the $\underline{x}$ vector, then at least n multiplies will be required per evaluation. As a result, using the

Phase One Test, we can avoid:

$$\text{Avoided multiplies} \approx .44nr^n$$

without performing any multiplies in the process.

It turns out that an application of the Phase One Test has greater impact on this problem than the problem of the previous chapter. In the previous chapter, either method might find a feasible solution on the first attempt and then quit. In this case, examination of all the possible solutions is required. The Phase One Test allows us to rule out many of the candidates that would otherwise need to be evaluated to assure optimality. Optimality tests can be applied against the remaining candidates to find the optimal feedback matrix.

If an applicable objective function is chosen for optimization, the Phase One Test can be used to reduce the number of candidates that require objective function evaluation. If the control problem is an optimal control problem that can be cast into a form suitable for pole-placement using Brogan's Method, then use of the Phase One Test reduces the amount of work necessary to find the optimal solution.

A More Intelligent Search

In Chapter II, the feedback problem was stated in equation (27) as:

$$\Delta'(\lambda) = \left| \lambda I_n - A - BF \right| = 0 \tag{80}$$

After some development, the problem was restated in equation (31) in the following form:

$$\Delta'(\lambda) = \Delta(\lambda) \left| \left[I_r - F\Phi(\lambda)B \right] \right| = 0 \tag{81}$$

In this equation, Brogan set columns inside the determinant equal to zero to force the expression to zero for given eigenvalues. In Brogan's writings on his procedure, he points out that other means could be used to force the determinant equal to zero, but that he would zero columns for convenience.

As an alternative and in general, consider the following equation where multiples of column vectors will be summed together to equal zero.

$$\left[I_r - F\Phi(\lambda)B \right] \underline{K} = \underline{0} \tag{82}$$

After some rearranging:

$$F\Phi(\lambda)B\underline{K} = \underline{K} \tag{83}$$

If this is done for each target eigenvalue, the resulting expression can be written as:

$$F\begin{bmatrix} \Phi(\lambda_1)BK_1 & \Phi(\lambda_2)BK_2 & \cdots & \Phi(\lambda_i)BK_i \end{bmatrix} = \begin{bmatrix} K_1 & K_2 & \cdots & K_i \end{bmatrix} \quad (84)$$

The problem of finding a feedback matrix F can now be stated as choosing K values and then solving for F. Notice that each K vector is a column vector with the number of rows equal to the number of columns in B. The number of K vectors is equal to the order of the system. In the rest of this chapter, the individual entries in the K vectors and F matrix will be referred to with lower case letters.

Stating the problem in this manner generalizes the work of Brogan. In his approach, he limits each K vector to all zeros except for one entry containing a one. This approach highlights the fact that there are potentially many more feedback matrices F that will result in the correct closed-loop response.

The following example problem should demonstrate the new freedom available with Equation (84). A second order system described below will be used.

$$A = \begin{bmatrix} 0 & 2 \\ 0 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (85)$$

The target eigenvalues for this system will be $\lambda_1 = -3$ and $\lambda_2 = -5$. First, calculate the associated $\Phi(\lambda)B$ terms then write the problem in the form of Equation (84).

$$F\left[\begin{bmatrix} -.3333 & .1111 \\ 0 & .1667 \end{bmatrix} K_1 \quad \begin{bmatrix} -.2000 & .0500 \\ 0 & .1250 \end{bmatrix} K_2\right] = \begin{bmatrix} K_1 & K_2 \end{bmatrix} \tag{86}$$

Now if Brogan's method were being used, we might choose:

$$K_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad K_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{87}$$

and then F is found as:

$$F\begin{bmatrix} .1111 & -.2000 \\ -.1667 & 0.000 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{88}$$

$$F = \begin{bmatrix} -5.00 & -3.33 \\ 0.00 & -6.00 \end{bmatrix} \tag{89}$$

If this feedback matrix is used, the resulting closed-loop system will have eigenvalues at the desired locations. From this example problem, it can be seen that if different values for the K vectors were chosen the resulting feedback matrix would be altered.

There is a catch to using this approach just as in Brogan's method. Namely, the $\Phi(\lambda_i)B$ matrix must be nonsingular. There is not one hundred percent freedom in choosing the K values.

The problem formulation in (84) is similar to a form found in Fahmy and O'Reilly[19].

Choosing a Best Feedback Matrix

If it is necessary in a given problem to find a 'best' solution, then clearly one would prefer to choose from as many alternatives as possible to find a 'best' solution. The generalization of the previous section was done to give us additional freedom and choices in choosing the feedback matrix F.

For the sake of this research, let the control goal be to minimize the control effort required to achieve the closed-loop feedback requirements. Namely, let us define an objective function to minimize the control signal $\underline{u}$ defined in Equation (3).

$$\underline{u} = F\underline{x} + \underline{w} \tag{90}$$

Since at this point, we have no control over the input signal $\underline{w}$ it will be ignored. Furthermore let us consider the remaining term in a minimum energy fashion. Therefore, let us modify the objective function as follows:

$$\text{Minimize} \quad (F\underset{\sim}{x})^t(F\underset{\sim}{x})$$

or

$$\text{Minimize} \quad \underset{\sim}{x}^t F^t F\underset{\sim}{x}$$

Now with one more observation the final objective function will be stated. Notice that if F was a vector, then the inner two terms would be an inner product. With that spirit in mind, let us define our final objective function as:

$$Minimize \sum_{\substack{1 \leq i \leq r \\ 1 \leq j \leq n}} f_{ij}^2 \tag{91}$$

In short, the objective to be achieved is to minimize the square of the individual terms in the feedback matrix F. The difficult part is choosing the k terms that result in an F matrix with minimal characteristics. If the example problem considered earlier is solved in terms of k the resulting equations are:

$$f_{11}(-.3333k_1 + .1111k_2) - .1667k_2 f_{12} = k_1 \tag{92}$$

$$f_{21}(-.3333k_1 + .1111k_2) - .1667k_2 f_{22} = k_2 \tag{93}$$

$$f_{11}(-.2k_3 + .02k_4) - .125k_4 f_{12} = k_3 \tag{94}$$

$$f_{21}(-.2k_3 + .02k_4) - .125k_4 f_{22} = k_4 \tag{95}$$

As can clearly be seen, the resulting four equations for the terms in the F matrix are nonlinear functions of the k terms. This nonlinearity complicates the problem of finding k terms that result in minimal f terms.

Upon further examination of this problem, it can be seen that we have a sum of squares objective function to minimize, where the terms to minimize (f) are non-linear functions of other terms (k). This type of problem is referred to as a non-linear least squares problem, and there are several numerical techniques available for solving these problems[20]. For the problem here, the method of Levenberg-Marquardt will be used. Without going into the details of this algorithm, it can be stated that the algorithm alternates between a steepest descent approach and a Gauss-Newton approach depending on run-time conditions. Furthermore, the algorithm is guaranteed to converge since the steepest descent part of the algorithm will always move in a minimizing direction until it can't find a lower point.

If the objective function is evaluated for the first example problem discussed earlier, then the cost is found to be:

Unoptimized Cost = 72.1

However, if we apply the Levenberg-Marquardt algorithm to the problem and let it find the K vectors and the resulting F matrix the solution is:

$$K = \begin{bmatrix} .8226 & 1.229 \\ 1.053 & .265 \end{bmatrix} \qquad F = \begin{bmatrix} -5.29 & 0.53 \\ -.327 & -5.71 \end{bmatrix}$$

Optimized Cost = 60.7

Application of this algorithm finds a solution that results in a fifteen percent reduction in the value of the objective function. In some applications, such a reduction could be significant.

An important point to realize is that the reduced cost solution was found away from the Brogan's method solution. If constrained to using only Brogan's method, such a cost reduction would not have been possible. Of the four possible Brogan's method solutions for this problem, the best results are shown in the example problem above. If one had been unfortunate enough to choose the following K vector:

$$K = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \quad F = \begin{bmatrix} 0 & 0 \\ -7.5 & -11. \end{bmatrix}$$

the cost without optimization would be 177.25! Application of the Levenberg-Marquardt method would result in a sixty-six percent improvement.

Comparison to Brogan's Method

The freedom to choose K vectors without Brogan's restriction can result in better solutions in some cases. The problem in the previous section is an example. At this point, let us consider three additional problems for a total of four problem to see what effect this new approach has on reducing the feedback cost. For each problem, we will evaluate the objective function described above and a new function which gives more of a physical interpretation of feedback cost.

The new cost function evaluated will be to sum the absolute value of each row vector in the feedback matrix F. Such a sum gives more of a physical interpretation since it represents gains instead of squared gains. This second merit function will not be used to select the F matrix, but only after one is chosen will it be evaluated. The function defined precisely is:

$$R = \sum_{\substack{1 \leq i \leq r \\ 1 \leq j \leq n}} \left| f_{ij} \right| \tag{96}$$

The first problem considered in detail is the example problem described earlier in this chapter. The second problem is found in Fahmy and O'Reilly[19] and is described as:

$$A = \begin{bmatrix} 0 & 1 & 2 \\ -2 & 3 & 0 \\ -2 & -1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \quad K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{97}$$

Target Eigenvalues: -1, -1, -2

The third problem for consideration is found in Brogan[16] in chapters 12 and 16. This model describes the lateral flight dynamics of an aircraft:

$$A = \begin{bmatrix} 10 & 0 & -10 & 0 \\ 0 & -.7 & 9 & 0 \\ 0 & -1 & -.7 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 20 & 2.8 \\ 0 & -3.13 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad K = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{98}$$

Target Eigenvalues: -2, -5, -8 and -10

Finally, a fourth problem will be taken from Brogan[16]. It is described as:

$$A = \begin{bmatrix} -2 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \quad K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{99}$$

Target Eigenvalues: -2, -3, -4

For each of these four systems, the two cost functions are evaluated under initial conditions and after the optimization algorithm is applied. The results are shown in the following table.

TABLE X

TEST CASE RESULTS

Problem	**Brogan** Cost A	**Brogan** Cost R	**Optimized** Cost A	**Optimized** Cost R
1	72.11	14.33	60.69	11.86
2	26.79	10	26.76	9.92
3	66.11	12.16	13.87	6.26
4	49	11	49	11

As can be seen from these four example problems, when the control problem is stated in this manner and the Levenberg-Marquardt algorithm is applied, the results can be significantly better than a simple application of Brogan's method. In the third problem, the least-squares algorithm resulted in a eighty percent reduction in the first objective function and fourth-nine percent reduction in the second objective function.

Since the Levenberg-Marquardt algorithm only moves in directions of lower cost, it will never leave a point if it is already at a minimum. Therefore, use of this technique will never result in worse performance than the initial guess. In the case of the fourth problem in the table, the initial guess was the 'best' solution.

Effects of Changing the F Matrix

Throughout this research, the primary interest has been in maintaining pole/eigenvalue locations after actuator failure. In Chapter III and this chapter, it has been assumed that any feedback matrix that results in the target eigenvalues was acceptable. However, it should be pointed out that the complete system response will be altered as different feedback matrices F are used. An important question to ask is whether or not there are any negative side effects to using the optimization scheme described in this chapter.

In the work of Brogan[16] and in the work of Fahmy and O'Reilly[19] the reason for the change is discussed. Both sources prove through different means that the eigenvectors are being changed as the K vectors are changed and as a result as F changes. Since the complete response of a system is a function of both the eigenvalues which do not change as F changes and the eigenvectors which do change the net effect is a change in complete system response.

Figures 4. and 5. provide some insight into the changes that occur when the F matrix is altered. Figure 4. is a plot of the $x(0)$ state variable in the first example problem in response to $[\ 1\ 0\ ]^t$ initial conditions. One line describes the response in the worst case Brogan selection described earlier while the other line describes the response to the least-squares selection. The response is clearly different when the two different F matrices are used in the closed-loop system.

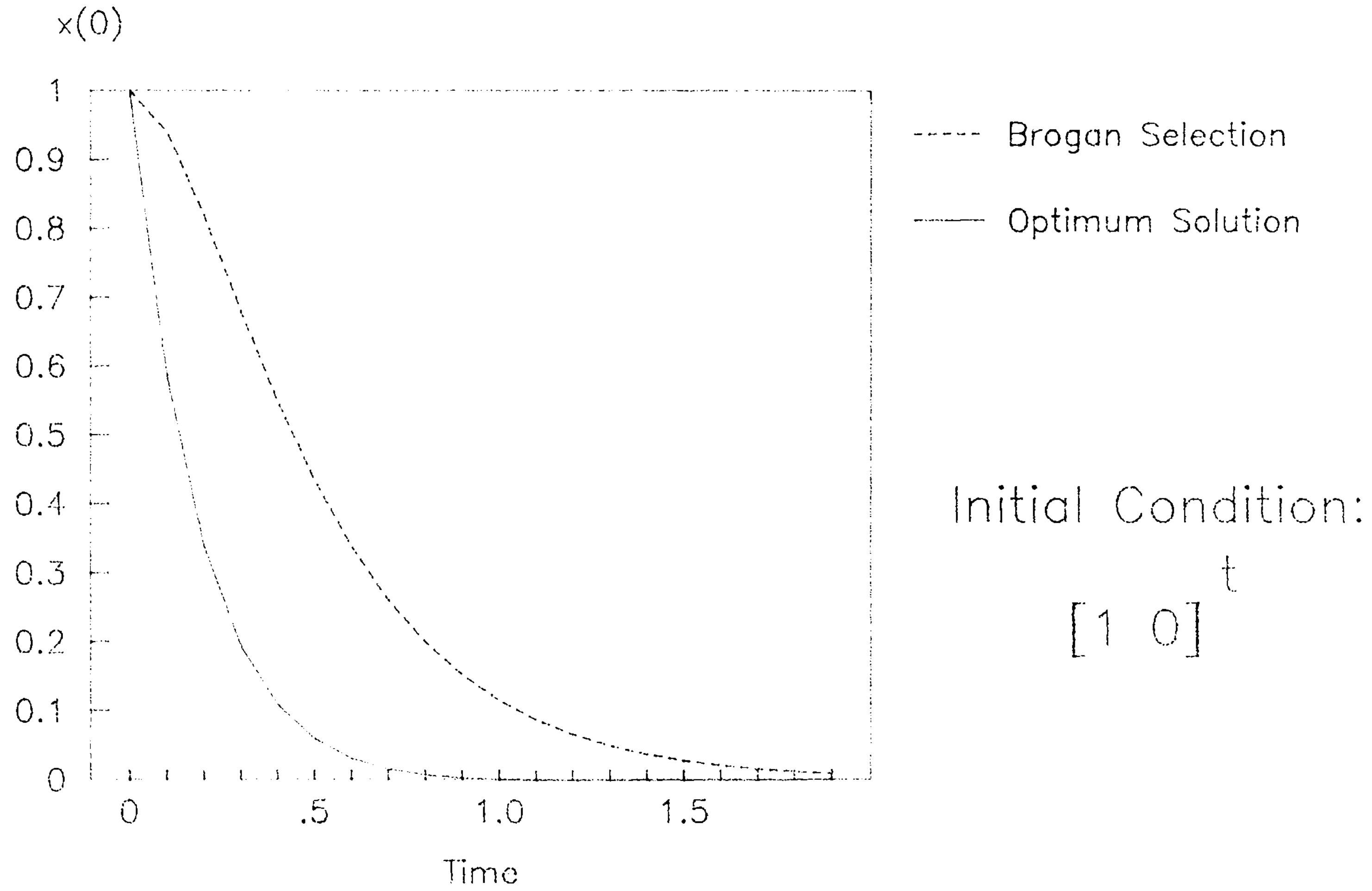

Figure 4. X(0) vs. Time

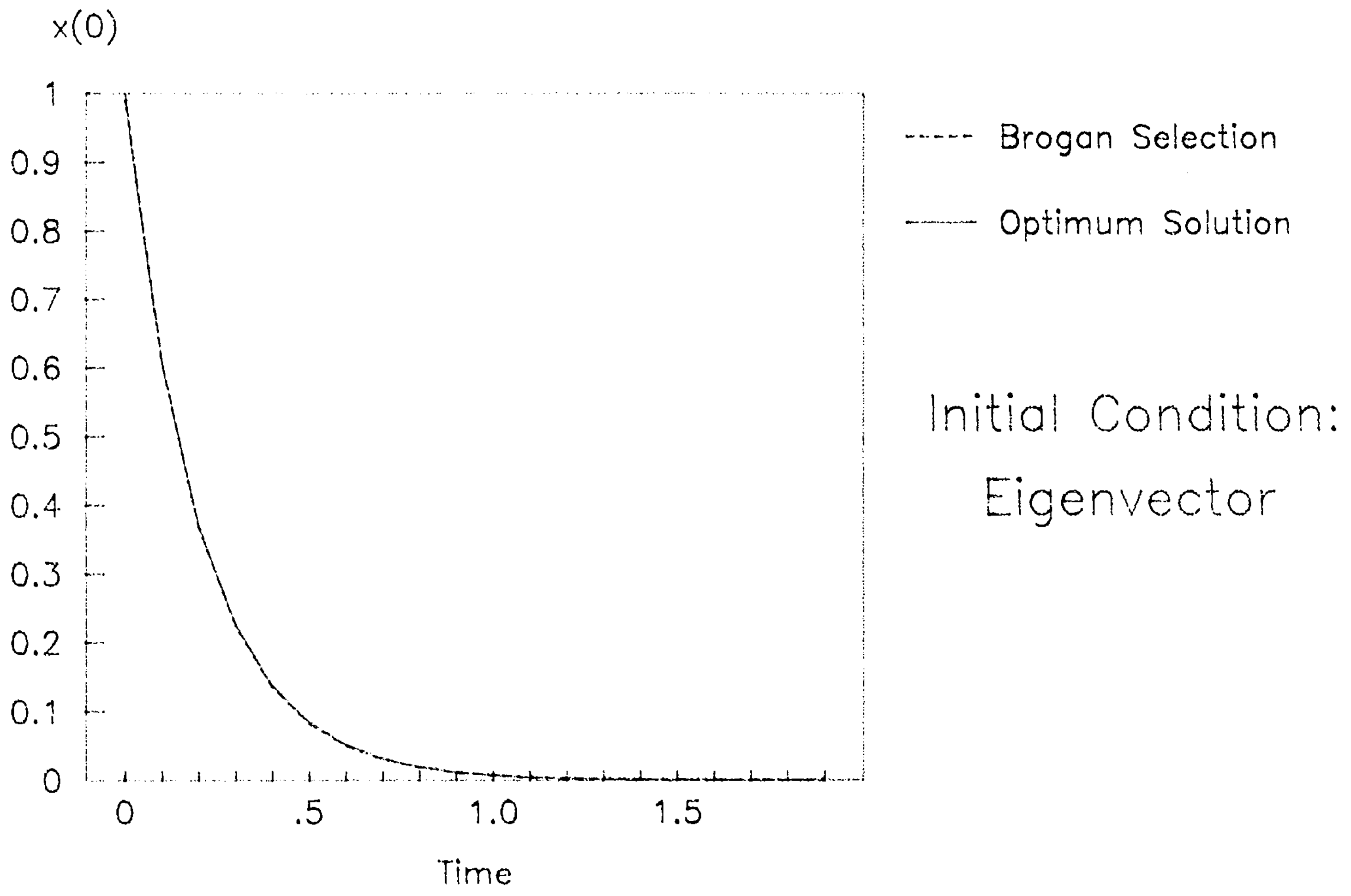

Figure 5. X(0) vs. Time

However, if the initial conditions are changed to point in the direction of one of the eigenvectors of the closed-loop system, then the response offers insight into the change that occurs when F is altered. Figure 5. reveals that when the initial conditions are chosen to correspond to the closed-loop eigenvectors, the response is identical. The eigenvector used in Figure 5. corresponds to the -5 eigenvalue in each case.

The difference between Figure 4. and Figure 5. suggests that the initial conditions highlight different modes of the closed-loop system. An adequate comparison requires that a common ground be established to evaluate the response changes due to the changes in F. Once the comparison is made with fair initial conditions, the response is seen to be identical.

Conclusions

The two methods outlined in this chapter provide dramatically different approaches to finding a 'best' feedback solution. The first method using the Phase One Test provides a simplistic method for finding a suitable feedback matrix F. The advantages of this approach are that it is conceptually simple and easy to implement on a computer. The primary disadvantage to this approach is that a full coverage search strategy of this type can require enormous amounts of time when the order of the system is high.

The non-linear least squares search strategy offers two significant advantages. First, the search strategy to find a solution is much more intelligent than a linear search using the Phase One Test. The Gauss-

Newton characteristics of the algorithm can allow it in some cases to jump close to a solution in a few steps. The second advantage of this approach is that since the restrictions of Brogan's method are removed, better solutions can generally be found. The four example problems clearly demonstrated that significant performance improvements can be obtained.

Unfortunately, the non-linear least squares search strategy has some disadvantages too. First, it requires an expensive to implement computer algorithm to run in real-time. The algorithm requires function derivatives to be determined at each point the algorithm moves. A second problem is that in some cases the solution convergence is slow. Recall that an ordinary Newton's method algorithm converges slowly when multiple zeros are present. The same characteristic can appear in the Levenberg-Marquardt method.

Overall, the Phase One Test search and the non-linear least squares search provide two diverse alternatives to the problem of finding a 'best' feedback solution to a given control problem.

CHAPTER V

COMPUTER IMPLEMENTATION DETAILS

Introduction

This chapter discusses numerous computer implementation details necessary to make the algorithms of the previous two chapters work. Specific computer details on calculating the Z and X matrices will be presented as well as details on implementing the Phase One Test. Example code will be used where appropriate.

In addition to computer details, this chapter will present a flowchart outlining use of the Phase One Test in solving the control problem of Chapter III.

Calculating the Z Matrix

The principle difficulty of implementing Brogan's Algorithm on paper or on a computer is the apparent necessity to do a symbolic matrix inversion. The calculation of $\Phi(\lambda)$ appears to require an off-line symbolic inversion. Fortunately, this problem can be avoided.

Let us calculate a separate $\Phi(\lambda)$ for each λ_i in our problem. Then concatenate the various $\Phi(\lambda)$ matrices together end-to-end to form the Z matrix.

If the problem is solved in this fashion, the Z matrix can be evaluated on the computer using numerical routines given the A matrix and the target eigenvalues λ_i. The psuedocode of Figure 6 demonstrates the solution.

```
for i = 1 to number_of_eigenvalues
   {
   copy(A,ATMP)

   for j = 1 to DIM(A)
      ATMP[j][j] -= pole[i];

   for j = 1 to DIM(A)
      {
      for k = 1 to DIM(A)
         ATMP[j][k] *= -1.0;
      }
   invert(ATMP,tmp);

   for j = 1 to DIM(A)
      {
      for k = 1 to DIM(A)
         Z[j][k+DIM(A)*i] = tmp[j][k];
      }
   }
```

Figure 6. Psuedocode For Calculation of Z

With the use of psuedocode in Figure 6, it becomes possible to randomly create A, B, and target eigenvalues for simulation purposes and not have to symbolically invert A each time. This outlined approach is implemented in a C function called prep() in the program included in the appendix.

Calculating the X Matrix

The calculation of the X matrix is achieved by multiplying each partition of the Z matrix times the B matrix. The resulting matrices are concatenated together to form the X matrix.

When the Phase One Test is implemented in a real environment where a fault detection system is updating actuator performance data and availability, the calculation of the X matrix can be optimized for cases when actuator performance is varying. In such cases, the X calculating procedure should only update the columns in X that are affected by changes in B. There is no need to recalculate the entire X matrix for every change in the B matrix.

Phase One Test

The most important part of the code in this work is the Phase One Test. The calculation of the Z matrix discussed above is not as crucial because it is executed one time, outside the loop. The calculation of the X matrix involves limited optimization opportunities since it involves a straight matrix multiplication; however, the Phase One Test will be executed inside the loop for each candidate G matrix.

Figure 7 outlines the first step in the two vector tests discussed in Chapter III. The first vector is stored in column i^{th} column of matrix d. The second vector is stored in the j^{th} column of matrix x.

```
TOL=1e-6;
stat=FALSE;
for m = 1 to DIM(A)
   {
   if(x[m][j] > TOL or x[m][j] < -TOL)
      {
      stat=TRUE;           /*not a zero vector*/
      break;
      }
   }
if(stat == FALSE)
   return(-1);                   /*zero vector*/

stat=FALSE;
for m = 1 to DIM(A)
   {
   if(d[m][i] > TOL or d[m][i] < -TOL)
      {
      stat=TRUE;           /*not a zero vector*/
      break;
      }
   }
if(stat == FALSE)
   return(-1);                   /*zero vector*/
```

Figure 7. Step One: Zero Vector Test

Notice from Figure 7 that the code completely checks one vector before checking the second vector. In this manner, if the first vector is a zero vector, the second vector will not even be examined. If both were tested in a single loop, then twice the work is performed if either vector is a zero vector.

Figure 8 presents steps two and three in our testing procedure. This part of the procedure is the most complicated. Notice the variable 'count' which is incremented every time a row in each column has non-zero entries. This count variable will be tested for the step three test.

```
TOL=1e-6
count = 0;
for m = 1 to DIM(A)
   {
   if(d[m][i] < TOL and d[m][i] > -TOL)
      {
      if(x[m][j] > TOL or x[m][j] < -TOL)
           return(0);        /*independent*/
      else
           continue;         /*0/0 case*/
      }
   else
      {
      if(x[m][j] < TOL and x[m][j] > -TOL)
         return(0);        /*independent*/
      else
         {
         count++;        /*non-0/non-0 case*/
         continue;
         }
      }
   }
if(count <= 1)
   return(-1);                   /*dependent*/
```

Figure 8. Steps One and Two

Figure 9 outlines the fourth and final step used to compare two vectors. The equal vectors test subtracts one vector entry from the other and then compares the difference to zero.

```
for m = 1 to DIM(A)
   {
   sum = d[m][i] - x[m][j];
   if(sum < -TOL or sum > TOL)
      {
      return(0);          /*independent*/
      }
   }
return(-1);               /*dependent*/
```

Figure 9. Step Four: Equal Vectors Test

In this final step, notice that the routine will examine individual rows until it finds a row where the two values are not equal. At that point, the routine returns a passing value. If the routine proceeds through all the rows without finding a row with differing values, the vectors are exactly equal and thus dependent.

Three appendices are attached that include the complete Phase One Test implemented in code, the main body of a simulation program, and a series of matrix utility functions used throughout the project. The simulations of this research were done in C.

Using the Phase One Test For Fault Adaption

In Chapter III, a simple algorithm to estimate the singularity of a given matrix was developed. In this section, the test will be used in the original fault-adaptive problem.

Consider Figure 10, on the next page, which summarizes an adaption approach for the problem of finding any feasible solution in minimum time. The approach begins with a process that detects actuator faults or changes. Throughout this paper, the detection of failures in systems has not been addressed because there are several good techniques available for the fault detection problem. However, to implement an adapting algorithm requires a fault detection/identification process in the loop. The detection process must quickly determine that a fault has occurred. The identification process must specify where the fault has occurred and the new actuator characteristics after the fault.

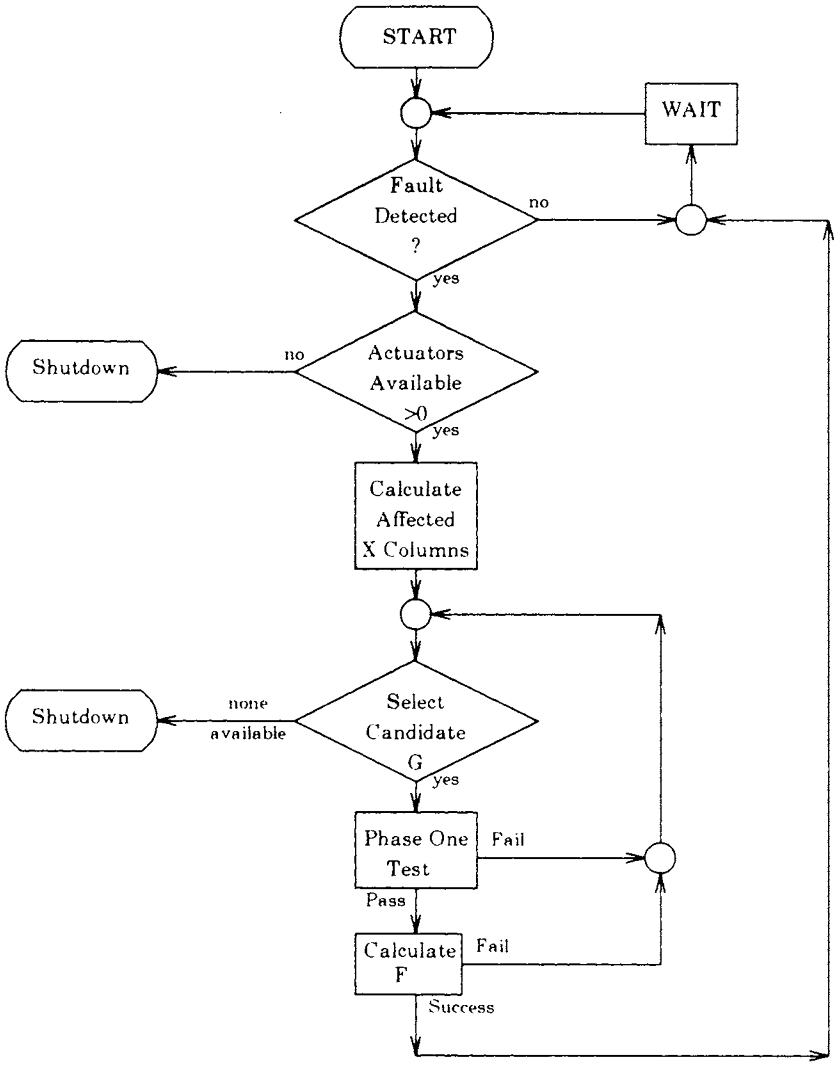

Figure 10. Fault Adapting Procedure

Immediately after a fault has been detected and the new actuator performance has been determined, the adaption process begins. The next step in the process is to determine whether or not there are actuators available after the fault. If none are available, the system is clearly not controllable and a shutdown process, if one is available, must begin. If actuators are available, the adapting process must first recalculate the affected columns in X. This step involves a straightforward matrix multiplication as outlined earlier.

Once a new X matrix has been determined, a selection process should select a candidate G matrix for analysis. At this point, the search strategy being implemented plays a role.

Once a candidate matrix has been chosen the Phase One Test is applied against it. If the candidate fails the test, then another candidate is selected. If no candidates pass the test, a shutdown procedure must begin because there is no feasible way to maintain the chosen system performance.

When a candidate passes the Phase One Test the value of F is calculated. If F exists, the process returns to the fault detection loop. From the simulation runs presented earlier, this case will occur over ninety-seven percent of the time. The remaining cases will require a new candidate matrix to be selected.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

Conclusions

The goal of this research has been to develop a new adaptive pole-placement algorithm specifically designed to adapt to controller changes. Requirements on the new algorithm were that it needed to maintain pole positions after actuator failure and that it needed to be well-suited for a fast computer implementation. These goals were driven by the apparent lack of work in this direction and by the stated need for such work in the April 1987 issue of Transactions on Automatic Controls[12].

In summary, while several specific contributions are outlined in the following paragraphs, it can be stated that the method developed in this work meets the stated research goals. A new approach to adapting to actuator faults is presented and shown to be efficient when implemented on a computer. This new approach takes a well known pole-placement method and extends it to solve the stated adaption problem. A simple four-step test is created and used to greatly reduce the amount of work that would otherwise be necessary in the evaluation of a feedback mechanism which compensates for the failure. In short, this research has put together a new tool for use in the fault-adaptive control problem.

This research has resulted in several significant contributions to the fault-adaptive control problem. The first contribution concerns the means to model actuator faults as changes in the B matrix. This means of modeling the faults allows the adapting problem to be considered using traditional control methods. The importance of this simple observation should not be overlooked in that it was essential for the adapting algorithm of this research to be developed.

The second major contribution from this work is the adapting algorithm itself. A large amount of time was spent examining many different control approachs before Brogan's algorithm was found to possess several key characteristics. The ability to push the costly matrix inversion step off-line ahead of the adaption loop makes Brogan's approach ideal for the fault-adaptive problem specified. Furthermore, the characteristic of any nonsingular G matrix resulting in a feasible feedback matrix F allows the adapting algorithm to have a very clear goal of finding a nonsingular matrix. A feasible solution is guaranteed if a nonsingular G matrix is found. An additional plus is the advantage of not requiring any new matrix multiplications or inversions if an actuator fails and is removed. These characteristics of Brogan's algorithm make it ideal for the fault-adaptive problem being addressed.

Once Brogan's algorithm was identified, it became necessary to cast the algorithm into a form suited for computer implementation. Such a form was developed and is now used in the resulting algorithm.

The task of finding a nonsingular candidate G matrix then became the goal. Simple search strategies and numerous condition number techniques were tried, but they either resulted in poor performance or required too much work to arrive at an answer.

Finally, the fundamental principle of matrix rank coupled with a problem formation characteristic in Brogan's method yielded a simple four-step approach to determining the feasibility of a candidate matrix. This four-step approach called the Phase One Test determines with approximately ninety-seven percent probability whether or not a candidate G matrix will result in a feasible solution. Furthermore, the Phase One Test results in no false negatives and requires no expensive multiplies or divides.

An important additional characteristic of the algorithm is that it is extremely well suited for implementation in a multiprocessor environment. The problem can be broken into distinct tasks that can be executed in parallel. Such a characteristic is ideal in a computer control environment that has multiple processors available for use.

Once a method was in place for finding a feasible solution, two seperate strategies where developed for finding a 'best' solution. One strategy used the Phase One Test to reduce the problem solution space by almost half. A linear search would then be used to determine the best solution of the remaining feasible candidates. A second strategy expanded on Brogan's development to cast the feedback problem in a more generalized form. A non-linear least squares algorithm was then shown to find a 'best' solution out of a much larger solution space than the first approach. The solution found by the second method will generally result in a better solution than the one found by the restricted first method.

Future Work

In this section, some extensions to the approach outlined in the previous chapters will be considered. Some of the techniques will speed up the Phase One Test while other techniques will attempt to reduce the number of unnecessary inversions still further. Some of the approaches will be minor ideas while others might be of significant use for particular classes of problems.

This section will also include discussion on some important points that were not covered earlier.

Regional Pole-Placement

The adaption method developed in this research deals with the problem of holding pole positions in required locations after actuator changes or failures. An interesting problem to be considered would be to relax the requirement that the poles remain fixed and let the poles move inside a specified region. For example, the design requirement could be relaxed to specify the pole positions inside an ellipse or left of a parabola. Such a specification would provide additional freedom in choosing a feedback mechanism. A challenging problem would be to allow pole movement while still finding a feedback matrix F.

Parallel Implementations

As mentioned earlier, the method developed in this research is well-suited for a multiprocessor environment. An important follow-up step to this research would be to implement the approach in a multiprocessor environment. A successful system would involve a fault detection/identification process coupled with the adaptive process. A complete implementation could explore the coupling issues between the detection phase and the adaptive phase. Furthermore, system shutdown questions could also be answered. Overall, a full parallel processing implementation could provide a complete start-to-end adaptive problem example.

Repositioning the Zero Vector Test

The first test in our two vector test was to detect the presence of zero vectors in the candidate G matrix. As the algorithm was implemented, that test was done at the vector test level and not one time on the entire matrix. If the zero vector tests were pulled out of the two vector test section and placed up front, then less work would be necessary to detect zero vectors.

In the current implementation, tests for zero vectors are repeated against the same vectors more than once. Unnecessary tests are performed when zero vectors exist but are positioned to be examined later.

Eliminating Redundant Vector Tests

In the Phase One Test, pairs of vectors are tested to determine if they are linearly dependent. If they fail the test, then another pair of vectors are tried until a pair fails the test or all the vectors pass the test and inversion begins.

However, if the test fails and another combination of vectors are chosen to make up the candidate G matrix, the Phase One Test is reapplied to the new G matrix. As before, testing pairs of vectors for dependency will begin, in which case a previously tested pair might be tested *again*. However, it is clear that if the two vectors passed or failed the Phase One Test the first time, the result will be unchanged during a second test. Therefore, the Phase One Test could be modified as:

1. As each pair of vectors is tested, store the result.
2. If the same pair of vectors must be tested again, recall the result from the first test.
3. If a change in a B matrix entry changes any vector that has a stored result, then clear the test results for the affected pairs.

This addition to the Phase One Test does not change the algorithm but merely its implementation. However, more storage and additional logic will be required.

The modification described in this section will not result in a significant performance improvement since no multiplies are saved; however, the number of additions/subtractions and indexing operations will be reduced. This improvement is definitely appealing since redundant work is being eliminated.

Tridiagonal Systems

In our approach, a nonsingular matrix is being *built*. Since there is freedom to choose vectors from each partition of our described X matrix, it may be possible to choose vectors that yield a G matrix structure that is easy to solve numerically. One such structure is the tridiagonal structure where a matrix has non-zero elements on the main diagonal and then above the diagonal or below the diagonal, but not both.

If the vectors chosen from X resulted in a tridiagonal G matrix, then a special algorithm, such as the one described in Press[21]. could be used instead of the traditional, general purpose inversion/solution algorithm generally used.

This approach is mentioned because it might be useful for some classes of problems. The necessary logic to build a tridiagonal matrix from X could be easily implemented.

Redundant Actuators

In the simulation section of Chapter III the issue of redundant actuators appeared. At that time, redundancy was not considered in the simulation process. The case of redundant actuators should be considered as a special case in the fault-adaption problem. If it is known that a redundant means of control is available, that knowledge should be included in our controller/adapter design. An adaption algorithm should not be required to determine that the redundant actuator should replace a failed actuator without changing the feedback gains. If redundant actuators are available, they should replace the failed actuator automatically when a failure is detected. It is for this reason that redundant actuators have not been discussed in this paper.

BIBLIOGRAPHY

1. W. L. Brogan, "Applications of a Determinant Identity to Pole-Placement and Observer Problems," IEEE Trans. on Automatic Control, pp. 612-614, October 1974.
2. D. Graupe, "Identification of Systems," Robert F. Krieger, New York, 1972, pp. 20-23.
3. D. G. Luenberger, "Introduction to Dynamic Systems," John Wiley & Sons, New York, 1979, pp. 276-285.
4. P. K. Sinha, "Multivariable Control: An Introduction," Marcel Dekker Inc., New York, 1984, pp. 159-164.
5. W. M. Wonham, "On Pole Assignment in Multi-Input Controllable Linear Systems," IEEE Trans. on Automatic Control, Vol. AC-12, No. 6, pp. 60-665, December 1967.
6. S. J. Raza and J. T. Silverthorn, "Use of the Pseudo - Inverse for Design of a Reconfigurable Flight Control System," American Inst. of Aeronautics and Astronautics, pp. 349-356, 1985.
7. A. Alos, "Stabilization of a Class of Plants with Possible Loss of Outputs or Actuator Failures," IEEE Trans. on Automatic Control, pp. 231-233, February 1983.
8. J. B. Pearson and P. W. Staats, Jr., "Robust Controllers for Linear Regulators," IEEE Trans. on Automatic Control, pp. 231-234, June 1974.
9. Juergen Ackermann, "Parameter Space Design of Robust Control Systems," IEEE Trans. on Automatic Control, pp. 1058-1072, December 1980.
10. H. Elliott and W. A. Wolovich, "A Parameter Adaptive Control Structure for Linear Multivariable Systems," IEEE Trans. on Automatic Control, pp. 340-352, April 1982.
11. H. Elliott, W. A. Wolovich, and Das, "Arbitrary Adaptive Pole Placement for Linear Multivariable Systems," IEEE Trans. on Automatic Control, pp. 221-228, March 1984.
12. "Challenges to Control: A Collective View," IEEE Trans. on Automatic Control, pp. 275-285, April 1987.
13. B. Friedland, "Control System Design," McGraw-Hill Inc., New York, 1986, pp. 224-236.
14. M. Pachter, "An Explicit Pole-Assigning Feedback Formula," IEEE Trans. on Automatic Control, pp. 263-265, April 1977.

15. J. O'Reilly, "Comments on 'An Explicit Pole-Assigning Feedback Formula,'" IEEE Trans. on Automatic Control, pp. 1012-1013, October 1980.

16. W. Brogan, "Modern Control Theory," Prentice-Hall Inc., New Jersey, 1985, pp. 401-403.

17. Samual Conte and Carl de Boor, "Elementary Numerical Analysis: An Algorithmic Approach," McGraw-Hill Book Company, New York, 1980, Third Edition, pp. 175-177.

18. J. King, "Introduction to Numerical Computation," McGraw-Hill Book Company, New York, 1984, pp. 122-128.

19. M. M. Fahmy and J. O'Reilly, "On Eigenstructure Assignment in Linear Multivariable Systems," IEEE Trans. on Automatic Control, pp. 690-693, June 1982.

20. L. E. Scales, "Introduction to Non-Linear Optimization," Springer-Verlag New York Inc., New York, 1985.

21. W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, "Numerical Recipes: The Art of Scientific Computing," Cambridge University Press, Cambridge, 1986, pp. 40-41.

VITA

Mark A. Brewer

Candidate for the Degree of

Doctor of Philosophy

Thesis: A FAULT-ADAPTIVE POLE-PLACEMENT ALGORITHM SUITED FOR COMPUTER IMPLEMENTATION

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Oklahoma City, Oklahoma, March 2, 1961, the son of Richard H. and Ellen F. Brewer.

Education: Graduated from Putnam City High School, Oklahoma City, Oklahoma, in May, 1979; received Bachelor of Science Degree in Electrical Engineering from Oklahoma State University at Stillwater in May, 1983; received Master of Science Degree from Oklahoma State University; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in July, 1988.

Professional Experience: Teaching Assistant, Department of Electrical Engineering, Oklahoma State University, January 1983 to December 1984; Summer Research Assistant, AT&T Bell Laboratories, Murray Hill, New Jersey, Summer 1983; Development Engineer, AT&T Technologies, Inc., Oklahoma City, Oklahoma, Summer 1984 to Present.