12-1994

A Gain Scheduling Optimization Method Using Genetic Algorithms

Robert C. Martin IV

A Gain Scheduling Optimization Method

Using Genetic Algorithms

THESIS

Robert C. Martin, IV
Second Lieutenent

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

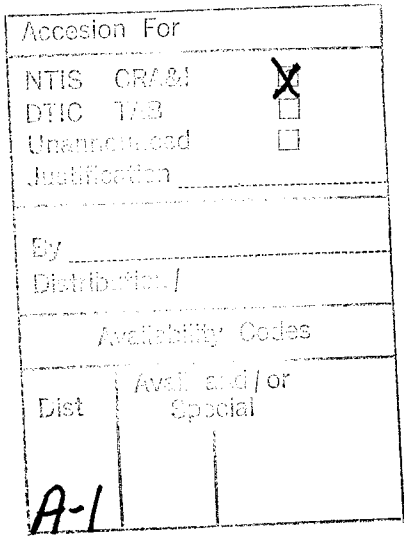
AFIT/GAE/ENY/94D-3

A Gain Scheduling Optimization Method Using Genetic Algorithms

THESIS
Robert C. Martin, IV
Second Lieutenent

AFIT/GAE/ENY/94D-3

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.



AFIT/GAE/ENY/94D-3

A Gain Scheduling Optimization Method Using Genetic Algorithms

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Robert C. Martin, IV, B.S.

Second Lieutenent

December 1994

Acknowledgements

I would like thank my advisor LtCol Stuart Kramer. He accepted me as a thesis student even though he was unfamiliar with genetic algorithms. I am very greatful for the support and advice from my committee, Dr Brett Ridgely and Dr Gary Lamont. I would also like to thank Jim Buffington and Lt Bill Reigelsperger for their advise and assistance in acquiring nonlinear flight simulation data.

I would like to say thank you to my parents who made me possible. Thank you very much for your love and support throughout everything that I have undertaken.

Finally, I would like to dedicate this thesis to Dr Jane Uva, M.D., M.P.H. She has been very supportive and understanding throughout the past year.

Robert C. Martin, IV

Table of Contents

List of Figures

List of Tables

List of Symbols

List of Abbreviations

Abstract

Gain scheduling, the traditional method of providing adaptive control to a nonlinear system, has long been an ad hoc design process. Until recently, little theoretical guidance directed this practioners' art. For this reason, a systematic study of this design process and its potential for optimization has never been accomplished. Additionally, the nonlinearities and the large search space involved in gain scheduling also precluded such an optimization study. Traditionally, the gain scheduling process has been some variation of a linear interpolation between discrete design points. By using powerful non-traditional optimization tools such as genetic algorithms there are ways of improving this design process.

This thesis utilizes the power of genetic algorithms to optimally design a gain schedule. First, a design methodology is validated on a simple pole placement problem, then demonstrated for an F-18 Super-maneuverable Fighter. From this experience, a general gain scheduling design process is developed and presented.

A Gain Scheduling Optimization Method Using Genetic Algorithms

I. Introduction

1.1 Background

Control design and optimization is based upon many assumptions that simplify a real world control problem to a mathematically manageable model. After designing a controller for a simplified model, it is implemented on the real world problem. Consequently, because of the simplifications in model reduction, there are difficulties in implementing the controller and commanding a desired response throughout the entire operating envelope of the system. One method of overcoming these difficulties is gain scheduling. Gain scheduling is a method of changing the controller depending upon the operating condition. There are four basic steps in designing a gain schedule [30]:

1. Select a family of constant operating point plants,
2. Design a controller for each plant in family,
3. Design a method of scheduling the controllers such that performance is maintained at each design point, and
4. Check non-local performance of the scheduled controller by simulation.

Typically, gain scheduling is some form of linear interpolation between the controllers designed at discrete operating points. There are two major concerns in gain scheduling. The primary concern is that the scheduled controller maintains stability throughout the operating envelope [43, 46, 45, 44]. The secondary concern is that acceptable performance is also maintained throughout the operating envelope. In designing a gain schedule, there are many design variables the control designer must choose [41]:

- The variable to schedule the controller,

- The number of members in the family of plants,
- The members in the family of constant operating point plants,
- The method of scheduling the controller,
- The number of discrete points within the scheduling variable range for linear interpolation, and
- The distance between each chosen discrete point in the scheduling variable range.

In numerically optimizing a gain schedule design, only the last two of these design variables are alterables. Unfortunately, traditional calculus-based optimization methods become numerically inaccurate and fail to find an optimal solution when trying to simultaneously optimize these variables. One method of overcoming this difficulty is to set one variable and optimize the other, however, this procedure is time consuming and inefficient when there are means of optimizing these variables simultaneously. Another significant difficulty is that the number of the design variables is a function of one of the variables. For instance, if three intervals of the scheduling variable are selected for optimization, then there are not as many variables to optimize as if eight intervals are selected. Additionally, the nonlinearities of the system and the high dimensionality of the problem makes this problem very difficult for traditional calculus-based deterministic search algorithms [27, 32, 35]. Since, genetic algorithms can overcome these difficulties they are used to simultaneously optimize these variables [27, 32, 35].

Genetic algorithms (GAs) have found global optima in problems with both nonlinearities and high dimensionality where calculus-based methods have fallen into local optimum, such as aircraft structural parameter design [9] and various control optimizations [27, 32, 35, 38, 39]. For these reasons, GAs are used as an optimization tool in this investigation.

1.2 Purpose

The purpose of this research effort is to:

- Demonstrate that gain scheduled controllers can be optimized using GAs,
- Illustrate the feasibility gain scheduling optimization process on a full envelope aircraft flight control example, and
- Present a general gain schedule design method.

1.3 Scope of Research

The first basic step in gain scheduling, selection of operating points, is a procedure that relies on common sense and engineering judgement, consequentially, it is not reviewed in this research effort. Similarly, the second basic step, controller design, is independent of the gain scheduling design process, consequentially, it also is not considered in this research work [30]. Conversely, the third step, controller scheduling, can be optimized with respect to a specified objective. Therefore, this research effort is limited to optimizing this procedure. Finally, the fourth basic step, non-local performance validation, merely verified the global performance of the designed controller. Additionally, this research does not focus on improvements in the computation efficiency of the GA optimization algorithm or the objective function calculation. This effort is concerned with the optimization results and justifying the approach.

1.4 Objectives

The specific objectives of this research are:

- Find a readily available, easy to use genetic algorithm platform that is compatible with a common computer-aided control design tool,
- Formulate an appropriate method for gain schedule optimization while considering alternatives,
- Validate the gain scheduling GA optimization process on a simple, representative control problem,
- Demonstrate the GA optimization process on a full envelope flight control example, and
- Compile the lessons learned into a concise design optimization procedure using GAs.

1.5 Preview

The next two chapters provide a brief summary of recent published work concerning gain scheduling and genetic algorithms. Chapter II summarizes developments in the theoretical aspects of gain scheduling. Chapter III presents some recent GA applications in the field of aircraft controls and briefly explains how GAs work. Additionally, some modifications

and improvements on the simple GA are discussed, and finally a brief GA software platform review is presented. Chapter IV presents a simple control problem to validate the use of GAs in gain schedule optimization. Chapter V presents information for an F-18 gain scheduled controller designed in [4] and the results of various optimizations of this gain schedule. Chapter VI presents a general gain scheduling optimization design process developed from the experience of this research effort, summarizes the results of this research work, and provides recommendations for future research.

II. Gain Scheduling

This chapter presents a review of gain scheduling and how it has been achieved in the past. By understanding the methods involved in scheduling a controller throughout an operating envelope, it becomes clearer how this proven technique can be improved. Section 2.1 provides an explanation of why gain scheduling is used. Next, section 2.2 describes the traditional methods of gain scheduling which are improved upon in this research effort. Concluding this chapter, section 2.3 explains how a gain schedule is "optimized" in this investigation.

2.1 Rational for Gain Scheduling

Many real world systems are nonlinear. Also, their dynamics may change as a function of the system's operating condition. There are two basic approaches to designing a controller for these systems. The preferred way is to design a controller that is robust enough to maintain stability and desired performance throughout the entire operating regime. There are several techniques for designing robust controllers including H_∞ and μ-synthesis [14, 15]. This can be accomplished for some problems as demonstrated by Shamma and Cloutier in designing a missile autopilot [47]. Unfortunately, this is not always possible. The second approach is to change a linear controller as the plant dynamics change. Hence, the controller parameters (typically *gains*) are *scheduled* as a function of system's operating condition. Gain scheduling has proven to be a very successful method of implementing a global controller from a set of linearized controllers [45].

There are some significant advantages to gain scheduling as pointed out by Rugh [41]. The main advantage is that linear design techniques can be utilized. When designing linear controllers a control designer has a wealth of computational tools, performance measures, experience, and knowledge to draw from to guide the design process. Another advantage is that a gain scheduled controller has the potential to respond rapidly to changing operating conditions whereas more modern techniques require more real time computation. However,

there are some difficulties involved in the design process [41]. The major difficulty is the selection of the scheduling procedure, the process by which the scheduling variables are changed. Another difficulty is in the selection of the scheduling variable. Fortunately, a couple "rules of thumb" have emerged to help overcome this second difficulty. First, schedule on a variable that captures the nonlinearities of the system, and second, schedule on a variable that varies *slowly*. However recent work by Shamma has shown that these rules actually have a rigorous mathematical justification [43, 46, 45, 44].

2.2 *Gain Scheduling Process*

Following is a typical method for designing a gain schedule [44].

1. Select several operating points covering the range of the plant dynamics.
2. Construct a linear time invariant (LTI) approximation to the plant and design a linear controller.
3. Interpolate the controller parameters between the operating points in order to determine control parameters for operating points between the selected design operating points.

To interpolate the controller parameters simple curve fitting techniques are used [41, 43]. However, there are no guarantees of stability or robustness without numerous simulations or theoretical analysis. An idealized gain schedule is defined as a gain schedule that has a controller designed at every operating point within the operating envelope [41]. Obviously, for any large number of operating points the idealized gain schedule is impractical. Nonetheless, approximations to this idealized gain schedule can be done where the resulting controller is selected by a table-look up method.

Fortunately, several methods and theorems have been published to analyze the system for global stability [29, 43, 46, 45, 44]. Unfortunately, these methods are based on scheduling simple controllers such as proportional-integral-derivative PID controllers and full state feed back gain matrices. These methods do not directly address the complexities of modern controller designs such as H_2, H_∞, and μ synthesis.

Recent works have been focusing on scheduling these modern controllers [6, 5, 18, 37]. These works use a method of linear fractional transforms (LFTs) to approach to problem of gain scheduling. The scheduling variable is included as an exogenous signal to be tracked or as a disturbance to be rejected. A bounded controller space is solved via linear matrix inequality (LMI) equations. Using this method, a controller for a linear parameter varying (LPV) plant is linear parameter varying itself. The resulting controller is thereby self scheduled.

2.3 Gain Scheduling Design

The traditional approach to gain scheduling is selected because of its wide use and acceptance. The goal of the design is to directly optimize the controller with respect to an objective function over the operating range of the system. This is different than optimizing a controller at a specific operating point and then finding a linear fit of the controllers over the operating range. Ideally the control designer would like to design a controller that responds exactly the same regardless of the operating condition. A gain scheduled controller of this type would be an optimal gain scheduled controller defined as follows:

Definition 1 (Optimal Gain Scheduled Controller) *An optimal gain scheduled controller provides a uniform response throughout the operating envelope.*

The optimality of a gain schedule can be measured by comparing a desired closed loop response to an actual response at a specific operating point. The desired response can be defined in either the time or frequency domain. The objective of the design optimization is to minimize the deviation between the actual response and the desired response at various points in the operating envelope. In other words, the gain scheduling error is minimized. Following are definitions to clarify the difference between tracking error and the gain scheduled error that is minimized in this investigation.

Definition 2 (Controller Tracking Error) *The difference between the commanded input and the closed loop output response at one specific operating point.*

Definition 3 (Gain Scheduling Error) *The difference between the closed loop response of the central controller and a non central controller.*

Definition 4 (Central Controller) *The controller selected at a specific operating point as having the desired response. The closed loop response of this controller is used in calculating the gain scheduling error.*

Therefore, the specific goal of the gain schedule design optimization is to minimize the gain scheduling error in the closed loop response throughout the operating envelope. The central controller can be a set of desired eigenvalues, a set of dominant closed loop poles, or a controller designed at a specific operating point [4]. The example optimization discussed in chapter IV selects a set of dominant closed loop poles to be the central controller, and in chapter V the central controller is a controller defined at a specific operating point. Each optimization performed demonstrates how the gain scheduling error can be minimized.

To clarify the difference the scheduling variable and the scheduled variable, they are defined as follows:

Definition 5 (Scheduling Variable) *The variable(s) that is representative of the changes in the plant dynamics and is a measurable signal.*

Definition 6 (Scheduled Variable) *The variable(s) that are changed as a function of the scheduling variable, thereby changing the controller as a function of the operating point.*

2.4 Optimization Approach

The first step is to discretize the scheduling variable range by breaking it into N intervals. At first the number of intervals are constrained to a arbitrary number. Additionally, the size of the intervals are also constrained, for simplicity all constrained interval sizes will be equal. Next, through some preliminary analysis the scheduled variables are bounded by their minimum and maximum values over the scheduling variable range. After defining an appropriate objective function, the gain scheduling error is minimized by scheduling the

controller parameters as piecewise constant or piecewise linear functions of the scheduling variable. For each interval, the scheduled variables are allowed to vary independently between their minimum and maximum bounds. Further optimization unconstrain the size of the intervals and the number of intervals used to discretize the scheduling variable.

III. Genetic Algorithms

Genetic Algorithms (GAs) are an optimization method based on Darwin's theory of "Survival of the Fittest.". GAs were first developed by Holland in 1975 [25]. Since then they have been extensively expanded by DeJong [11, 12], Goldberg [20, 21, 22], Grefenstette [23] and many others. GAs differ from classical methods of optimization in many ways. The next section describes these differences. The following two sections briefly explain how and why GAs work. Some improvements on these simple GAs are presented in section 3.4. A review of GA software is presented in section 3.5. Section 3.6 concludes this chapter with an explanation of the implementation of the GA.

3.1 Classical Optimization

Classical methods of optimization fall into three categories: calculus-based, enumerative, and random. Consider a general optimization problem statement [50]:

$$\text{Minimize: } F(\mathbf{X}) \qquad \text{the objective function} \tag{3.1}$$

Subject to:

$$g_j(\mathbf{X}) \leq 0 \quad j = 1, m \qquad \text{inequality constraints} \tag{3.2}$$

$$h_k(\mathbf{X}) = 0 \quad k = 1, l \qquad \text{equality constraints} \tag{3.3}$$

$$\mathbf{X}_i^l \leq \mathbf{X}_i \leq \mathbf{X}_i^u \quad i = 1, n \qquad \text{side constraints} \tag{3.4}$$

$$\text{where } \mathbf{X} = \left\{ \begin{array}{c} X_1 \\ X_2 \\ X3 \\ \vdots \\ X_n \end{array} \right\} \text{ the design variables}$$

Calculus based methods optimize this given problem by satisfying three Kuhn-Tucker conditions [50]. The Kuhn-Tucker conditions are (X^* is the optimal design vector):

$$1.\ \mathbf{X}^* \text{ is feasible} \tag{3.5}$$

$$2.\ \lambda_j\, g_j(\mathbf{X}^*) = 0 \qquad j = 1, m \qquad \lambda_j \geq 0 \tag{3.6}$$

$$3.\ \nabla F(\mathbf{X}^*) + \sum_{j=1}^{m} \lambda_j \nabla g_j(\mathbf{X}^*) + \sum_{k=1}^{l} \lambda_{k+m} \nabla h_k(\mathbf{X}^*) = 0 \tag{3.7}$$

$$\lambda_j \geq 0 \qquad \lambda_{k+m} \text{ unrestricted}$$

$$\text{where} \qquad \nabla F(\mathbf{X}) = \begin{bmatrix} \delta F(\mathbf{X})/\delta X_1 \\ \delta F(\mathbf{X})/\delta X_2 \\ \delta F(\mathbf{X})/\delta X_3 \\ \vdots \\ \delta F(\mathbf{X})/\delta X_n \end{bmatrix} \tag{3.8}$$

Both the power and the drawback of calculus-based methods becomes apparent in the third Kuhn-Tucker condition, Eq (3.7). The gradient of both the objective function and the constraints must be evaluated to determine the direction the search will proceed. For a function where these gradients do not have analytical solutions, they must be evaluated by a finite-difference method [50]. Additionally, for the calculus-based algorithm to determine whether the optimum that it has found is a maximum or minimum, it must evaluate the Hessian matrix of second derivatives (Eq (3.9)).

$$H = \begin{bmatrix} \frac{\delta^2 F(\mathbf{X})}{\delta X_1^2} & \frac{\delta^2 F(\mathbf{X})}{\delta X_1 \delta X_2} & \cdots & \frac{\delta^2 F(\mathbf{X})}{\delta X_1 \delta X_n} \\ \frac{\delta^2 F(\mathbf{X})}{\delta X_2 \delta X_1} & \frac{\delta^2 F(\mathbf{X})}{\delta X_2^2} & \cdots & \frac{\delta^2 F(\mathbf{X})}{\delta X_2 \delta X_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta^2 F(\mathbf{X})}{\delta X_n \delta X_1} & \frac{\delta^2 F(\mathbf{X})}{\delta X_n \delta X_2} & \cdots & \frac{\delta^2 F(\mathbf{X})}{\delta X_n^2} \end{bmatrix} \tag{3.9}$$

Even though calculus-based methods can be very efficient, they "break down" when the objective function or its gradient is not continuous or "well behaved". Moreover, the solution is dependent upon the initial conditions given to the algorithm. For a given function, a

calculus-based algorithm is guaranteed to find the nearest local extremum if it is exists. The solution is a local minimum if the Hessian matrix is positive definite at the solution point [50]. Unfortunately, the solution found may still not be the global minimum.

Enumerative techniques were developed to guarantee that the global minimum was found by exhaustively searching all points in the design space. Unfortunately, as the number of design variables increases the number of possible solutions increases exponentially. This is the "curse of dimensionality." Obviously for problems with a large number of design variables, the enumerative technique is inefficient.

Random methods were designed to increase the efficiency of the enumerative methods by randomly evaluating points in the search space. Since only random points in the solution space are evaluated there is no guarantee of finding the global optimum, therefore the algorithm continues until a large percentage of the search space has been evaluated. Again, as the number of design variables increases the number of possible solutions increases at a greater rate. Since random methods lack efficiency, they are not expected to obtain good results [27]. GAs provide an approach that overcomes these difficulties.

GAs differ from traditional optimization techniques in four basic ways: 1) they code the parameter set instead of the parameters themselves, 2) they search a population of points instead of a single point in the solution space, 3) they do not require an initial guess of the solution, and 4) they use probabilistic transition rules instead of deterministic transition rules. GAs are a zero order optimization method that does not depend on the behavior of the objective function. Consequently, they have provided robust optimization over a wide range of problems [3, 27, 32]. Some problems that have been optimized using GAs are shipping scheduling [34], dynamic control [32], control optimization [35, 27], floor plan area optimization [40], and investment market timing strategies [7]. Traditional and genetic optimization techniques were compared for efficiency and accuracy in an aircraft parameter design optimization [9]. In the study the authors performed an extensive comparison of calculus, enumerative, random, genetic, and simulated annealing algorithms. GAs and Simulated Annealing[26] were the only two methods to find the global optimum out of the fifteen algorithms tried.

Additionally advantages of GAs are: 1) they do not require the derivative of the function to exist, 2) they can handle objective functions with penalty functions without difficulty, unlike a calculus-based method that must transform the objective function into a more manageable form [27], 3) as the dimension of the problem grows the computation time for the GA grows in a linear fashion, whereas calculus-based methods grow at least quadratically [34, 35], 4) they are not dependent on the shape of the solution space for finding an optimum, and 5) the optimum found by the GA is more likely to be the global optimum value for the objective function since the solution is not dependent upon the initial condition given to the algorithm.

However, there are a couple of disadvantages to GAs. First, GAs are a naturally parallel algorithm that are typically run on a serial machine [32]. This observation was also made by Goldberg [20]:

> "In a world where serial algorithms are usually made parallel through countless tricks and contortions, it is no small irony that genetic algorithms are made serial through equally unnatural tricks and turns."

The inefficiency of serial computation significantly increases the computation time of the GA. Additionally, parallel GAs require fewer function evaluations than the best alternative algorithms [32]. Another disadvantage of GAs is that they are robust methods for searching the solution space to find the area in which the global optimum occurs, but they are inefficient at obtaining a solution where high precision is desired. The following paragraph reviews some literature that compares GAs and traditional calculus-based algorithms.

A GA was compared to Powell's conjugate search direction for two control system optimization problems [27]. The first problem was a linear quadratic regulator problem for a lateral autopilot which was solved via the algebraic Riccati equation [19] for the exact answer. Both the GA and Powell's method found the exact solution. A significant difference was that Powell's method required an order of magnitude more function evaluations. A second comparison was performed on a wind shear problem where the objective was to find a controller that minimized the deviation in the velocity and flight path of an aircraft due to a wind burst. The GA was able to find the exact solution, but Powell's method was dependent

upon the initial conditions and unable to find the global optimum. Similar experiments were performed by Michalewicz [34], who drew the following conclusions on the effectiveness of GAs applied to optimal control: 1) the objective function need not be continuous over the search space for GAs to find an optimum; 2) GAs give intermediate information while the problem is being solved so that when a desired degree of accuracy is reached computation can be halted, whereas other packaged optimization methods do not return a result until the optimization is complete; and 3) GAs grow linearly with the size of the problem. GAs are not the best optimization method for all problems, but for problems with high dimensionality and nonlinearities, GAs do have an advantage.

3.2 GAs: The Inner Workings

GAs explore and exploit various solutions by manipulating building blocks called *schemata.* GAs explore different possible solutions, and exploit the best solutions through the implementation of three genetic operators–*mutation, crossover,* and *reproduction.* GAs evolve a population of solutions from generation to generation through these genetic operators to an optimal solution.

GAs can either minimize or maximize a given objective function. Any minimization problem can be transformed into an equivalent maximization problem through the mapping depicted in Eq (3.10).

$$g_{max} = -f_{min} \tag{3.10}$$

Therefore, for simplicity all problems in this chapter will be considered maximization problems.

GAs encode the parameter range of the design variable and map it to a binary string of a specified length. For example, a binary string of length seven has a minimum value, 0000000, which is mapped to the minimum value of the design variable, x_{min}. The maximum value of the string, 1111111, is mapped to the maximum value of the design variable, x_{max}. Between these two extremes there is a linear mapping of the string values to the parameter

values. For a function of more than one variable, the strings representing each variable are concatenated together to form one chromosome. For example, a function with two design variables, with each having a string length of seven would form a chromosome of length fourteen, 00000001111111.

The following two sections will explain how the GA explores and exploits solutions through genetic operators.

3.2.1 Exploration. The main operator responsible for exploration of the search space is mutation. The mutation operator is controlled by a user specified probability, p_m. This probability is the chance that each bit in the chromosome string is mutated to its complement value of 0 or 1. For each bit in the population, a random number between zero and one is generated. If the random number is less than p_m, that bit is mutated. For example, suppose that a chromosome has the form shown in Fig. 3.1(a). If the fourth bit is chosen for mutation, the chromosome is changed to the new chromosome shown in Fig. 3.1(b). This new chromosome represents a new possible solution.

1011010110
(a)
1010010110
(b)

Figure 3.1 Mutation

The frequency of mutation for a specified probability, p_m, is dependent on the string length, m, of the chromosome. Eq (3.11) describes the expected number of mutations per generation.

$$E[\text{\# of mutations}] = n \times m \times p_m \tag{3.11}$$

where n = the population size, m = the string length, and p_m = the user specified mutation probability.

For a string length $m = 100$ and $p_m = 0.01$, the expected number of mutations per string is 1. For a string length $m = 50$, then expected number of mutations is only 1 per every two

01110|**10110**
10101|**01100**
(a) Parents
01110|**01100**
10101|**10110**
(b) Offspring

Figure 3.2 Crossover

strings in the population. Consequently, for a population size of $n = 20$, a string length of $m = 50$, and $p_m = 0.01$, the expected number of mutations per generation is 10. Typically, values of p_m range from 0.01 to 0.001 [23]. For values of $p_m \geq 0.05$, the GA approaches a random search method.

3.2.2 Exploitation. Two genetic operators are responsible for exploitation of good solutions, crossover and reproduction. The following two sections will describe these operators in more detail.

3.2.2.1 Crossover. The crossover operator acts on two solution strings simultaneously by swapping information included in each string. The probability that crossover occurs, p_c, is set by the user. Once two strings are chosen for crossover, a position between the bits is randomly chosen with a uniform distribution. Suppose that the two strings shown in Fig. 3.2(a) are chosen for crossover, and that the position between the fifth and sixth bit is randomly selected as the crossover point. At the crossover point the strings are cut. The partial strings are swapped, thereby forming the two new strings shown in Fig. 3.2(b).

Crossover, unlike mutation, is independent of the string length, but is still a function of the population size. Eq (3.12) describes the expected number of crossovers per generation.

$$E[\text{\# of strings selected for crossover}] = n \times p_c \tag{3.12}$$

For a population size of $n = 50$, and $p_c = 0.6$, the expected number of strings selected for crossover would be 30. Typically, values for p_c range from 0.6 to 0.9 [23].

Table 3.1 Reproduction Example

string	*fitness value*	f/f_{tot}
01101001	28.6	0.36
10010010	16.4	0.21
01011011	23.7	0.30
10110110	9.8	0.13
	78.5	1.00

3.2.2.2 Reproduction. The reproduction operator acts on the population only after both the mutation and crossover operators have been implemented. The reproduction operator reproduces each solution with a probability proportional to the fitness value of each string. The strings with above average fitness values are reproduced at higher rates than strings with below average fitness values. Consequently, poor solutions die out of the population. By not eliminating the worst solutions immediately, the population remains diverse and therefore does not converge prematurely to a local optimum.

Consider a population of size $n = 4$ shown in Table 3.1. Each string is evaluated using an objective function to find the respective fitness of each string. The total sum of all the fitness values is 78.5. Each string has a chance of being selected for the next generation proportional to its fitness value, f, divided by the total fitness value of the population, f_{tot}. Therefore, the probability that a string is selected for the next generation is:

$$P(\text{selection for next generation}) = \frac{f}{f_{tot}} \tag{3.13}$$

The sum of these probabilities totals 100 per cent. Reproduction takes place by giving each string a space on a roulette wheel proportional to its relative fitness, f/f_{tot} (See Fig. 3.3). The roulette wheel is spun four times to select the individuals for the next generation.

This concludes the basics of a GA. The next section will explain how these genetic operators work together to find an optimal solution to a given objective function.

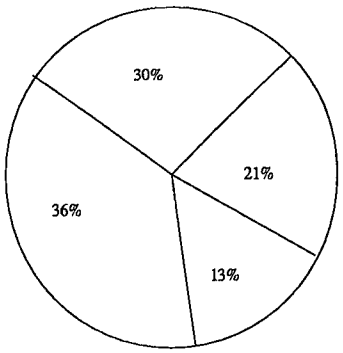


Figure 3.3 Reproduction Roulette Wheel

3.3 Schemata

Schemata [1] are a representation of a set of binary strings used in the theoretical structure of GAs. To understand schemata, a "don't care" symbol '*' is introduced. A schema of length eight, $S_0 = \{**101011\}$, would represent any string of the same length that had 1s and 0s in the same positions as the schema itself. A schema with r "don't care" symbols has 2^r strings matching that schema. Every schema matches 2^r strings, where r is the number of "don't care" symbols in the schema. Thus, the following strings would match schema S_0.

$$(00101011),\ (01101011),\ (10101011),\ (11101011)$$

Conversely, each string of length m is matched by 2^m schemata. For example, the string (10110) is matched by the following 2^5 schema:

(10110)
(*0110)
(1*110)
(**110)
(10*10)
$\vdots$
(*****)

[1]Note: Schemata is the plural of schema.

There are two main characteristics used to define schema, order and defining length. Order, denoted by o(S), is the number of 1s and 0s specified in the schema. It defines the specialty of the schema. A high order schema is more specific than a low order schema. A schema of high order is more likely to be disrupted by mutation than a low order schema. The order of two schemata are shown in Fig. 3.4.

The defining length, denoted by $\delta(S)$, of a schema is the distance between the first and last fixed positions. The defining length quantifies the compactness of a schema. The minimum defining length is zero for a single specified position, and the maximum defining length for a string of length m is $(m-1)$. Defining length is a determining factor in a schema surviving crossover, with compact schemata more likely to survive. The defining length of two schemata are also shown in Fig. 3.4.

S_0={***11***} $\quad o(S_0) = 2 \quad \delta(S_0) = 5 - 4 = 1$
S_1={*10**1*1} $\quad o(S_1) = 4 \quad \delta(S_1) = 8 - 2 = 6$

Figure 3.4 Order and Defining Length

3.3.1 Survival of the Fittest. For a population of strings with length m, there are a total of 3^m possible schema. With a population of size n, there will only be 2^m to $n \times 2^m$ schemata represented, with some more fit than others. The fitness of a schema is determined by the average fitness of the strings matching schema S. Let N(S,t) represent the number of strings in generation t that match schema S. Then the following equation calculates the fitness of a schema S in generation t.

$$eval(S,t) = \sum_{i=1}^{N(S,t)} \frac{f_i}{N(S,t)} \tag{3.14}$$

From this relation the expected number of strings in the next generation, $t+1$, can be determined for a population with given average fitness ($f_{avg} = f_{tot}/n$).

$$E[N(S,t+1)] = N(S,t)\frac{eval(S,t)}{f_{avg}} \tag{3.15}$$

A schema with an above average fitness will receive an increasing number of strings in the subsequent generations as shown in Eq (3.15). This increase is exponential for a schema with a consistently above average fitness [25, 34].

However, the genetic operators, crossover and mutation, can destroy a schema and therefore reduce this growth rate of the schema. Consequently the probability of crossover destroying a schema is:

$$p_d = \frac{\delta(S)}{m-1} \times p_c \tag{3.16}$$

Therefore the probability of survival is:

$$p_s = 1 - \frac{\delta(S)}{m-1} \times p_c \tag{3.17}$$

However, there is a chance that even if crossover occurs within the defining length of a schema, the schema will survive. For an example of this see Fig. 3.5. Therefore, the probability of survival is slightly larger:

$$p_s \geq 1 - \frac{\delta(S)}{m-1} \times p_c \tag{3.18}$$

S_0={***11***}
0111|**1011**
1101|**1100**
(a) Parents
0111|**1100**
1101|**1011**
(b) Offspring

Figure 3.5 Schema Surviving Crossover

The probability that a schema survives mutation depends on its specialty:

$$p_s = (1 - p_m)^{o(S)} \approx 1 - p_m \times o(S) \quad \text{for } p_m \ll 1 \tag{3.19}$$

Combining the effects of crossover and mutation, Eq (3.15) becomes:

$$E[N(S,t+1)] \geq N(S,t)\frac{eval(S,t)}{f_{avg}}\left[1 - p_c\frac{\delta(S)}{m-1} - p_m o(S)\right] \tag{3.20}$$

To demonstrate how this growth works consider the following schema:

$$\text{S=}\{**110**************\}$$

$$m = 21$$

$$\delta(S) = 5 - 3 = 2$$

$$o(S) = 3$$

$$N(S,t) = 4$$

$$eval(S,t) = 20$$

$$f_{avg} = 12$$

$$p_c = 0.6$$

$$p_m = 0.001$$

$$E[N(S,t+1)] \geq 4.0 \times 1.667 \times 0.938 \approx 6$$

For a below average schema the number of strings in subsequent generations would decrease in a similar manner. The following theorem describes the concept given quantitatively in Eq (3.20). This theorem is known as the fundamental theorem of GAs [34].

Theorem 1 (Schema Theorem) *Short, low-order, above average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.*

3.4 Improvements on the Simple GA

Several modifications have been proposed to improve the performance of the simple GA (sGA). One of the simplest ways of improving the precision of the GA is to use a floating point representation of the design variables instead of mapping them to binary strings. Two different research efforts have found that the use of floating point genes has increased the speed, precision and the accuracy of the GA [10, 34].

There are two main qualities of the GA that have proven it to be a robust optimization method for a wide range of practical problems: its power to explore new solutions, and the ability to exploit the best solutions it has found. The user has control over these parameters by selecting the crossover probability, p_c, and the mutation probability, p_m. For improved results the user might want to vary these parameters over the computation time of the algorithm. The following two sections discuss some of the ways that crossover and mutation operators may be varied or adaptively changed. The third section 3.4.3 discusses the advantages of using a messy GA (mGA) instead of a sGA.

3.4.1 Crossover Operator . One simple way of changing the crossover operator is to vary the number of points at which crossover can occur. By allowing multi-point crossover to occur, some schemata with long defining lengths can be preserved which would otherwise be destroyed with single point crossover. For example, two point crossover would randomly select two points along the length of the chromosome for crossover (See Fig. 3.6). Then the portion of the string between the two crossover points are swapped between the two parents to produce two new offspring.

01110|**10110**|110111
10101|**01100**|011010
(a) Parents
01110|**01100**|110111
10101|**10110**|011010
(b) Offspring

Figure 3.6 Two Point Crossover

Further generalization of this concept can lead to the idea of uniform crossover. Uniform crossover decides with probability, p_c, which bit positions of the first parent will be exchanged with the second parent. This is similar to the mutation operator in the sense that each bit has a chance to be crossed with another bit from the second parent. For an example of this type of crossover, see Fig. 3.7 ($p_m = 0.5$).

1001100110
0110101011
(a) Parents
1100100110
0011101011
(b) Offspring

Figure 3.7 Uniform Crossover

Another innovative way of changing the crossover operator is to change the probability that crossover occurs as a function of the average and best fitness of the population [48]. This operator is shown in Eq (3.21).

$$p_c = k_1(f_{max} - f')/(f_{max} - f_{avg}) \quad \text{for} f' \geq f_{avg}$$
$$p_c = k_2 \quad \text{for} f' < f_{avg} \tag{3.21}$$

where f' is the largest fitness value of the two parents selected for crossover, f_{max} is the largest fitness value of the population, and f_{avg} is the average fitness value of the population.

The parameters k_1 and k_2 are scaling parameters that can be chosen arbitrarily. The authors of [48] chose $k_1 = 1.0$ and $k_2 = 1.0$. With this crossover operator each individual in the population has a different crossover probability depending on its fitness value. Individuals with above average fitness values have a higher crossover probability than those with below average fitness. The probability of crossover is 0.0 for individuals with fitness values equal to f_{max}. Davis [10] presented another method of adaptive crossover where the crossover probability is dependent upon the fitness of the offspring produced. The greater the offspring's fitness, the greater the probability that crossover will occur at that bit location.

3.4.2 Mutation Operator. The choice of p_m is critical in the performance of the GA [11]. Two ways of adaptively changing the mutation operator are presented. Both have shown to provide increased performance in the GA in a sampling of test problems [34, 48]. The first method changes p_m as a function of the current generation number and the total number of

generations. This mutation operator was designed as a floating point mutation operator, but it can be transformed into a binary operator with an appropriate mapping [34]. This mutation operator is shown in Eq (3.22).

$$\Delta(t, y) = y(1 - r^{(1-t/T)^b}) \tag{3.22}$$

where T is the maximum number of generations, t is the current generation number, r is a random number $\in$ [0,1], b is a system parameter determining the degree of dependence on iteration number (a value of 5 is used for experimental results), and y is the value of the gene being mutated.

The effect of this operator is that when the GA begins, there is a high probability of mutation. This maintains the diversity of the population searching for new solutions. As the GA nears completion, $t \rightarrow T$, mutation is decreased allowing crossover to become dominant and converge to an optimal solution.

Another published adaptive mutation operator changes the mutation probability depending on an individual's fitness value instead of the generation number [48]. Eq (3.23) defines the proposed mutation operator.

$$\begin{aligned} p_m &= k_3(f_{max} - f)/(f_{max} - f_{avg}) \quad \text{for } f \geq f_{avg} \\ p_m &= k_4 \qquad\qquad\qquad\qquad\quad \text{for } f < f_{avg} \end{aligned} \tag{3.23}$$

where f is the fitness of the individual, f_{max} is the maximum fitness in the population, and f_{avg} is the average fitness of the population.

The values of k_3 and k_4 can be chosen by the user; the authors of [48] chose values of 0.5 for both. Individuals with above average fitness have lower mutation probabilities than individuals with below average fitness. The mutation probability is 0.0 for individuals with fitness equal to f_{max}. Individuals with below average fitness are totally disrupted. The combined effect of this mutation operator and the adaptive crossover in Eq (3.21) is to preserve individuals with the best fitness and to completely disrupt individuals with the worst fitness.

3.4.3 Messy Genetic Algorithms. Messy GAs (mGAs) were originally developed by Goldberg [21, 22] to overcome the problem of the sGA converging to local optima. As mentioned in section 3.3, for a string length, m, there are 3^m different possible schemata. Unfortunately, for a population of size n, only 2^m to $n2^m$ schemata are represented. This problem is overcome by mGAs. The algorithm is divided into two phases. The first phase is known as a tournament and the second phase is similar to traditional GAs. The tournament generates a large number of schemata of various sizes and then reduces this population of schemata down to a manageable size for the second phase. The second phase takes the reduced population and uses genetic operators splice, cut, and mutation to achieve an optimal solution. The operators splice and cut replace the crossover operator. See [21, 34] for a description of the splice and cut operators. Goldberg states that the mGA was able to find the optimal solution to a difficult problem, where the sGA only found the optimal solution 25 per cent of the time [21]. Developments and improvements of the mGA have been accomplished here at AFIT [33, 16].

3.5 Software Review

This section provides a survey of available software that was considered for this thesis. For a complete review of existing software available, the reader is referred to [17]. The GA software available can be divided into three main categories: application specific, algorithm specific, and general purpose toolkits. There are no application specific software available for this research topic. The next two sections will discuss software that was investigated in the two remaining categories.

3.5.1 Algorithm Specific. For algorithm specific software a source code is provided and the user is able to make alterations to the code. Typically the code is in a higher level language like 'C' and the user interfaces are rudimentary. The following is a list of the software that was reviewed:

GENESIS (GENEtic Search Implementations System) was written by John Grefenstette and has been under development since 1981. The source code is in 'C' and allows for a high degree of modifiability with large amount of statistical information. It was primarily developed for work in a scientific environment.

GAUCSD (Genetic Algorithm University of California San Diego) was written by Nicol Schraudolph. It is based on GENESIS version 4.5, but it provides a higher level of abstraction for defining the evaluation function. It allows direct use of most 'C' functions and it has parallel capabilities.

3.5.2 General Purpose Toolkits. These software systems provide toolkits of accessories that can be used interchangeably. The toolkits include different graphics interface and a library of genetic operators. The software is designed to have a user friendly interface. The library of genetic operators enables the user to experiment with different operator combinations. Following is a list of the software reviewed.

SPLICER was developed by Software Technology Branch of Information Systems Directorate at NASA/Johnson Space center with support from MITRE Corporation. It has a modular architecture that works using Xwindows. It was programmed in 'C' and has graphics output capabilities.

GAME (Genetic Algorithm Manipulation Environment) is an algorithm used by a large majority of the European community. It is designed for parallel capabilities in the 'C++' programming language.

3.6 Implementation

3.6.1 Software Selection. GENESIS was selected as the software package to use for this thesis for a couple of reasons. First, it is a readily available public domain software. Second, work has been accomplished here at the Air Force Institute of Technology (AFIT) that has modified the basic GENESIS code into a mGA [33, 16]. With these improvements any work accomplished in this thesis could easily be implemented with the modified codes to

improve the computation time. The goal of the implementation is to make it simple to take a controller design and optimize it with as little additional work as possible. For this reason the evaluation of the objective function will be accomplished in the $Matlab^{©}$ environment. This is convenient since $Matlab^{©}$ has a large amount of controller evaluation routines already programmed. GENESIS is coded in 'C'. To link the two together, $Matlab^{©}$ is used as a computational engine. The GA sends a vector of the design variables to $Matlab^{©}$ and a user written m-file calculates the objective function and passes the scalar value back to GENESIS.

A number of the GENESIS files were modified to facilitate this passing of variables between programs. The modifications required to run $Matlab^{©}$ as a computational engine are documented in the $Matlab^{©}$ user's manual [31]. Both the evaluation files, the 'C' code and the m-file, are listed in Appendix B.

IV. Sample Control Example

This chapter presents a simple control problem for which a gain schedule is designed and optimized using GAs. The results of this design clearly demonstrate that the gain schedule design process can be optimized. Further, for comparison, the gain schedule is optimized with calculus based algorithms and GAs. Since, this is a simple optimization problem, the calculus based methods do show good results, but they are useless on the full optimization problem shown in section 4.4. The problem is defined in section 4.1 and the optimization results are presented in sections 4.2, 4.3, 4.4, and 4.5.

4.1 Problem Statement and Analysis

The sample problem considered is a SISO third-order linear parameter varying plant with proportional feedback shown in Fig. 4.1. The scheduling variable $c \in (0, 10)$ determines the plant dynamics. The range of c is divided into several intervals I_i, $i = 1 \ldots n$, and a gain, k_i, is selected for each. The controller gain is simply the gain corresponding to the interval the current value of c is in. The objective is to find the set of gains that minimize the average deviation of the actual dominant closed loop poles from desired dominant pole locations over the entire range of c . For efficiency, the objective function was approximated by evaluating the deviation at 500 equally spaced values of c and adding them together. The desired pole locations are $-2 \pm 2j$. Therefore, the objective function is:

$$J_1 = \sum_{i=1}^{500} \{\max[\mathrm{Re}(\lambda_i)] + 2\}^2 + \{\max[\mathrm{Im}(\lambda_i)] - 2\}^2 \tag{4.1}$$

where λ_i are the closed loop poles of the system at a given value of c.

There are four increasingly difficult variations on this basic problem. First, the number of intervals and their sizes are chosen and the gains for each interval are found. Results of this optimization are in section 4.2. Second, only the number of intervals is chosen and the interval size and gains are optimized. These results are in section 4.3. Third, all three parameters, the

number of intervals, the size of each interval, and the gain for each interval , are optimized. Results for this are in section 4.4. Finally, the gains are allowed to be a linear function of the scheduling variable and the coefficients of the function are optimized. These results are in section 4.5.

In summary, the interval size and number can be specified where the gain of each interval is a design variable, or just the number of intervals can be specified and both the interval endpoints and gains are design variables, or the number of intervals and their respective endpoints and gains can be design variables and trade off complexity (number of intervals) for accuracy of the closed loop poles.

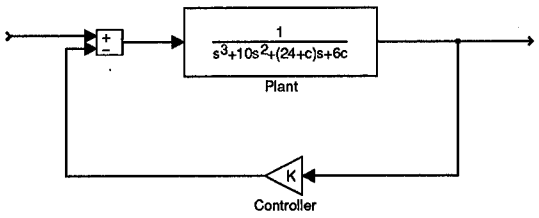


Figure 4.1 Example System Block Diagram

4.2 Fixed Interval Results

Initially, five equal intervals are chosen, (0,2), [2,4), [4,6), [6,8), and [8,10), and the optimal gains for each interval are found. This optimization is done using a simple GA with the following parameters, $p_c = 0.95, p_m = 0.01, popsize = 50$. Each design variable had a gene of length 14 for a precision of two decimal places. This resulted in a total string length of 100. For comparison, the same optimization is done with a Broydon-Fletcher-Goldfarb-Shanno (BFGS) Quasi-Newton method with a mixed quadratic and cubic line search method. There are five design variables to be optimized, the gain for each interval, I_i, $i = 1 \ldots 5$. The results of both the GA and the BFGS method are shown in Table 4.1. A mapping of the

Table 4.1 Fixed Interval Results

	GA	*BFGS*
# Iterations	182	15
# Function Evals	8323	128
Interval 1 Gain	31.61	31.61
Interval 2 Gain	22.12	22.12
Interval 3 Gain	13.02	13.02
Interval 4 Gain	4.40	4.41
Interval 5 Gain	-3.62	-3.59
Obj. Function	91.20	91.20

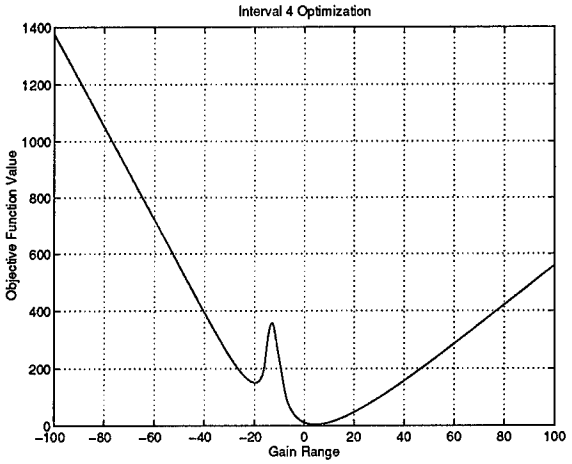


Figure 4.2 Exhaustive Interval Optimization

solution space of one of the intervals is shown in Fig. 4.2. For each fixed interval the solution is very similar, with one global optimum and one local optimum.

The GA required a significantly greater number of function evaluations than the BFGS method. However, this was expected since the objective function is smooth and the derivative is well-behaved for this example. Using a GA at this level is clearly overkill, but it is necessary to verify that the expected results are obtained.

4.2.1 Parameter Study. A restricted experimental study was performed to find the best combination of GA control parameters, p_c and p_m. Twelve different combinations of these parameters were tried. Each experiment had a population size of 50 and was run

for 10,000 function evaluations. Each experiment was performed ten times with a different initial population. The results of each experiment are listed in Table 4.2. Since GAs are not efficient at obtaining a solution with high precision, each experiment measured the number of generations and function evaluations required for the GA to converge to within 1 per cent of the optimal solution. Fig. 4.3 shows the average values and their respective standard deviations for each experiment.

A number of interesting trends are prevalent from these graphs. First, as the crossover probability is reduced, the number of function evaluations per generation decreases. This is a result of the GA only evaluating the individuals of the population that are new. The second interesting trend is that experiment number 1 had the lowest number standard deviation for both the number of generations and the number of function evaluations, 7.71 and 365 respectively. However, the next smallest standard deviation values, 13.01 and 561, respectively, are significantly higher. Additionally, these values are not from the same experiment. Since the parameter combination of experiment number 1 showed the greatest degree of consistency in converging to a solution, these parameters are used in the remainder of the optimization processes.

4.3 Variable Interval Results

The same design problem is now repeated with the interval end points as additional design variables. There are now a total of nine design variables. For comparison, the same problem is optimized using a sequential quadratic programming (SQP) method. For both the GA and the SQP method the interval end points are constrained to be within the range of $(0, 10)$ and the gains are constrained to be within the range of $(-50, 50)$. The results from these optimizations are shown in Table 4.3.

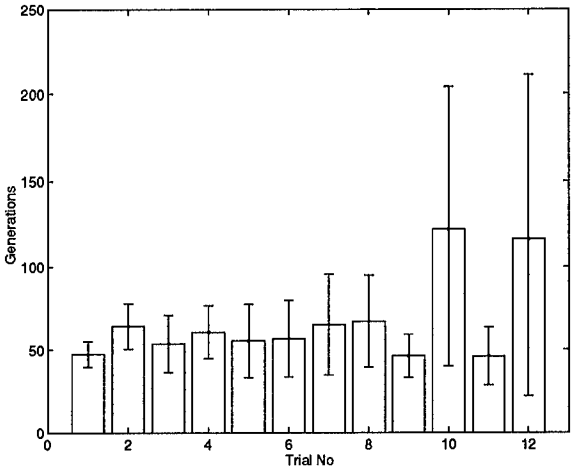
In Table 4.3, the first column of the SQP category shows the results of the SQP optimization when given the same initial condition as the fixed interval optimization. The SQP method stalled at one solution point for numerous iterations trying to find the correct direction to proceed. In addition, when evaluating the derivative of the function, the derivative matrix

Table 4.2 Parameter Study Results

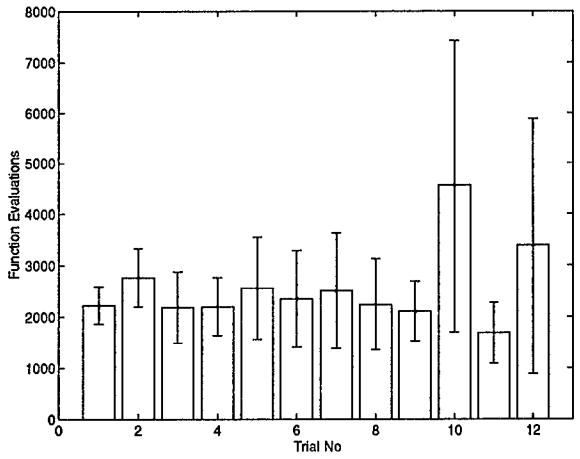
Trial No.	p_c	p_m	Average Number of Generation	Average Number of Function Evaluations	Average No. Function Evaluations per Generation
1	0.95	0.010	47.40	2227	46.97
2	0.85	0.010	64.40	2764	42.99
3	0.75	0.010	53.70	2186	40.83
4	0.65	0.010	60.70	2202	36.36
5	0.95	0.005	55.40	2558	46.37
6	0.85	0.005	56.70	2356	41.78
7	0.75	0.005	65.20	2510	38.86
8	0.65	0.005	67.10	2244	33.68
9	0.95	0.001	46.20	2108	45.76
10	0.85	0.001	122.10	4565	38.48
11	0.75	0.001	46.00	1684	36.96
12	0.65	0.001	116.40	3384	29.94

Table 4.3 Variable Interval Results

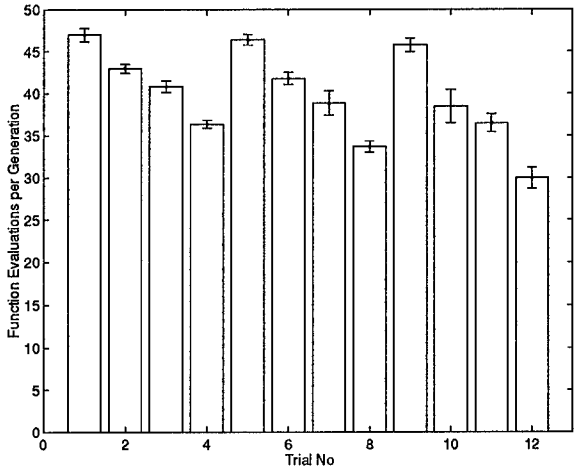
	GA	*SQP*	
# Iterations	704	109	47
# Function Evals	39000	1802	821
Endpoint 1	2.18	0.37	1.94
Endpoint 2	4.36	0.40	3.88
Endpoint 3	6.16	4.43	5.81
Endpoint 4	8.02	5.10	7.74
Interval 1 Gain	31.10	35.33	31.74
Interval 2 Gain	21.01	-11.67	22.59
Interval 3 Gain	11.86	25.83	13.69
Interval 4 Gain	4.04	13.77	5.34
Interval 5 Gain	-3.63	3.72	-3.02
Obj. Value	90.58	132.41	90.54



(a) Generations



(b) Function Evaluations



(c) Functions Evaluations per Generation

Figure 4.3 Parameter Study Results

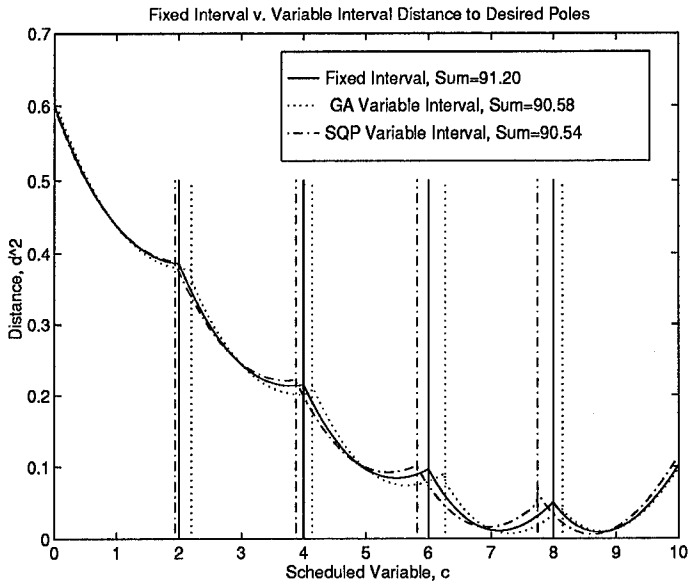


Figure 4.4 Variable Interval vs. Fixed Interval Pole Placement Error

became ill conditioned and nearly singular. The resulting solution of this SQP optimization is a local minimum. The SQP method was restarted with the fixed interval *optimal* solution as its initial conditions. Although the same numerical difficulties were encountered again, the SQP algorithms did eventually find an optimal solution. However, the numerical inaccuracies encountered during the optimization make the validity of the solution questionable. The GA did not have these problems because it is not dependent upon the local behavior of the objective function or its derivative.

The second derivative of the objective function was evaluated to gain insight into the shape of the solution space. Since an analytical derivative is not practical, a second order finite difference was used to calculate the diagonal elements of the Hessian. At the optimal values of the fixed interval solution found from the BFGS method, the diagonal elements are $[-6 \times 10^7 \ -3 \times 10^7 \ -1.5 \times 10^7 \ -1.5 \times 10^7 \ 0 \ 0 \ 0 \ 0 \ 0]$. The first four terms are the derivatives with respect to the interval end points, and the last five terms are the derivatives with respect to the gains of each interval. From these results we conclude that the solution space is a long, steep n-dimensional trough.

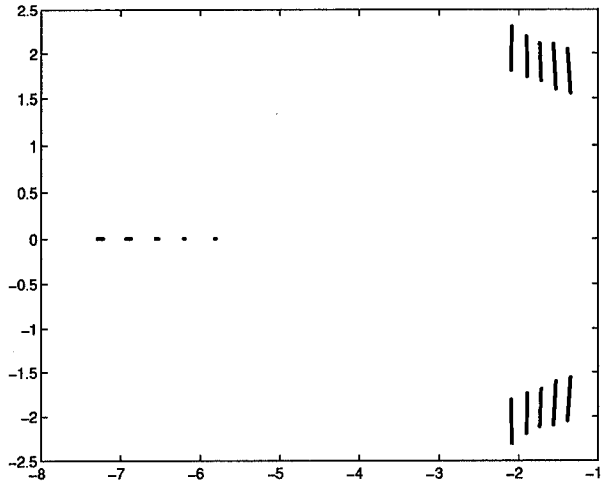


Figure 4.5 Locus of Roots of the Closed Loop System

Fig. 4.4 shows how the pole placement error varies with c for both the variable interval solutions and the fixed interval solutions. Since the objective of the optimization is to minimize the distance from the desired dominant closed-loop pole locations, the time response of the system at various values of the scheduled variable should be very similar. Fig. 4.5 shows a root locus of the closed loop system as c varies from $(0 \ldots 10)$. Fig. 4.6 shows the step response for different values of the scheduled variable from $(0 \ldots 10)$ in increments of 0.5. As expected, the time response in terms of overshoot, rise time, and settling time are all quite similar.[1]

4.4 Variable Number of Intervals Results

For this optimization the objective function is changed to penalize the number of intervals. The parameters of the sample problem are the same as those for the variable interval case in section 4.3, except that the number of intervals is allowed to vary between two and nine. For this example, the number of design variables varies. If there are only two intervals, there are three design variables (one interval end point, and two controller gains). Similarly, if there are nine intervals, there are seventeen design variables. Hence, if N intervals are chosen, there are 2N-1 design variables. Two different objective functions are used to observe the

[1]The steady state error shown could be corrected with a PI controller.

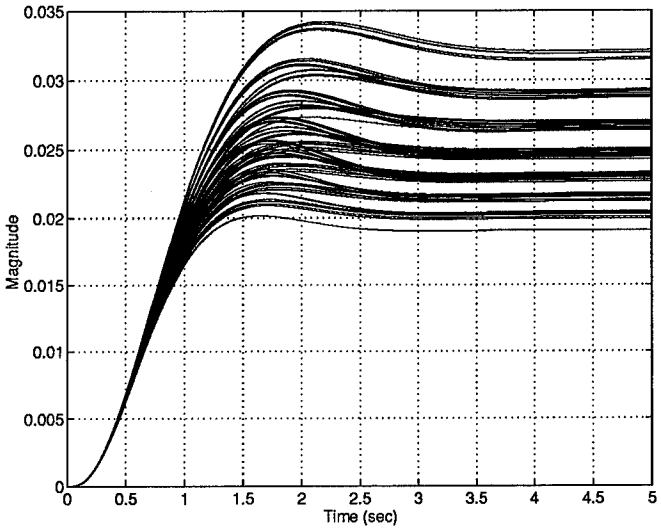


Figure 4.6 Time Response for Various Values of Scheduled Variable

effects of the penalty functions; Eq (4.2) uses a quadratic penalty function, and Eq (4.3) uses a linear penalty function.

$$J_2 = \sum_{i=1}^{500} \{\max[\mathrm{Re}(\lambda_i)] + 2\}^2 + \{\max[\mathrm{Im}(\lambda_i)] - 2\}^2 + (N-1)^2 \tag{4.2}$$

$$J_3 = \sum_{i=1}^{500} \{\max[\mathrm{Re}(\lambda_i)] + 2\}^2 + \{\max[\mathrm{Im}(\lambda_i)] - 2\}^2 + (N-1) \tag{4.3}$$

where λ_i is the closed loop poles of the system at a given value of c.

Both equations were optimized using the GA with the same parameters presented in section 4.2. The results of these optimizations are shown in Table 4.4. A calculus based algorithm was not able to optimize this problem because of the varying number of design variables.

These results show that the initial arbitrary use of five intervals is too large given these objective functions. Of course, this result depends on the relative weighting between the two parts of the objective. A more general objective function would include a multiplicative

Table 4.4 Variable Number of Intervals Optimization Results

	J_2	J_3
# Iterations	603	853
# Function Evals	18100	27520
# of Intervals	4	4
Endpoint 1	2.01	2.24
Endpoint 2	4.31	4.57
Endpoint 3	6.86	7.11
Interval 1 Gain	31.60	31.33
Interval 2 Gain	21.50	20.64
Interval 3 Gain	10.68	9.86
Interval 4 Gain	-0.89	-1.51
Obj. Function	105.3	98.6

weighting term in the penalty function. It is interesting to note that there was not a significant increase in computation time to do this analysis and optimization of the number of intervals.

4.5 Linear Interpolation

Since the trend in the optimized gains is linear, a final optimization is accomplished. The controller gain is chosen to be a linear function of the scheduling variable c.

$$K = a_1 c + a_0 \tag{4.4}$$

The coefficients of the linear function, a_1 and a_2, are the design variables. The BFGS calculus method used in section 4.2 was also used here. The GA parameters are $p_c = 0.95$, $p_m = .001$, and $popsize = 50$. Table 4.5 compares the results of the GA and the BFGS method. A plot of the root locus is shown in Fig. 4.7. Since, this is a simple system, the gain varies linearly with the scheduling variable c. Because of this direct relation the calculus based method quickly finds the optimal set of coefficients. The probabilistic nature of the GA requires more function evaluations to hone in on the optimal solution.

4.6 Summary

This work has shown some distinct advantages to using a GA for gain schedule optimization. For the simple fixed interval case, the GA reached the same solution as a calculus based method. However, when the interval size was allowed to vary the calculus based method stalled and its result was dependent on the initial conditions. More importantly, the GA allowed further trade-offs between the basic objective function and the number of intervals with practically no increase in computational effort. Considering these key points, the GA definitely shows promise for application to real world gain scheduling problems.

Table 4.5 Linear Scheduling Results

	GA	BFGS
# Iterations	48	7
# Function Evals	1954	38
a_1	-4.350	-4.377
a_0	34.946	35.1139
Obj. Function	82.31	82.30

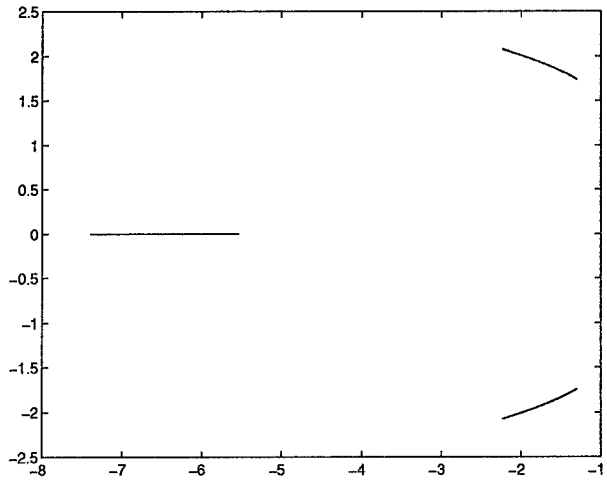


Figure 4.7 Locus of Roots of the Closed Loop System

V. Flight Envelope Design Example

The previous chapter demonstrated the applicability of gain scheduling optimization on a simple SISO system. Now, an optimization of a real world system is completed. A full envelope controller for an F-18 Supermanueverable fighter in Fig. 5.1 was designed in [4] using a simple gain scheduling technique of linear interpolation. This chapter optimizes this gain schedule using the methods in Chapter IV. First, the linearized equations of motion are derived in section 5.1. Next, the form of the controller and the gain schedule developed in [4] are presented in section 5.2 as a baseline for optimization. Afterwards, a measure of relative error is presented and the optimization function is developed in section 5.3.

5.1 Equations of Motion

The nonlinear equations of motion of an aircraft are shown in Eq (5.1) and Eq (5.2). Eq (5.1) describes the forces along the aircraft body axes, and Eq (5.2) describes the rotational forces about the same axes.

$$\begin{aligned} F_x &= m(\dot{U} + WQ - VR + \mathrm{g}\sin\Theta) \\ F_y &= m(\dot{V} + UR - WP - \mathrm{g}\cos\Theta\sin\Phi) \\ F_z &= m(\dot{W} + VP - UQ - \cos\Theta\cos\Phi) \end{aligned} \tag{5.1}$$

$$\begin{aligned} L &= \dot{P}I_x + \dot{R}I_{xz} + QR(I_z - I_y) - PQI_{xz} \\ M &= \dot{Q}I_y + PR(I_x - I_z) - R^2I_{xz} + P^2I_{xz} \\ N &= \dot{R}I_z + \dot{P}I_{xz} + PQ(I_y - I_z) + QRI_{xz} \end{aligned} \tag{5.2}$$

F_x, F_y, and F_z are the external forces and L, M, and N, are the external moments. U, V, W are the translational velocities along the X, Y, and Z body axes directions respectively. P, Q, and R are the rotational velocities about the X, Y, and Z axes, respectively. I_x, I_y, I_z, and I_{xz} are the moments of inertia, g is gravity, and m is the aircraft mass.

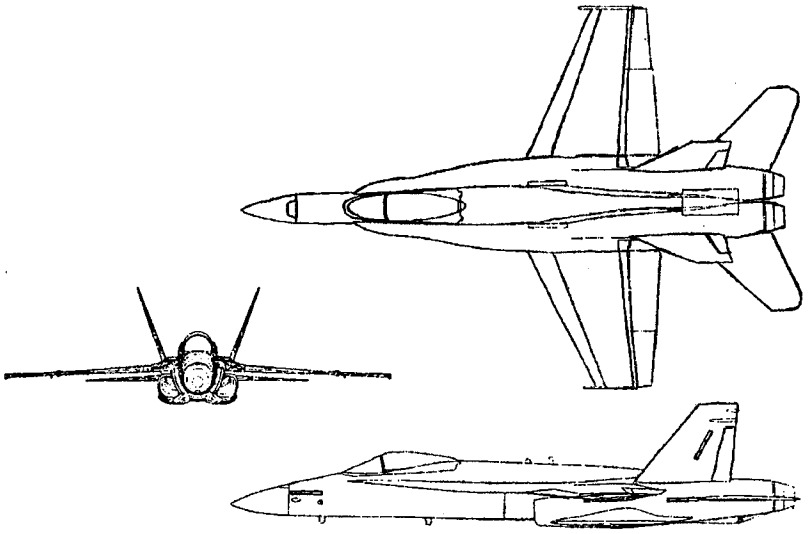


Figure 5.1 F-18 Aircraft

Since these forces and moments are in the body axis system, Eq (5.3) relates the aircraft orientation to the Earth through the Euler angles Θ, Φ, and Ψ.

$$\begin{aligned}
\dot{\Theta} &= Q\cos\Phi - R\sin\Phi \\
\dot{\Phi} &= P + Q\tan\Theta\sin\Phi + R\tan\Theta\cos\Phi \\
\dot{\Psi} &= \frac{R\cos\Phi}{\cos\Theta} + \frac{Q\sin\Phi}{\cos\Theta}
\end{aligned} \tag{5.3}$$

where Θ is the pitch angle, Φ is the roll angle, and Ψ is the yaw angle.

These nonlinear equations are linearized assuming the following:

- The aircraft is a rigid body,
- The aircraft mass is constant,
- The aircraft is trimmed to an equilibrium condition such that all accelerations are zero,
- The aircraft is in straight and level flight,
- The aircraft dynamics can be decoupled into longitudinal and lateral/directional components,
- The linear and rotational velocities, and the external forces and moments can be considered perturbed from equilibrium values,

- The product of small perturbed values are small and negligible, and
- The angles between the the equilibrium and the disturbed value are considered small.

Additionally, since the short period mode dominates the aircraft response to pilot inputs, the longitudinal equations are simplified as shown in Eq (5.4). Further explanation of these assumptions and simplifications can be found in [4, 8].

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \end{bmatrix} = \begin{bmatrix} Z_\alpha & Z_q \\ M_\alpha & M_q \end{bmatrix} \begin{bmatrix} \alpha \\ q \end{bmatrix} + \begin{bmatrix} Z_{\delta_e} & Z_{\delta_{PTV}} \\ M_{\delta_e} & M_{\delta_{PTV}} \end{bmatrix} \begin{bmatrix} \delta_e \\ \delta_{PTV} \end{bmatrix} \tag{5.4}$$

where α is the angle of attack, q is the linearized pitch rate, δ_e is the elevator deflection, δ_{PTV} is the pitch thrust vectoring, Z and M are longitudinal stability derivatives.

Through the use of a nonlinear control selector, developed in [4], the thrust vectoring commands are only used when the aerodynamic forces are not sufficient to complete the commanded maneuver. Furthermore, the use of the control selector enables the commanded inputs to be transformed into generalized command inputs: pitch acceleration, $\dot{p}_c$, yaw acceleration, $\dot{q}_c$, and roll acceleration, $\dot{r}_c$.

From this mapping, the longitudinal linear design model is that shown in Eq (5.5).

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \end{bmatrix} = A_{long} \begin{bmatrix} \alpha \\ q \end{bmatrix} + B_{long} \dot{q}_c$$

$$\begin{bmatrix} \alpha \\ q \end{bmatrix} = C_{long} \begin{bmatrix} \alpha \\ q \end{bmatrix} \quad G_{long} \equiv \left[\begin{array}{c|c} A_{long} & B_{long} \\ \hline C_{long} & 0 \end{array}\right] \tag{5.5}$$

$$\text{where } B_{long} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \; C_{long} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.6}$$

where A_{long} is the longitudinal plant matrix, B_{long} is the longitudinal input matrix, and C_{long} is the longitudinal output matrix.

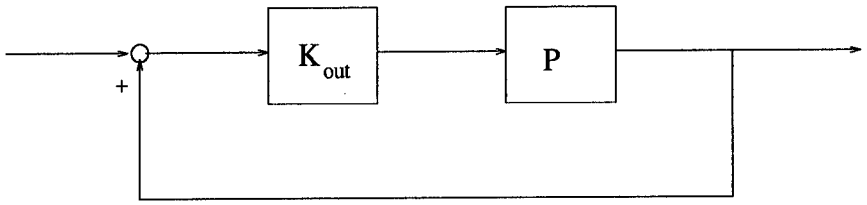


Figure 5.2 Outer Control Loop

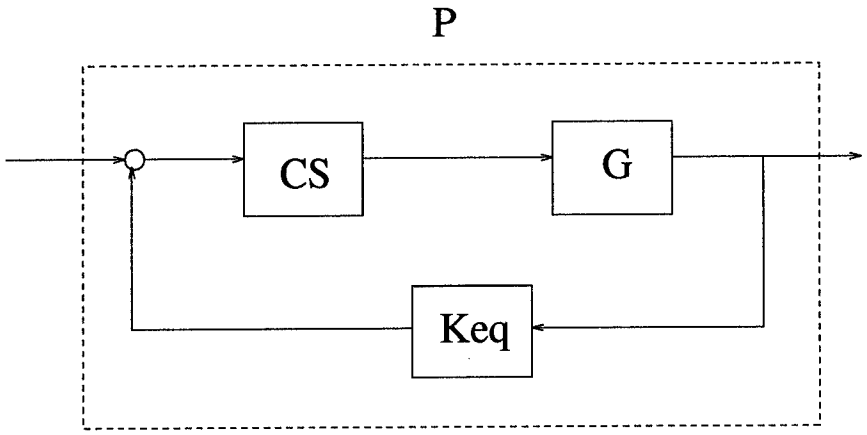


Figure 5.3 Inner Equalization Loop

5.2 Controller Design

The control of the F-18 longitudinal dynamics is decomposed into an inner and outer loop (See Fig. 5.3 and Fig. 5.2). The objective of the inner loop is to equalize the closed loop frequency response over the flight envelope so that the outer loop controller does not need to be scheduled. Therefore, this investigation focuses on the inner loop controller. The original design selected twelve flight conditions, shown in Fig. 5.4, at which to design an inner loop controller. At each design point a minimal H_∞ controller was designed [52]. Then, using engineering judgement, a central controller was selected as Mach 0.95 and an altitude of 20000 feet. The closed inner loop of this central controller, P_0, is used to design the outer loop controller, K_{out}.

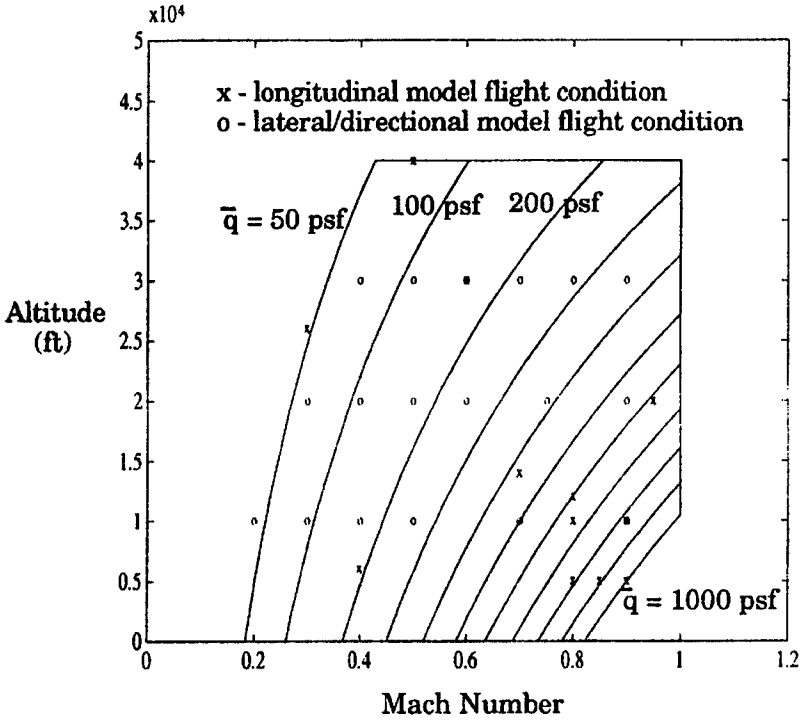


Figure 5.4 F-18 Flight Envelope

The inner loop controller, shown in Fig. 5.5, has three parameters that are scheduled with dynamic pressure, $\bar{q}$.

$$K_{eq} = \left[\begin{array}{c|c} F - GN & K_f - GM \\ \hline -N & -M \end{array}\right]$$

$$\text{where} \quad F = [-40] \qquad K_f = [\ 1 \quad 1\ ] \qquad G = [.0247]$$

$$N = N(\bar{q}) \qquad M = [\ M_1(\bar{q}) \quad M_2(\bar{q})\ ] \tag{5.7}$$

The parameters N and M are scheduled as a linear function of dynamic pressure. The schedules for these parameters are shown in Eq (5.8) and graphically in Fig. 5.6.

$$\begin{aligned} N(\bar{q}) &= -0.312\bar{q} + 461 \\ M_1(\bar{q}) &= -0.058\bar{q} + 50.5 \\ M_2(\bar{q}) &= -0.006\bar{q} + 8.11 \end{aligned} \tag{5.8}$$

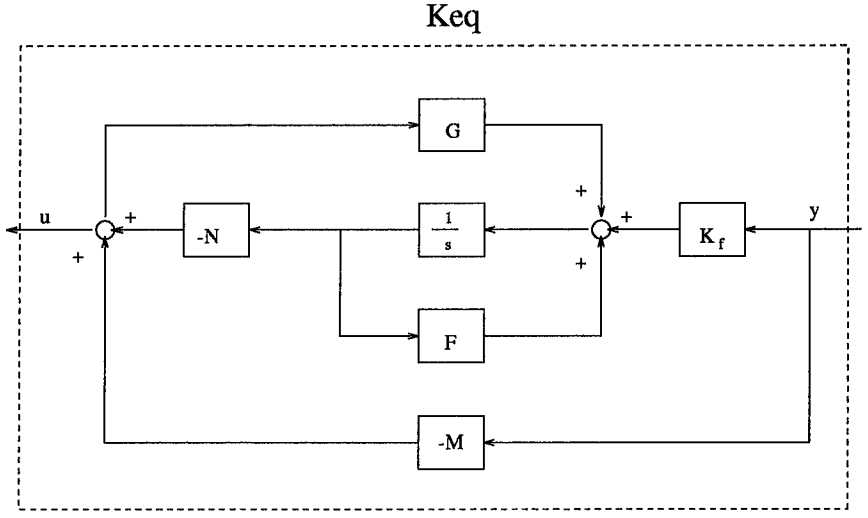


Figure 5.5 Inner Loop Controller

The schedules were developed by plotting the H_∞ scheduled controller parameters as a function of dynamic pressure. A linear fit of the parameters was performed using a least square error method. These controller schedules are the baseline design for comparison with the 'optimized' schedules. The goal of the optimization is to improve this schedule in terms of relative error, which is described in the next section.

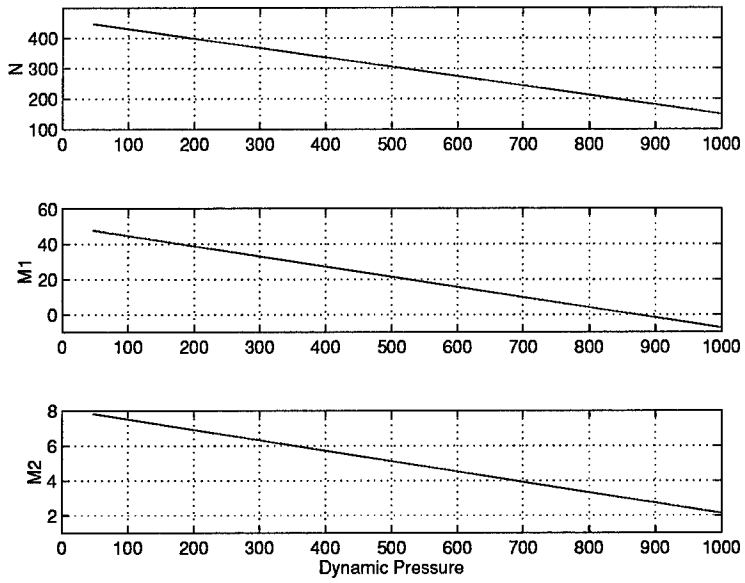


Figure 5.6 Baseline Schedule of Design Parameters

5.3 Relative Error

Relative error, defined by Eq (5.9), is used to measure the equalization of the closed inner-loop.

$$\tilde{\Delta}_m \doteq (P - P_0)P_0^{-1} \tag{5.9}$$

> P_0 is the closed inner loop used for outer loop controller design, and P is a closed inner loop at another flight condition.

Safonov and Chiang's *Robustness Theorem* [42] provides a weak sufficient condition for stability using relative error. The authors in [4] succinctly stated this sufficient condition.

> If $\bar{\sigma}(\tilde{\Delta}_m) < 1$ for $\omega < \omega_r$, then the closed loop system will be stable provided that the control bandwidth, ω_b, is less than ω_r.

Therefore, any outer loop controller designed for P_0 will be stable for any inner loop plant, P, provided the relative error, $\tilde{\Delta}_m$, is sufficiently small.

A summary of the proof was presented in [4] and is reproduced here for completeness.

Consider the system with equalized plant, P, outer loop controller, K_{ol}, and relative error, $\tilde{\Delta}_m$. The bandwidth of the control system is defined as the frequency range where the loop transfer function is *big*. That is:

$$\underline{\sigma}(PK_{ol}) \gg 1 \qquad \forall \omega < \omega_b \tag{5.10}$$

A sufficient condition for stability is

$$\bar{\sigma}(\tilde{\Delta}_m)\bar{\sigma}(PK_{ol}(I + PK_{ol})^{-1}) < 1 \tag{5.11}$$

It follows from Eq (5.11) that for $\omega < \omega_b$,

$$\bar{\sigma}(PK_{ol}(I + PK_{ol})^{-1}) \approx 1 \tag{5.12}$$

So, for frequencies where the loop transfer function gain is *big*, a sufficient condition for stability is

$$\bar{\sigma}(\tilde{\Delta}_m) < 1 \tag{5.13}$$

Relative error is a frequency domain measure of the differences between closed loop system responses at various flight conditions. The main goal of the inner-loop of this design is to achieve a relative error less than one for all flight conditions. The logical optimization objective is to minimize this error for all flight conditions and thereby achieve a more equalized inner-loop. Therefore, the objective function for this optimization is

$$\min J = \sum_{i=1}^{n} \bar{\sigma}(\tilde{\Delta}_{m_i}) \tag{5.14}$$

where $\tilde{\Delta}_m$ is defined in Eq (5.9), and n is the number of flight conditions to be evaluated. [1]

The flight conditions used in evaluating the objective function are the original twelve used for design in [4] plus an additional eight flight conditions chosen to more uniformly represent the

[1] For n flight conditions there are only $(n-1)$ nonzero relative errors.

flight envelope as a function of dynamic pressure. The original twelve flight conditions are shown in Fig. 5.4. All of these flight conditions and their plant matrices are listed in Appendix A. The goal of this optimization is that by reducing the relative error of the inner loop the time response of the system will become more uniform throughout the operating envelope.

5.4 Optimization Results

This section presents the optimization results of the F-18 longitudinal inner equalization loop controller. Multiple optimizations were performed with various alterations on the basic objective function given in Eq (5.14). The first section provides on overview of the optimization cases considered and the respective sections in which the results are presented. Next, section 5.4.11 compares and summarizes the optimization results, and section 5.4.12 demonstrates the responses of off design flight conditions using the optimized gain schedule.

5.4.1 Overview. The optimization process is performed with a family of twenty flight conditions representing the flight envelope shown in Fig. 5.7. The linearized plants are listed in Appendix A. The optimization results are based on four objective functions

$$\min J_1 = \sum_{i=1}^{20} \bar{\sigma}(\tilde{\Delta}_{m_i}) \tag{5.15}$$

Note: Since the relative error is a measure of relative difference, one of the relative error values is zero.

$$\min J_2 = \sum_{i=1}^{20} \bar{\sigma}(\tilde{\Delta}_{m_i}) + f(N) \tag{5.16}$$

where $f(N)$ is a functional weighting on the number of intervals.

$$\min J_3 = \sum_{i=1}^{20} \bar{\sigma}(\tilde{\Delta}_{m_i}) + e_i(t) \tag{5.17}$$

where $e_i(t)$ is the error between the time response of the chosen central flight condition and the time response of the flight condition i.

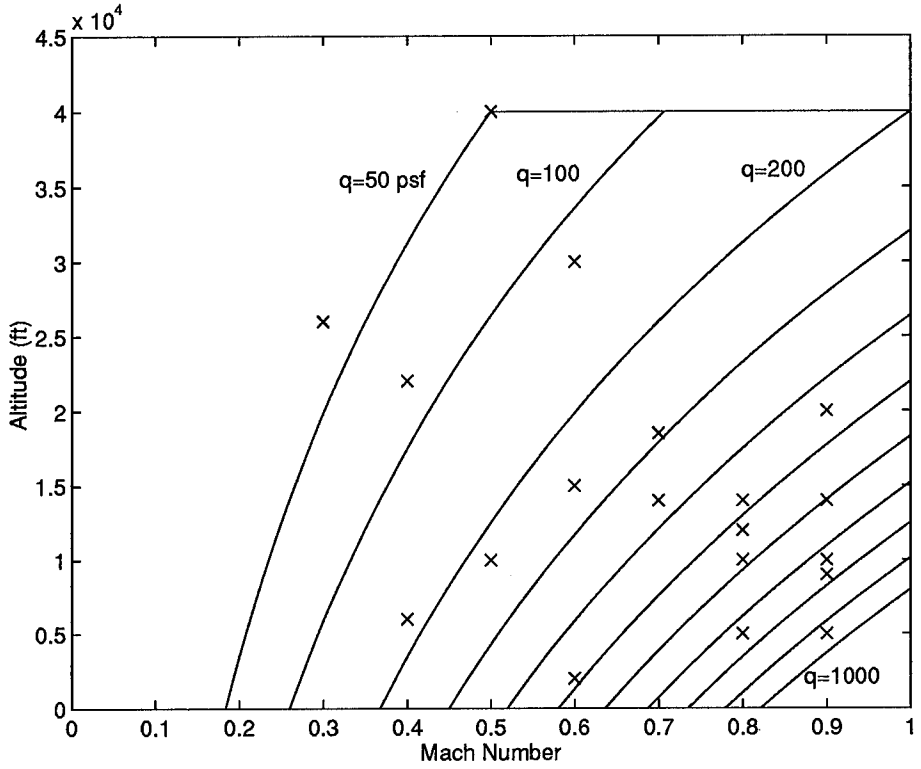


Figure 5.7 F-18 Flight Envelope

$$\max J_4 = \sum_{i=1}^{20} \bar{\sigma}(\tilde{\Delta}_{m_i}) + \frac{1}{5} e_i(t) \tag{5.18}$$

Eight optimization cases are considered and are summarized in Table 5.1. A brief explanation of each optimization is presented in the following list:

Case I The scheduled variables are piecewise constant functions of the scheduling variable. Four intervals of the scheduling variable were arbitrary chosen to be of equal size: $(0, 250], (250, 500], (500, 750], (750, 1000)$. J_1 is the objective function. The central flight condition, P_0, was chosen as Mach 0.95 and an altitude of 20,000 feet [4].

Case II This case is the same as Case I except that the size of the interval is allowed to vary to reduce the objective function further.

Case III This case is the same as Case II except that the number of intervals is also a design variable. J_2 is the objective function.

Case	Objective Function	Results Section
0	Baseline	5.4.2
I	J_1	5.4.3
II	J_1	5.4.4
III	J_2	5.4.5
IV	J_2	5.4.6
V	J_3	5.4.7
VI	J_1	5.4.8
VII	J_3	5.4.9
VIII	J_4	5.4.10

Table 5.1 Optimization Summary

Case IV This case is the same as Case III except the central flight condition is allowed to vary within the family of plants to further reduce the objective function, J_2.

Case V This case uses objective function J_3. The central flight condition, the number of intervals, and the size of each interval are allowed to vary.

Case VI For this case the scheduling variables are piecewise linear functions of dynamic pressure. The objective function of the optimization is J_1. The central flight condition, number of intervals, and the size of the intervals are allowed to vary.

Case VII This case is the same as Case VI except that the objective function is J_3.

Case VIII This case is the same as Case VI except that the objective function is J_4.

5.4.2 Baseline Design Results. The baseline design developed in [4], was not optimized. Relative error was only used as a weak sufficient condition for stability of the controller throughout the flight envelope. The scheduling variables were chosen as described in section 5.2. The open loop frequency responses of the inner loop are shown in Fig. 5.8 and Fig. 5.9. The flight condition with the lowest dynamic pressure has the highest low frequency gain of all the flight conditions. The general trend is that as the dynamic pressure increases the low frequency gain decreases.

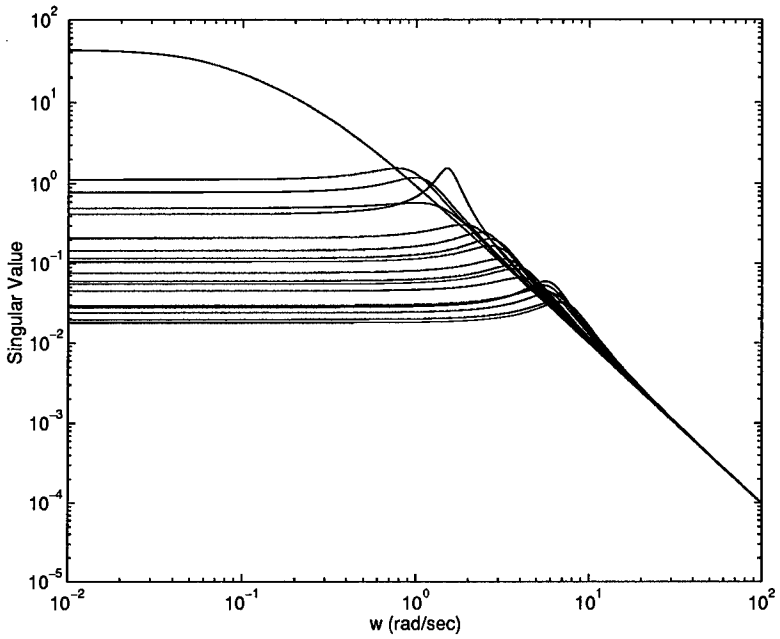


Figure 5.8 Case 0: $\dot{q}_c$ to α Open-Loop Dynamics

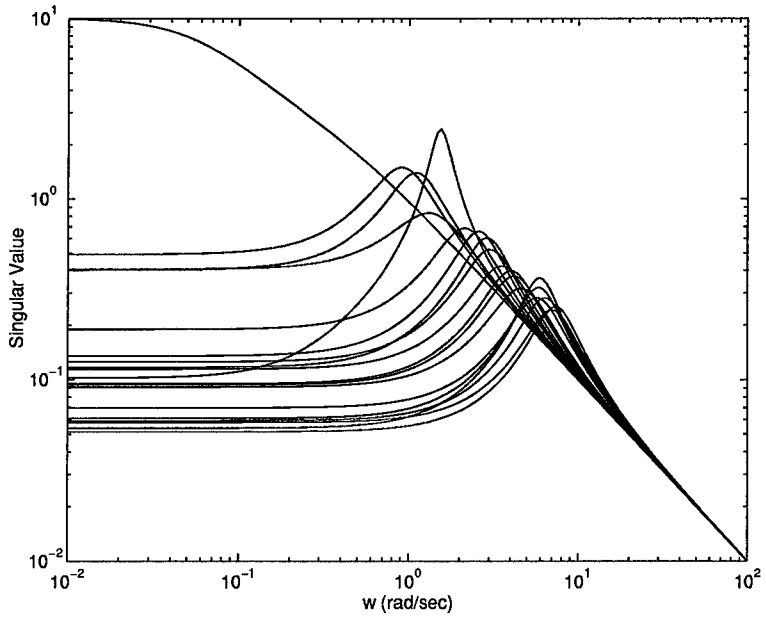


Figure 5.9 Case 0: $\dot{q}_c$ to q Open-Loop Dynamics

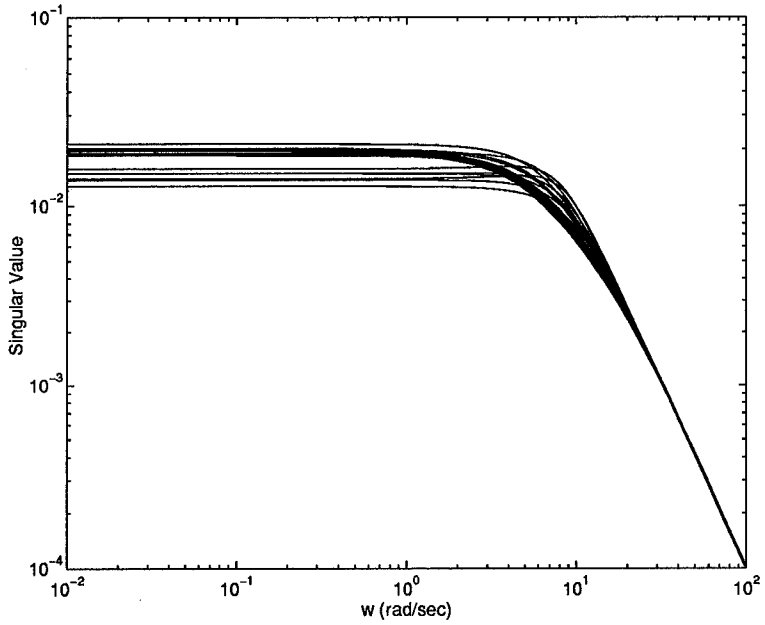


Figure 5.10 Case 0: $\dot{q}_c$ to α Closed-Loop Dynamics

The closed loop dynamics of the inner loop are shown in Fig. 5.10 and Fig. 5.11. The two most important graphs for demonstrating the optimization results are Fig. 5.12 and Fig. 5.13. These graphs depict the relative error of the closed inner equalization loop and the time response to a step pitch acceleration command of the closed outer loop, respectively. Note that the time response has two distinct groupings. This is because there are actually two outer-loop controllers, one for low dynamic pressure ($\bar{q} < 200$ psf) and one for high dynamic pressure ($\bar{q} < 200$ psf). This is a consequence of two desired pitch responses in different regions of the flight envelope. At higher velocities, the pilot likes to feel a faster time response. Also note that the relative error for most flight conditions has a maximum singular value around 0.4 to 0.5.

The baseline schedules of the parameters N and M are shown in Fig. 5.14. The five graphs, Fig. 5.10, Fig. 5.11, Fig. 5.12, Fig. 5.13, and Fig. 5.14 are the baseline for comparison with the optimization results.

The values of the objective functions are $J_1 = 6.37$, $J_3 = 23.24$, and $J_4 = 9.74$.

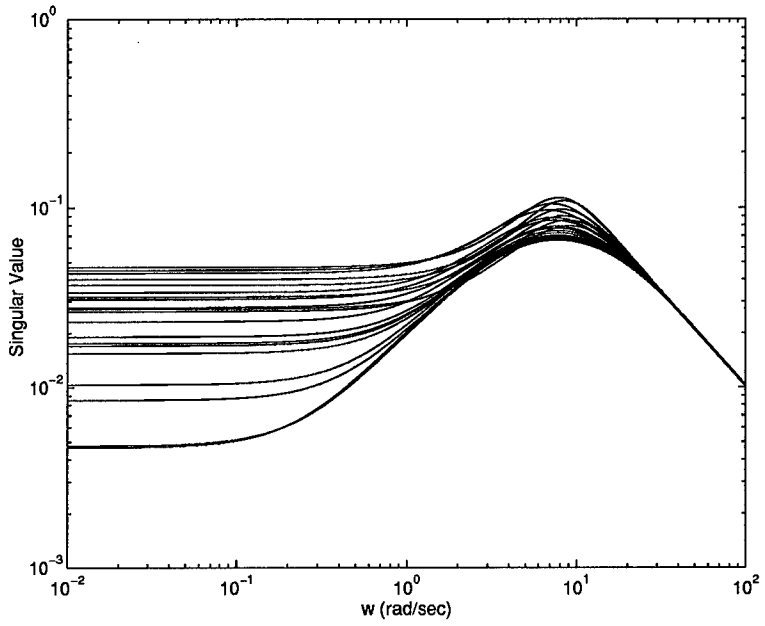


Figure 5.11 Case 0: $\dot{q}_c$ to q Closed-Loop Dynamics

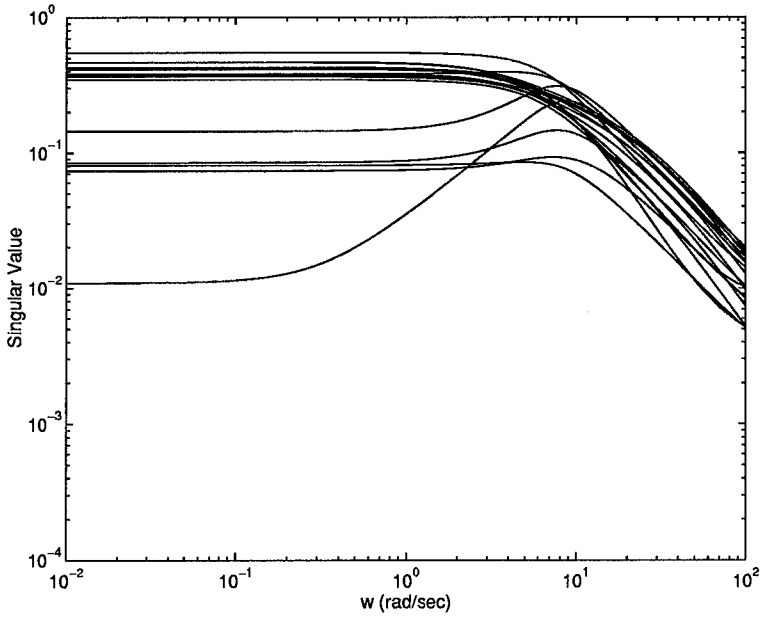


Figure 5.12 Case 0: Relative Error

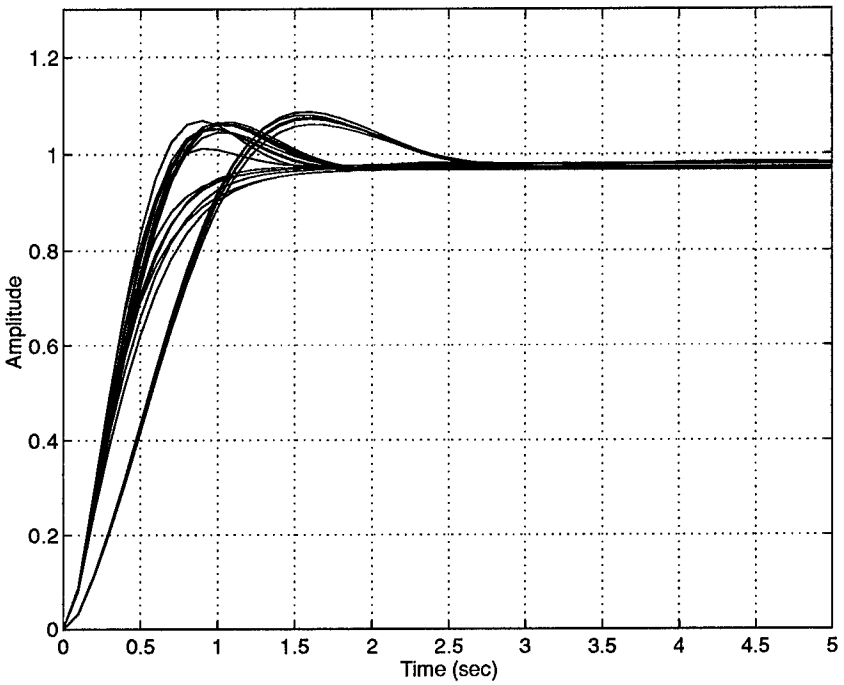


Figure 5.13 Case 0: Time Response for a Step Input

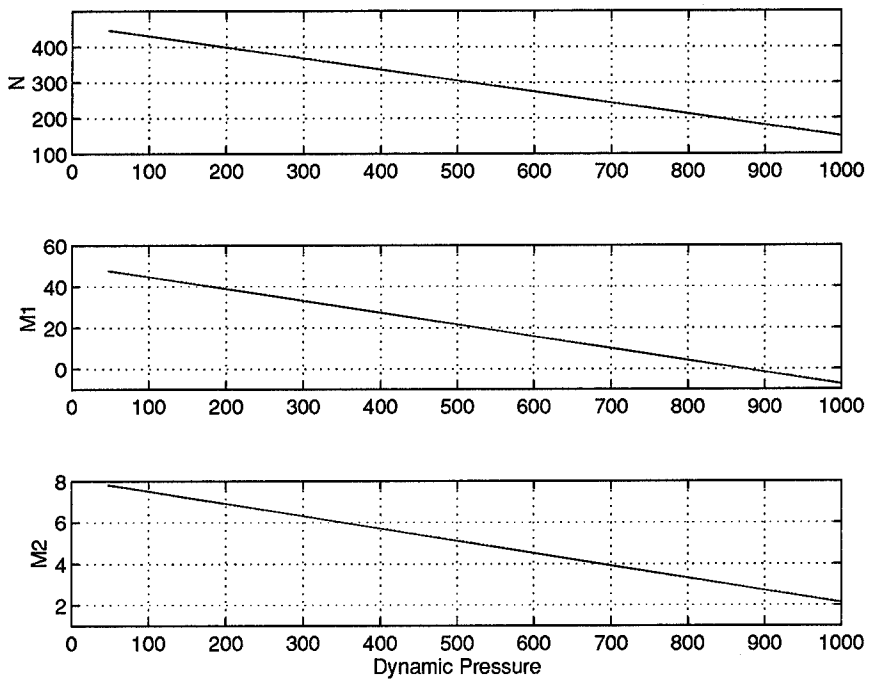


Figure 5.14 Case 0: Controller Parameter Schedule

5.4.3 Case I. The design variables of the GA optimization are the scheduled parameters defined in Eq (5.8). The schedules are evaluated at $\bar{q}_{\min} = 0$ and $\bar{q}_{\max} = 1000$ to obtain the minimum and maximum values of the parameters N and M. For a first look at optimizing these parameters, the range of the scheduling variable, $\bar{q}$, is arbitrarily divided into four intervals, (0,250], (250,500], (500,750], and (750,1000). The scheduled parameters are chosen to be piecewise constant values of dynamic pressure in each interval. The parameters N and M are optimized for each interval to minimize objective function J_1. There are twelve design variables to optimize.

The optimized objective function value is 3.09, half of the baseline relative error. The closed inner-loop responses are shown in Fig. 5.15 and Fig. 5.16. The relative error and time responses are shown in Fig. 5.17 and Fig. 5.18. The resulting scheduled variables are shown in Fig. 5.19.

As a result of the reduction in relative error, the overshoot in the time response has decreased for most of the flight conditions. However, for three flight conditions the overshoot increased over the baseline. This can also be seen in the relative error graph, Fig. 5.17, where three relative errors are increased over the baseline and the remaining relative errors are reduced. Additionally, the closed inner-loop pitch response is more uniform than the baseline design.

5.4.4 Case II. For this optimization the interval size is allowed to vary. The number of intervals is still constrained to be four. Now the number of design variables is fifteen, three interval endpoints and twelve scheduled variable values. Each interval endpoint is allowed to vary between 0 and 1000 psf.

The objective function value is 2.51 which is less than Case I. The closed inner-loop responses are shown in Fig. 5.20 and Fig. 5.21. The relative error and time responses are shown in Fig. 5.22 and Fig. 5.23. The resulting scheduled variables are shown in Fig. 5.24.

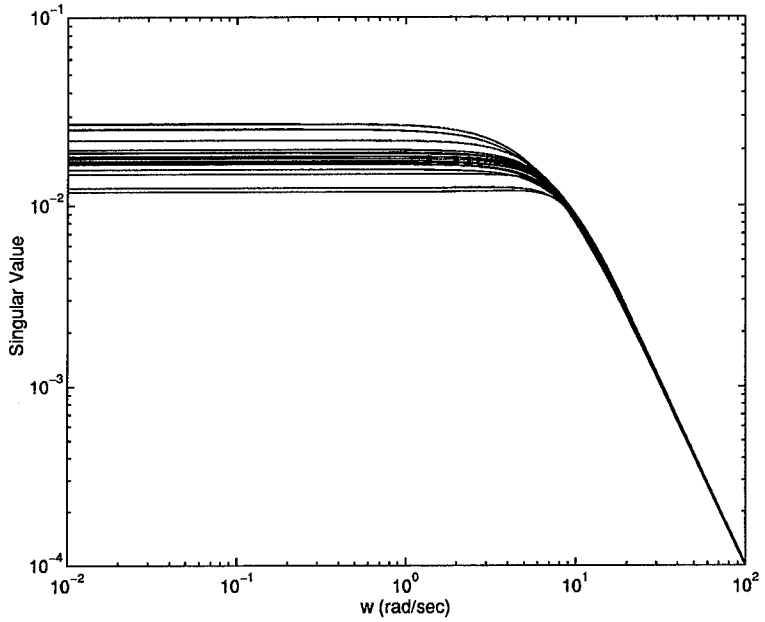


Figure 5.15 Case I: $\dot{q}_c$ to α Closed-Loop Dynamics

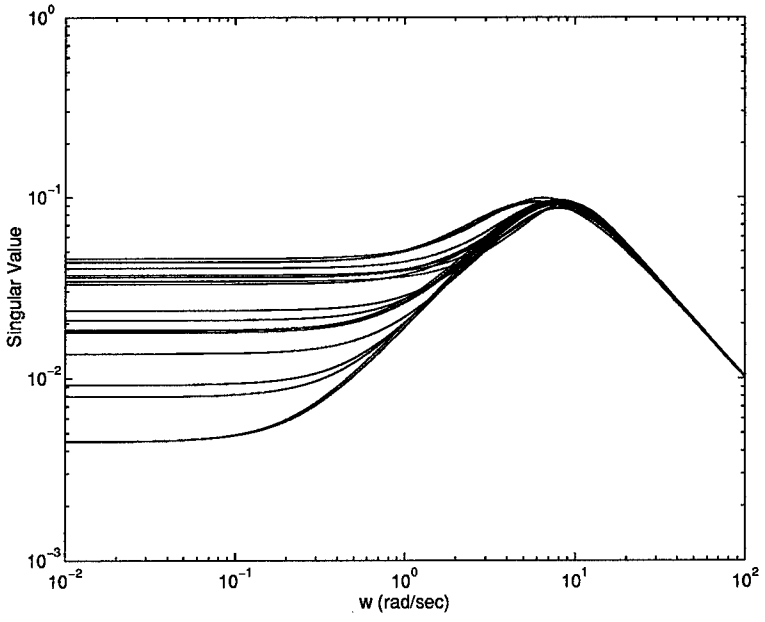


Figure 5.16 Case I: $\dot{q}_c$ to q Closed-Loop Dynamics

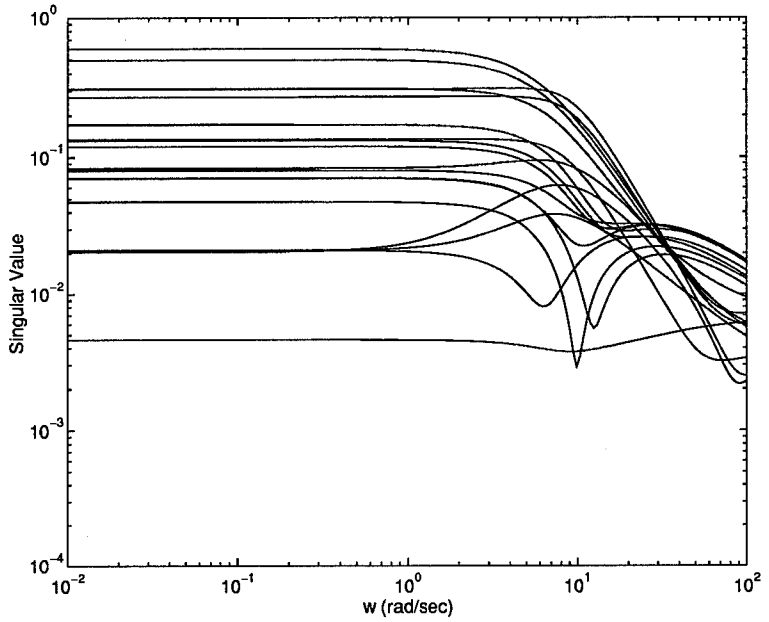


Figure 5.17 Case I: Relative Error

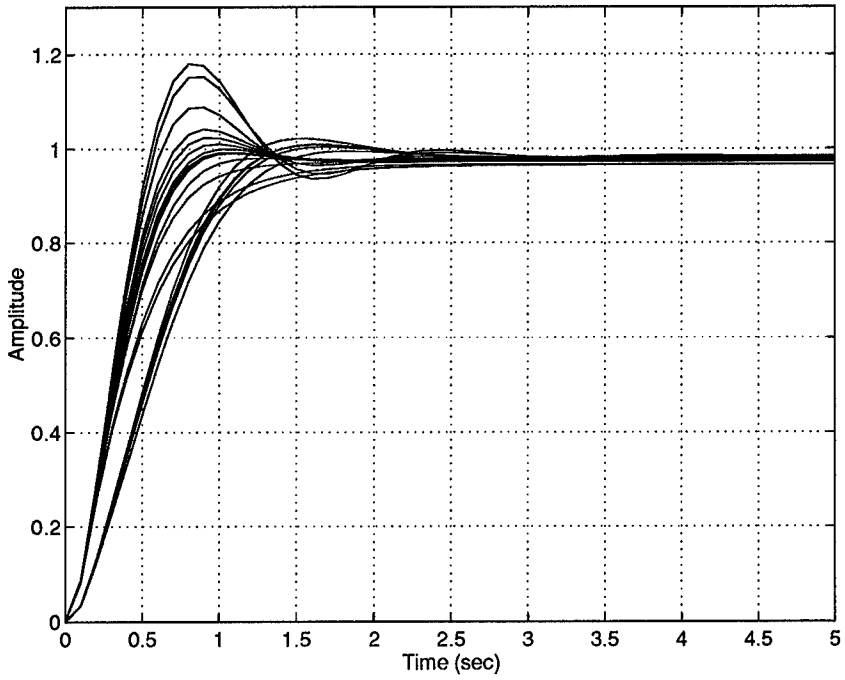


Figure 5.18 Case I: Time Response for a Step Input

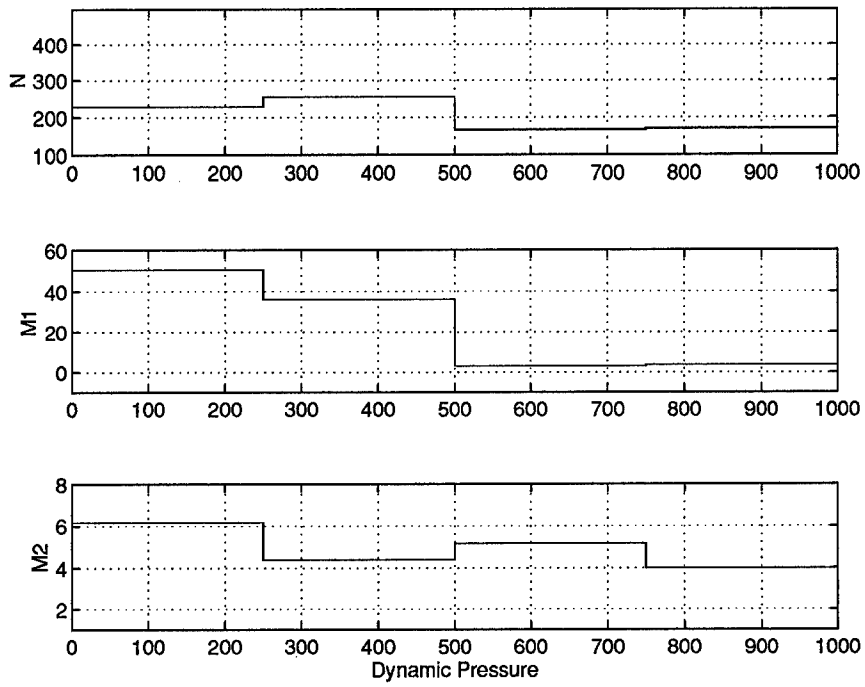


Figure 5.19 Case I: Controller Parameter Schedule

As in Case I, the time response of two flight conditions have overshoots larger than the baseline. The significance of this optimization is that the relative error is decreased by allowing the interval size to vary.

5.4.5 Case III. For this optimization the number of intervals is allowed to vary between (2..9). Additionally, the interval endpoints are allowed to vary between (0..1000). Therefore the number of design variables varies between (8..36). Four different penalty functions on the number of intervals are optimized.

a) $f_1(N) = N$
b) $f_2(N) = N^2$
c) $f_3(N) = N/9$
d) $f_4(N) = (N/9)^2$

The results of each of these optimizations are in the following subsections.

5.4.5.1 Case IIIa. Two intervals are chosen as with an objective function value of 5.24. The closed inner-loop responses are shown in Fig. 5.25 and Fig. 5.26. The

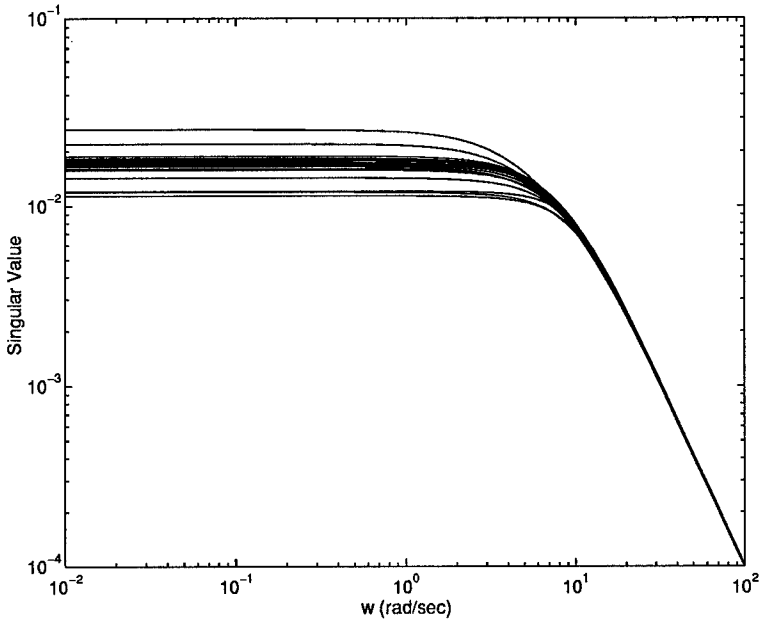


Figure 5.20 Case II: $\dot{q}_c$ to α Closed-Loop Dynamics

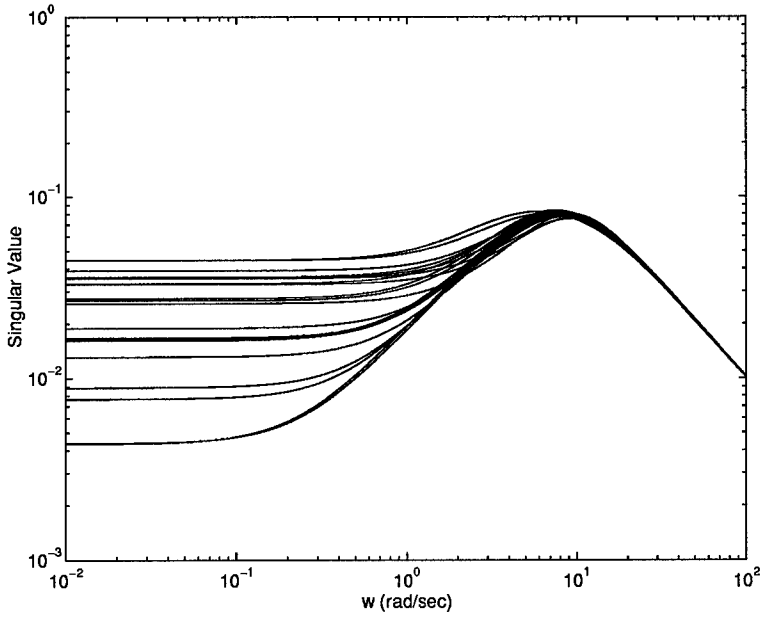


Figure 5.21 Case II: $\dot{q}_c$ to q Closed-Loop Dynamics

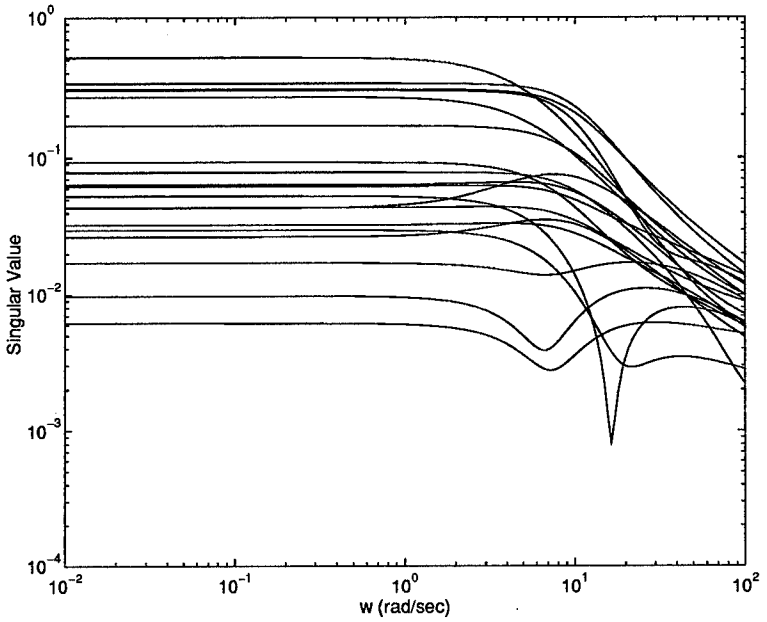


Figure 5.22 Case II: Relative Error

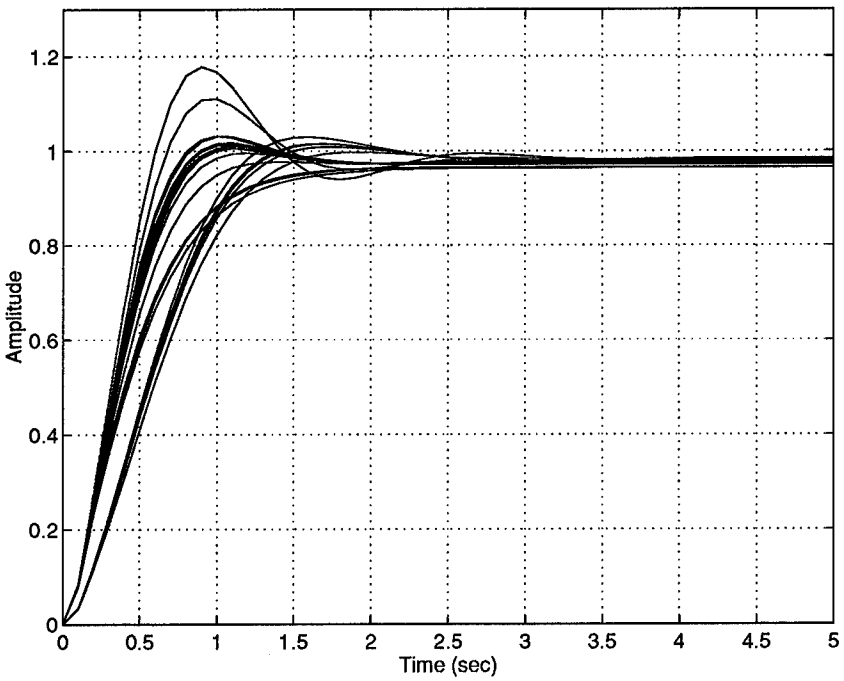


Figure 5.23 Case II: Time Response for a Step Input

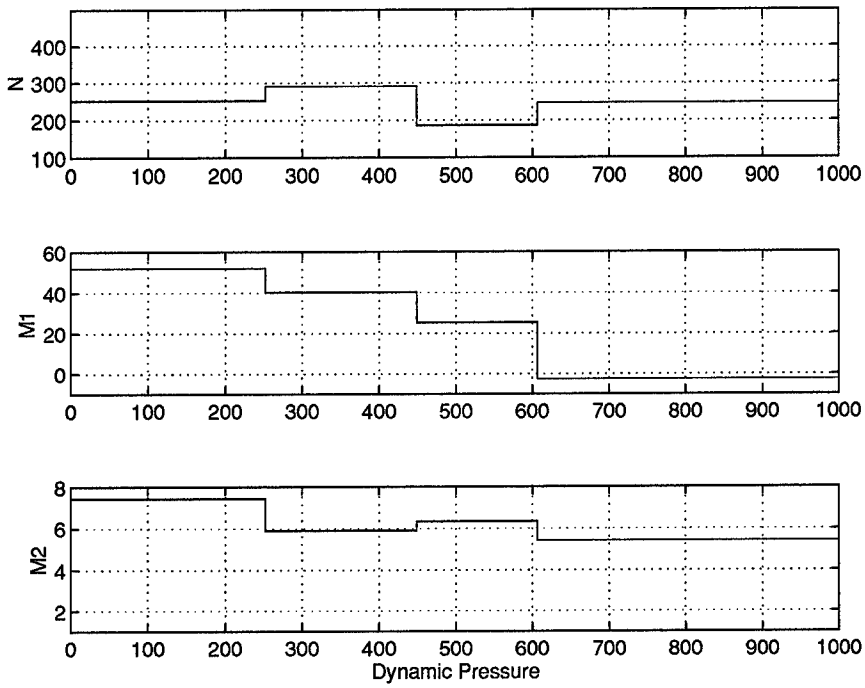


Figure 5.24 Case II: Controller Parameter Schedule

relative error and time responses are shown in Fig. 5.27 and Fig. 5.28 and the resulting scheduled variables are shown in Fig. 5.29.

The time responses show a significant improvement in overshoot when compared with Cases I and II. However, there are still three time responses with overshoots larger than the baseline. The rise time and settling time are consistent with variations in the flight condition. The relative error is about the same value as Case I when the penalty function value is subtracted off. The significant finding is that the same minimization of relative error can be achieved with half the intervals. Therefore, the same improvements in the equalization of the inner-loop can be achieved with half the complexity.

5.4.5.2 Case IIIb. Two intervals are again chosen as optimal with an objective function value of 7.23. The closed inner-loop responses are shown in Fig. 5.30 and Fig. 5.31. The relative error and time responses are shown in Fig. 5.32 and Fig. 5.33 and the resulting scheduled variables are shown in Fig. 5.34.

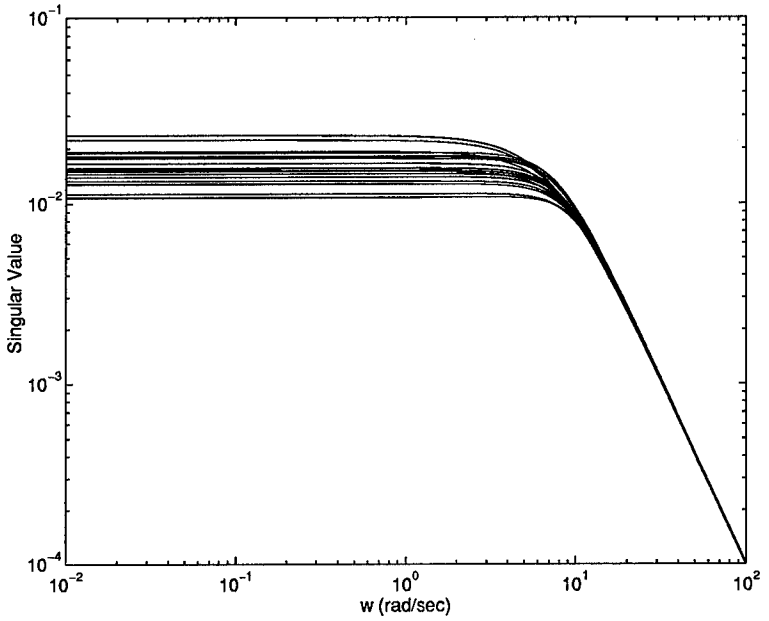


Figure 5.25 Case IIIa: $\dot{q}_c$ to α Closed-Loop Dynamics

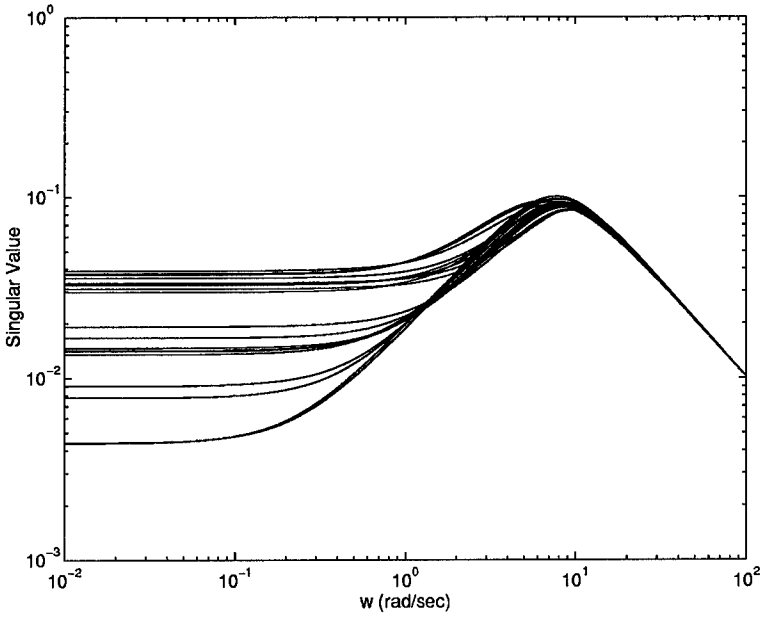


Figure 5.26 Case IIIa: $\dot{q}_c$ to q Closed-Loop Dynamics

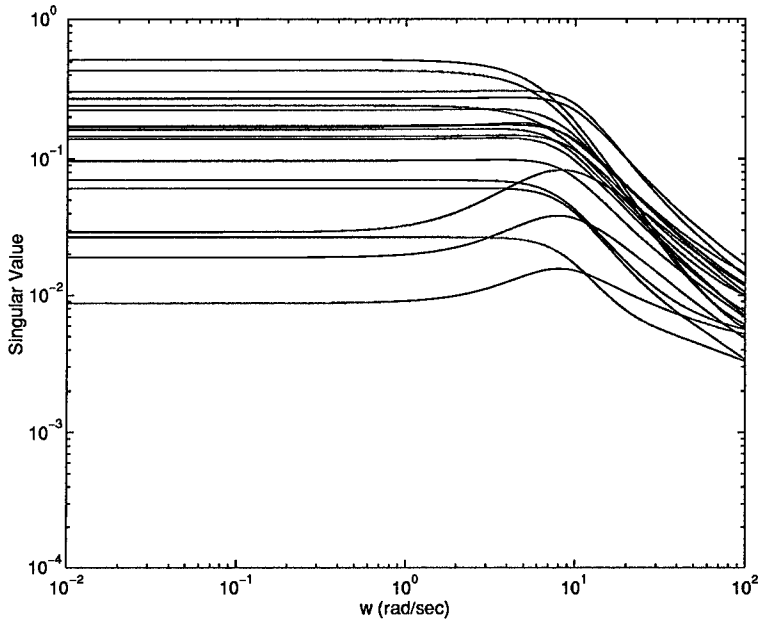


Figure 5.27 Case IIIa: Relative Error

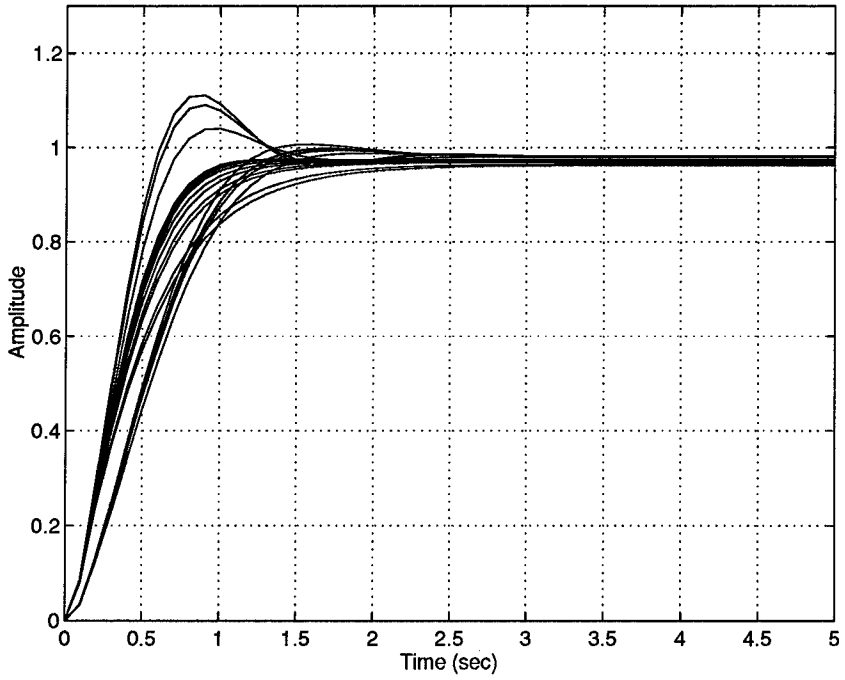


Figure 5.28 Case IIIa: Time Response for a Step Input

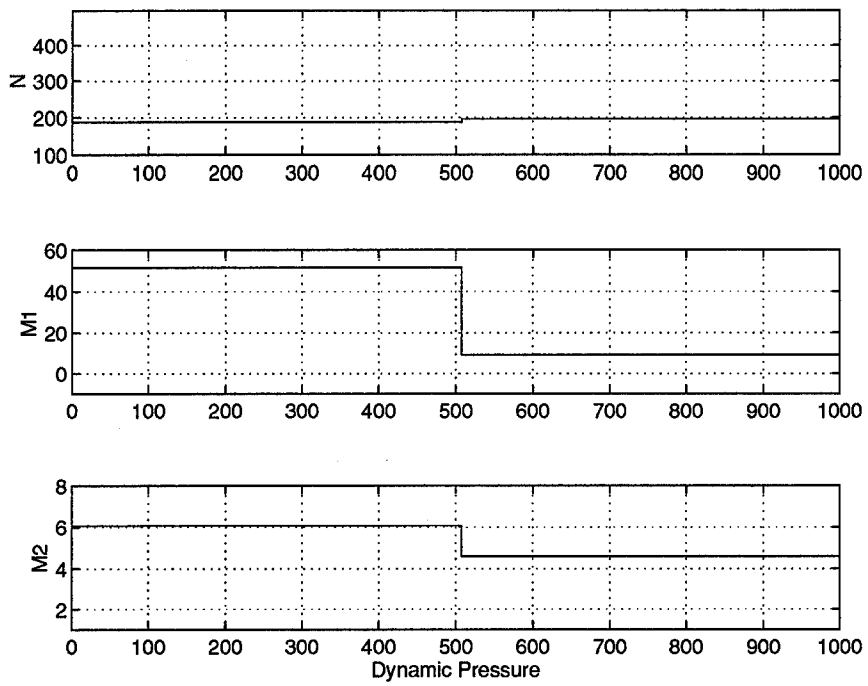


Figure 5.29 Case IIIa: Controller Parameter Schedule

There is little to no difference in the closed loop responses, the relative error, and the time responses between this case and Case IIIa. The only difference is a small change in the scheduled parameter N. The parameters M1 and M2 are even the same for these two cases. This seems to indicate that the parameter N is not very significant in the scheduling process.

5.4.5.3 Case IIIc. Two intervals are again chosen with an objective function value of 3.45. The closed inner-loop responses are shown in Fig. 5.35 and Fig. 5.36. The relative error and time responses are shown in Fig. 5.37 and Fig. 5.38 and the resulting scheduled variables are shown in Fig. 5.39.

For this case the closed loop responses, the relative error, the time responses, and the scheduling variables N, M1 and M2 all show the same trends as Case IIIa. Again, N is slightly different.

5.4.5.4 Case IIId. Here, three intervals are chosen with and objective function value of 3.24. Since the penalty for the number of intervals is the least for this optimization, a

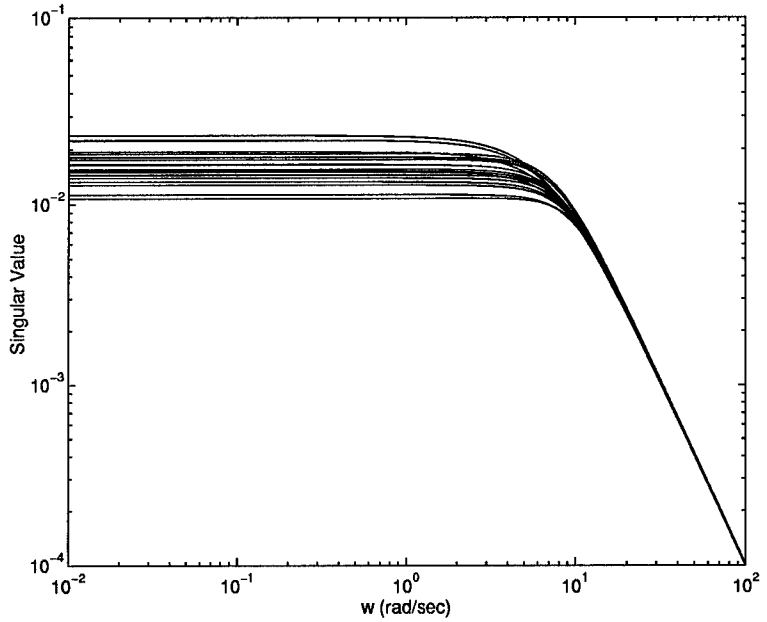


Figure 5.30 Case IIIb: $\dot{q}_c$ to α Closed-Loop Dynamics

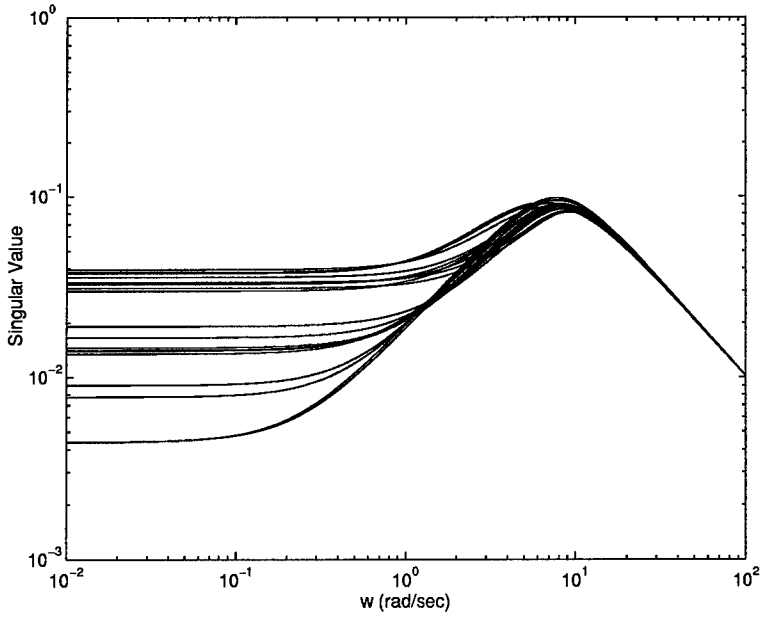


Figure 5.31 Case IIIb: $\dot{q}_c$ to q Closed-Loop Dynamics

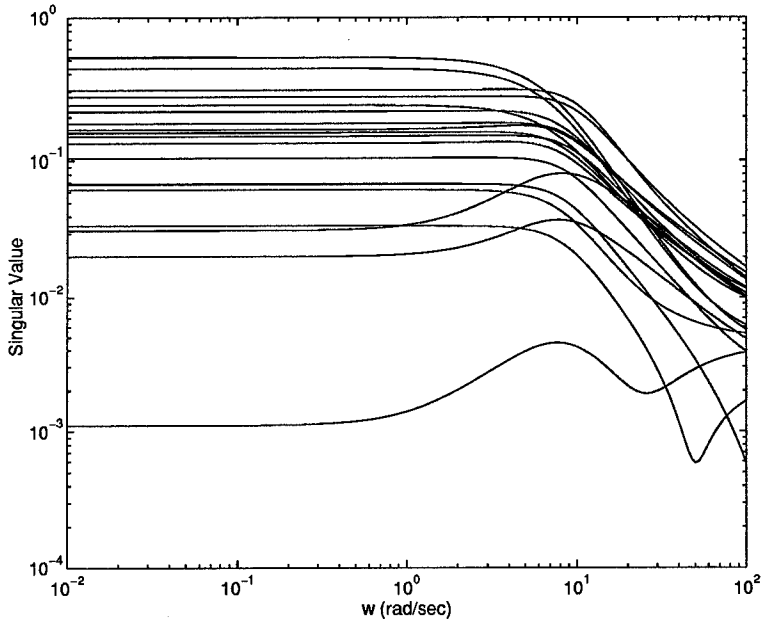


Figure 5.32 Case IIIb: Relative Error

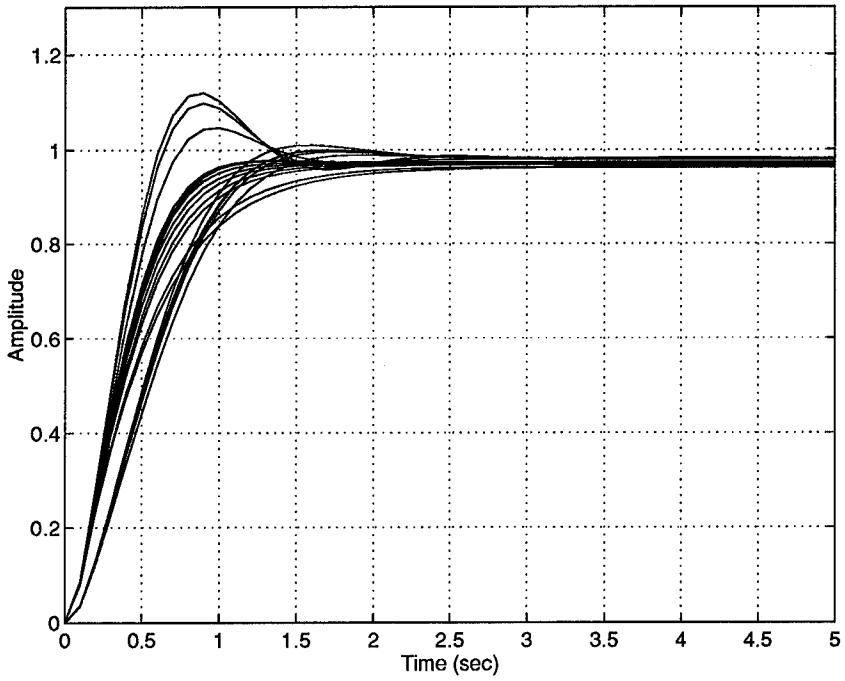


Figure 5.33 Case IIIb: Time Response for a Step Input

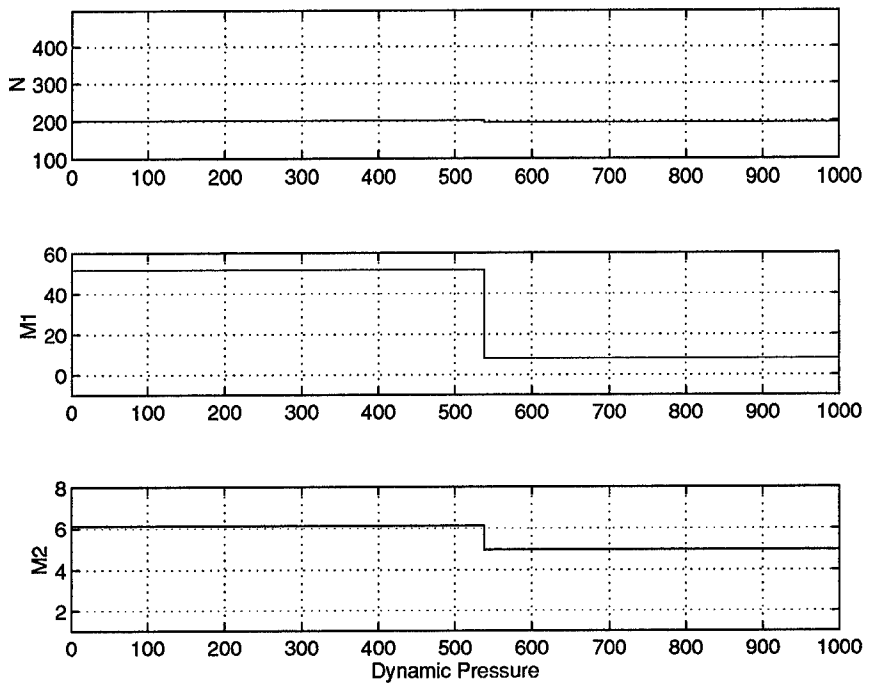


Figure 5.34 Case IIIb: Controller Parameter Schedule

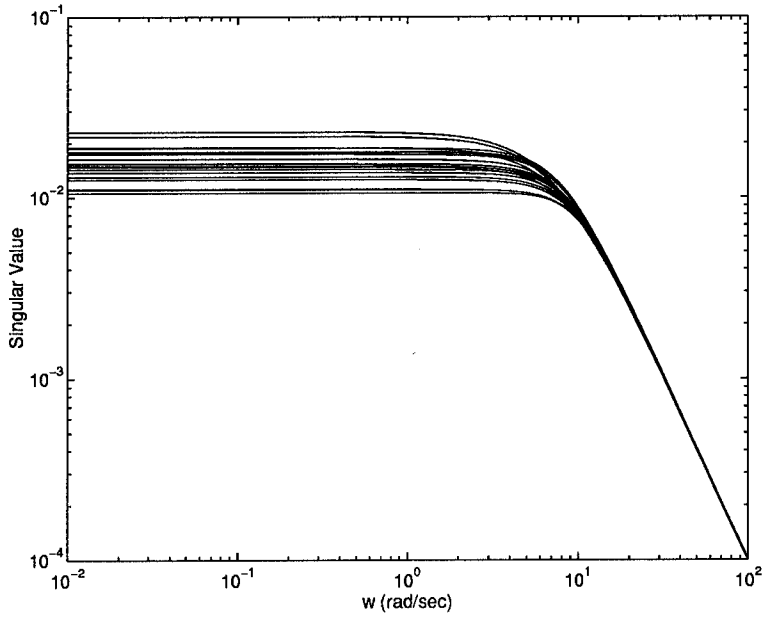


Figure 5.35 Case IIIc: $\dot{q}_c$ to α Closed-Loop Dynamics

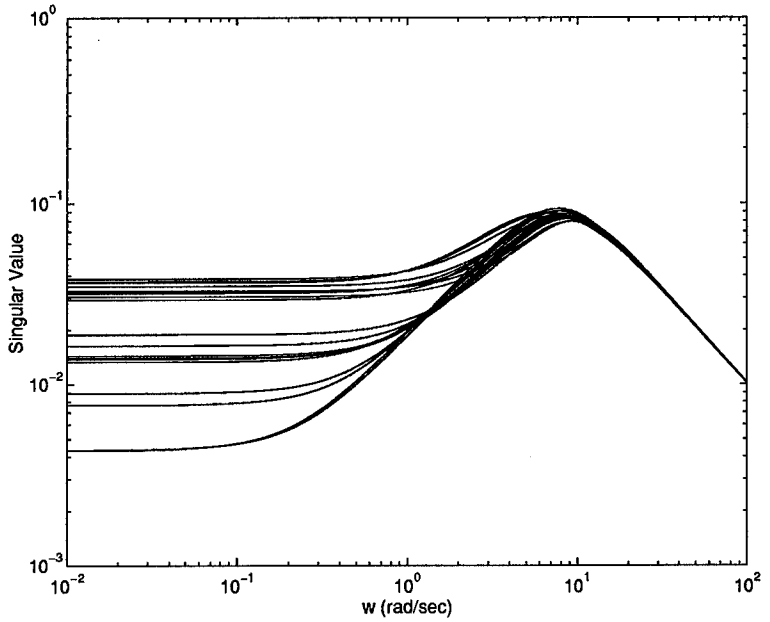


Figure 5.36 Case IIIc: $\dot{q}_c$ to q Closed-Loop Dynamics

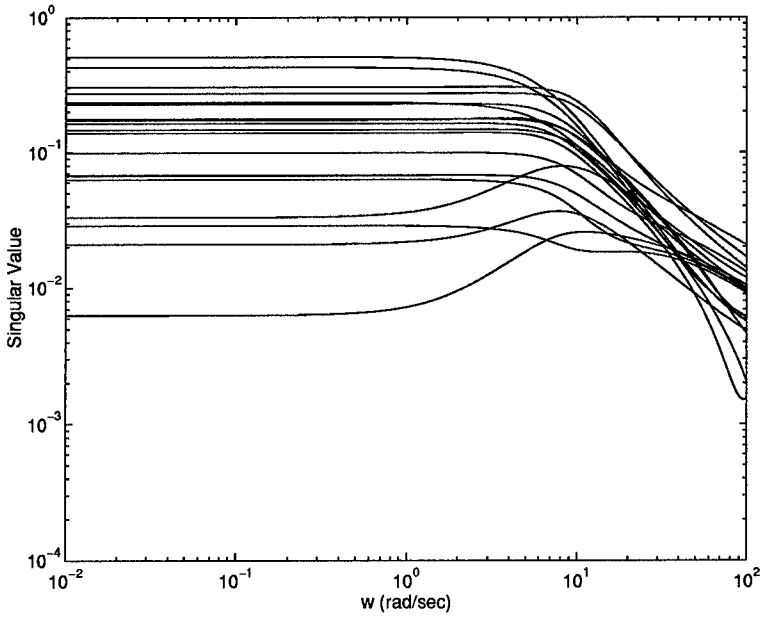


Figure 5.37 Case IIIc: Relative Error

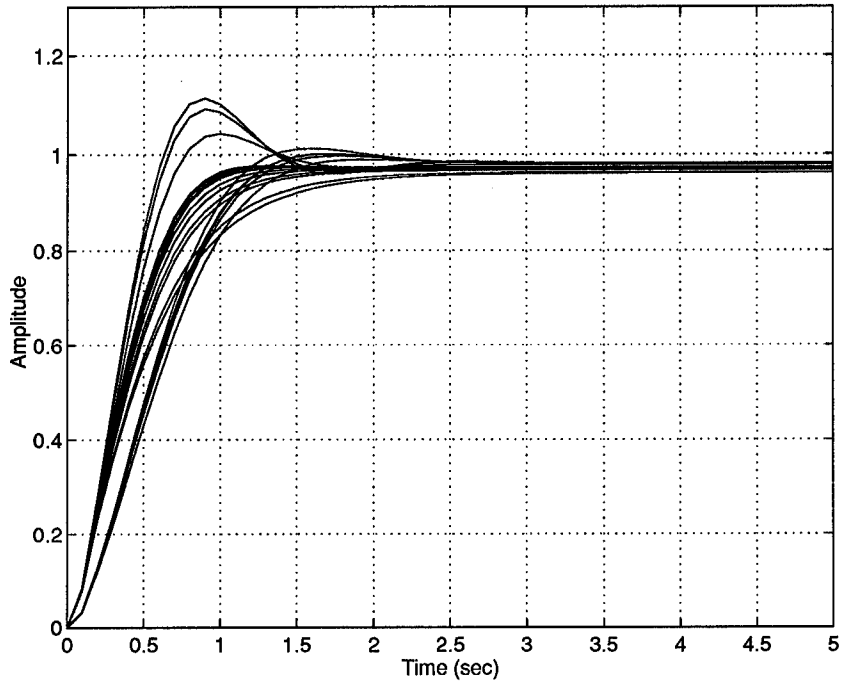


Figure 5.38 Case IIIc: Time Response for a Step Input

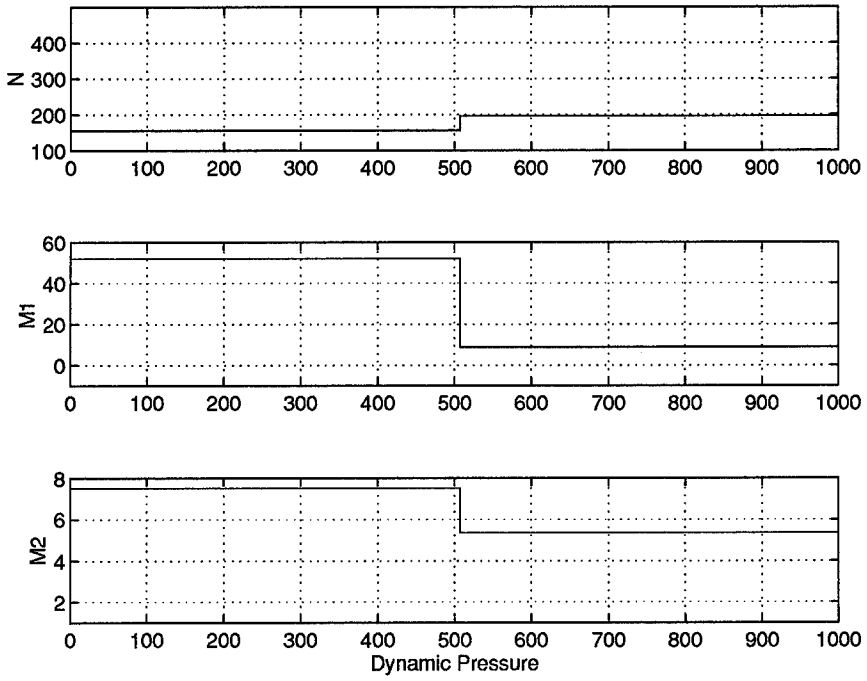


Figure 5.39 Case IIIc: Controller Parameter Schedule

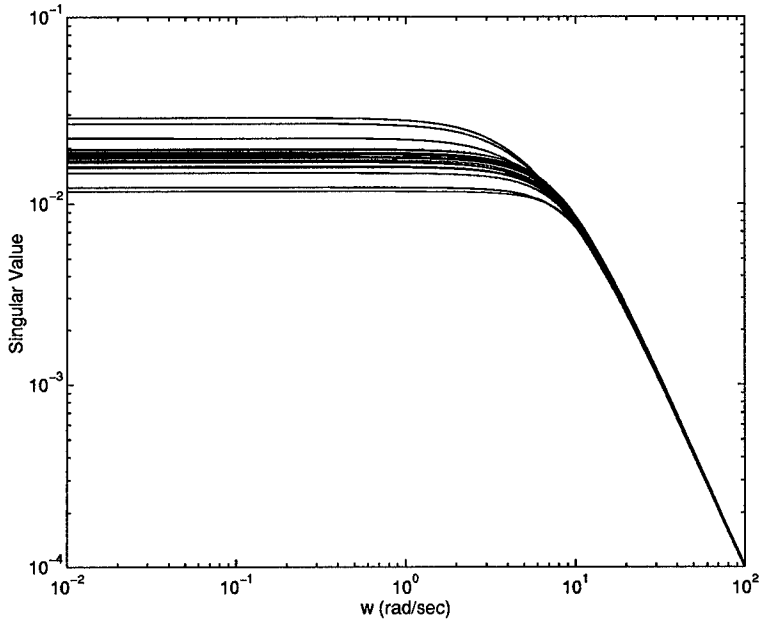


Figure 5.40 Case IIId: $\dot{q}_c$ to α Closed-Loop Dynamics

larger number of intervals is selected. The closed inner-loop responses are shown in Fig. 5.40 and Fig. 5.41. The relative error and time responses are shown in Fig. 5.42 and Fig. 5.43 and the resulting scheduled variables are shown in Fig. 5.44.

The only significant difference between this case and Cases IIIa, b, and c is the time response. The time response resulting from this optimization is comparable to the time response of Case I. The time responses for this case are slightly worse than Cases IIIa, b, and c.

5.4.6 Case IV. For this optimization, the central flight condition which was chosen by engineering judgement for the baseline design is allowed to vary. The number of design variables is increased by one; the number of design variable now varies between 9 and 37. Additionally, two different penalty functions on the number of intervals are optimized.

a) $f_1(N) = 0$
b) $f_2(N) = N/100$

The results of each optimization are in the following subsections.

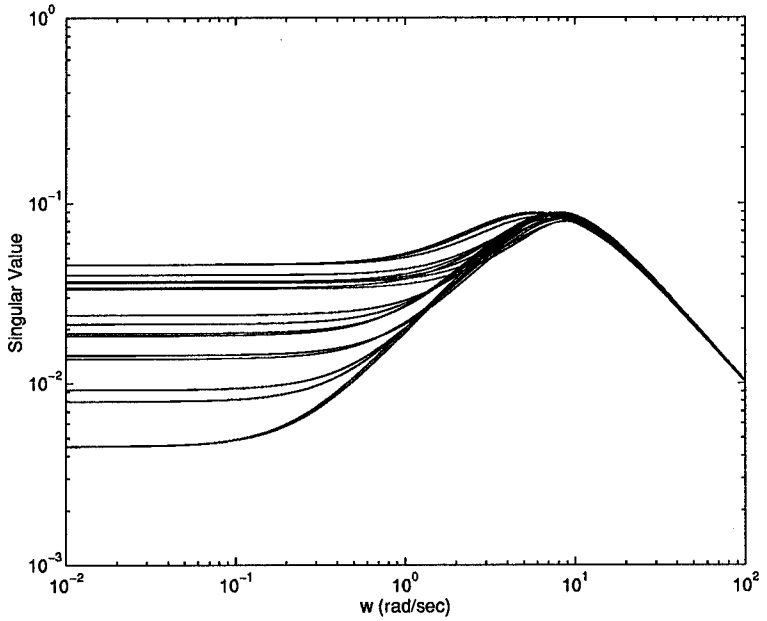


Figure 5.41 Case IIId: $\dot{q}_c$ to q Closed-Loop Dynamics

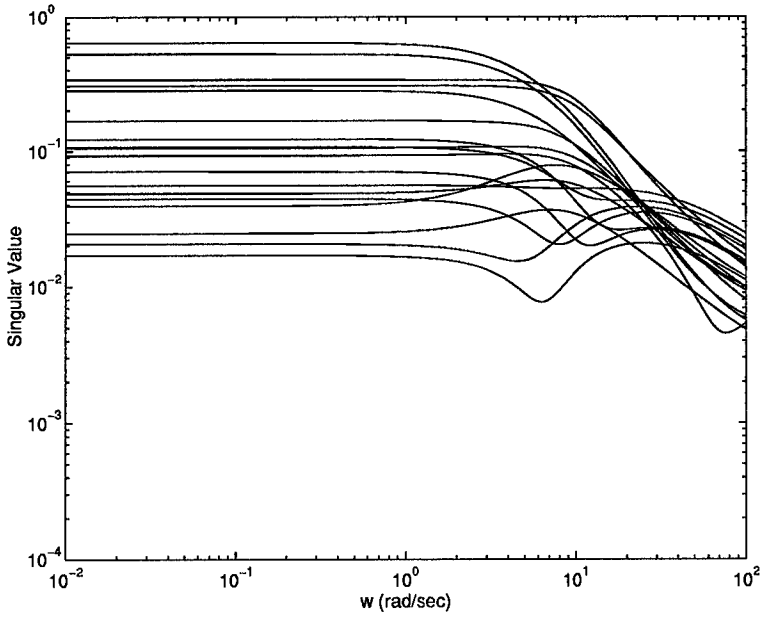


Figure 5.42 Case IIId: Relative Error

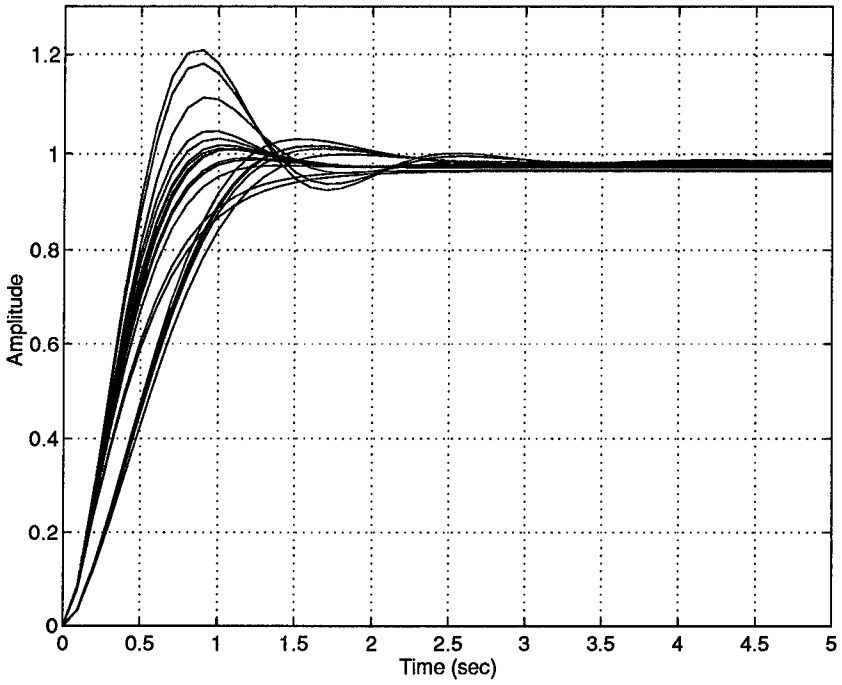


Figure 5.43 Case IIId: Time Response for a Step Input

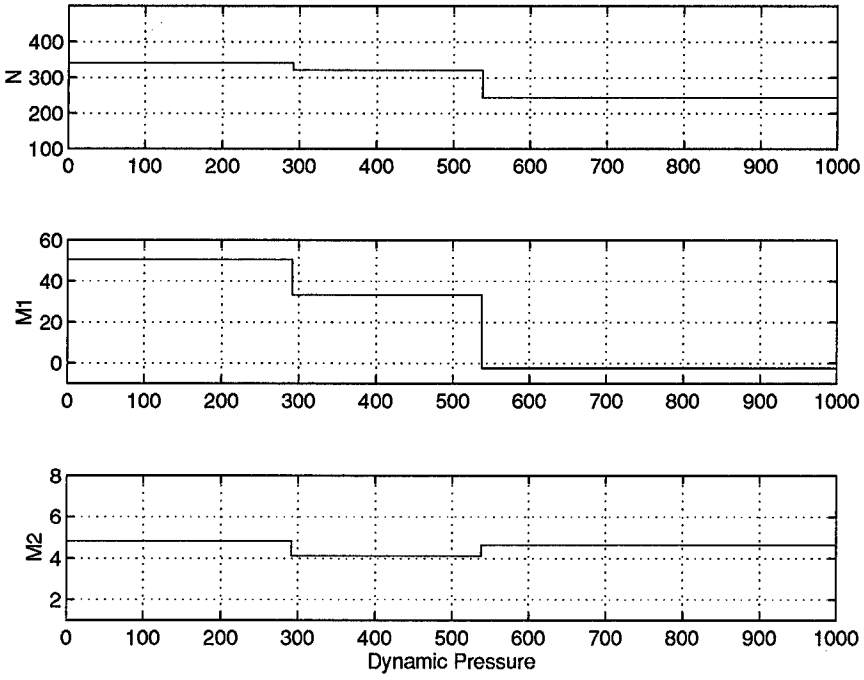


Figure 5.44 Case IIId: Controller Parameter Schedule

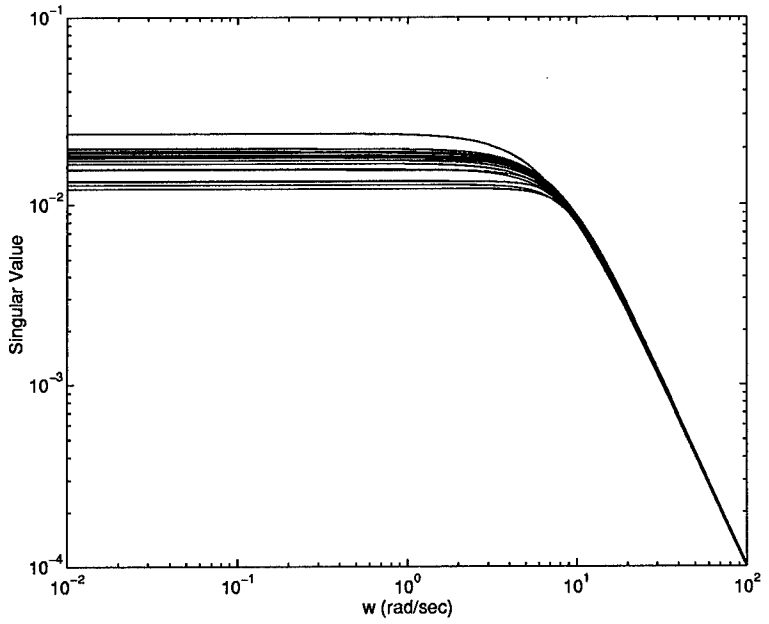


Figure 5.45 Case IVa: $\dot{q}_c$ to α Closed-Loop Dynamics

5.4.6.1 Case IVa. Five intervals are chosen with an objective function value of 2.07. This can be reduced to four intervals to eliminate the 'spike' in the schedule. The closed inner-loop responses are shown in Fig. 5.45 and Fig. 5.46. The relative error and time responses are shown in Fig. 5.47 and Fig. 5.48 and the resulting scheduled variables are shown in Fig. 5.49.

The most significant result of this optimization is that the relative error is reduced by allowing the central flight condition to vary. Note that the worst relative error has a value of 0.32 and the best relative error has a value of 0.0001. Additionally, even though there was no penalty on the number of intervals, the maximum allowable number of intervals was not optimal. The closed loop responses and the time responses are slightly better than those in Case III.

5.4.6.2 Case IVb. Here, seven intervals are chosen with an objective function value of 2.26. However, this can be reduced to six by eliminating the 'spike' in the schedules. The closed inner-loop responses are shown in Fig. 5.50 and Fig. 5.51. The relative error and

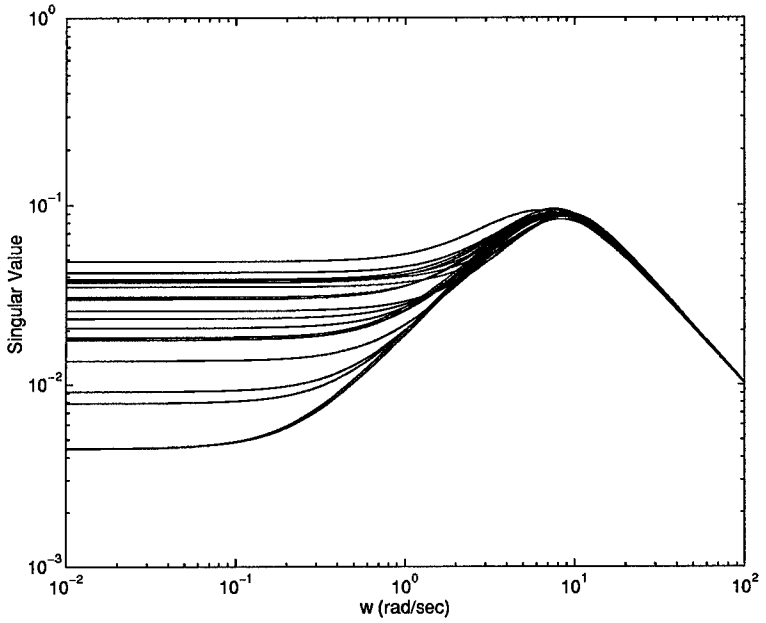


Figure 5.46 Case IVa: $\dot{q}_c$ to q Closed-Loop Dynamics

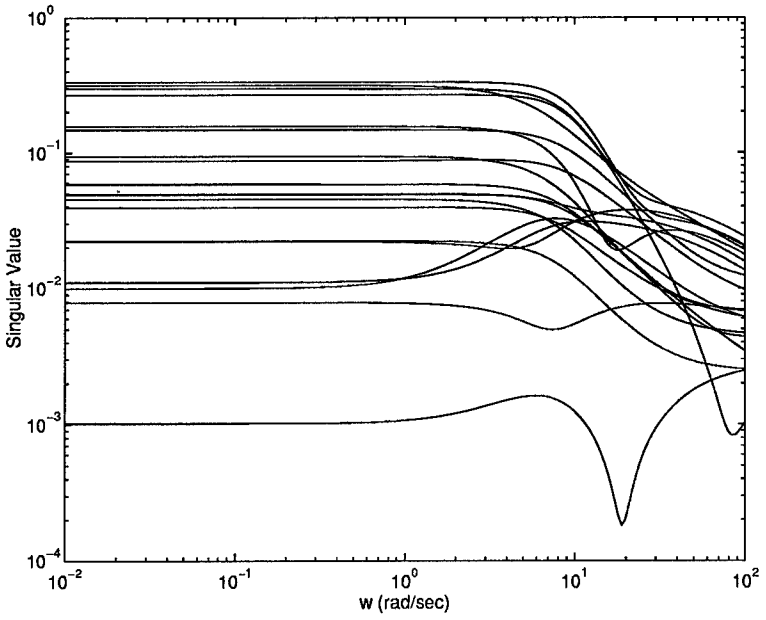


Figure 5.47 Case IVa: Relative Error

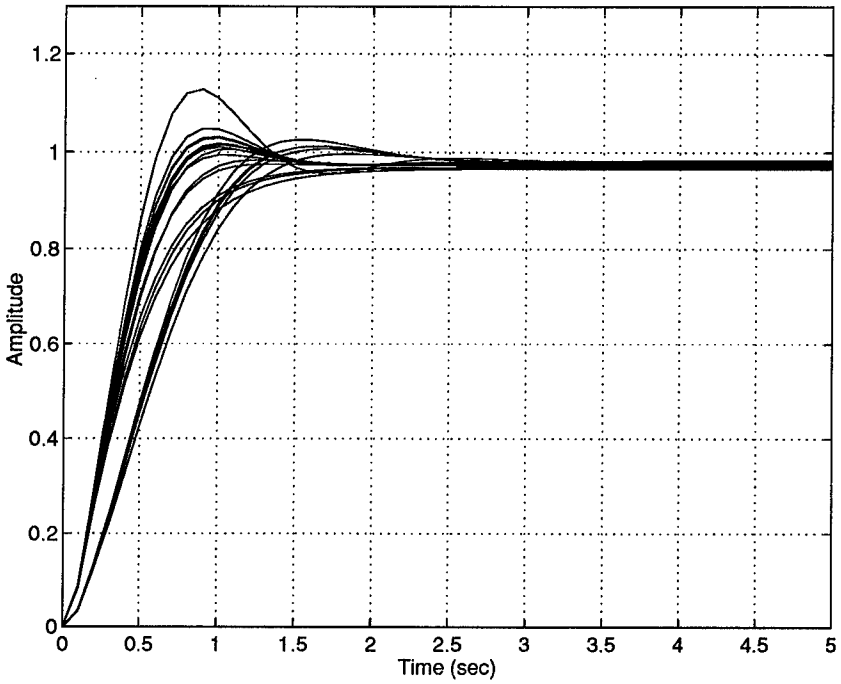


Figure 5.48 Case IVa: Time Response for a Step Input

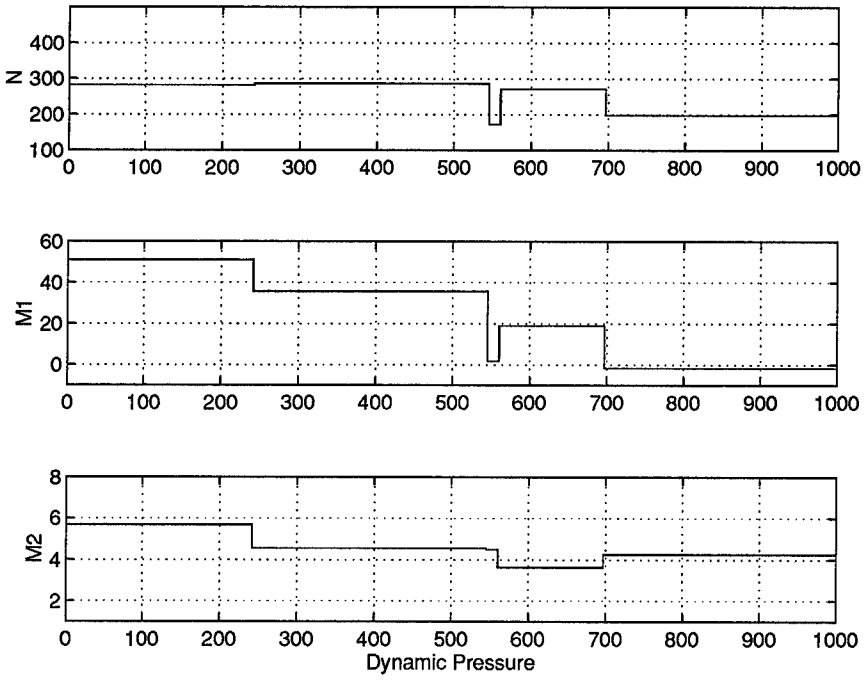


Figure 5.49 Case IVa: Controller Parameter Schedule

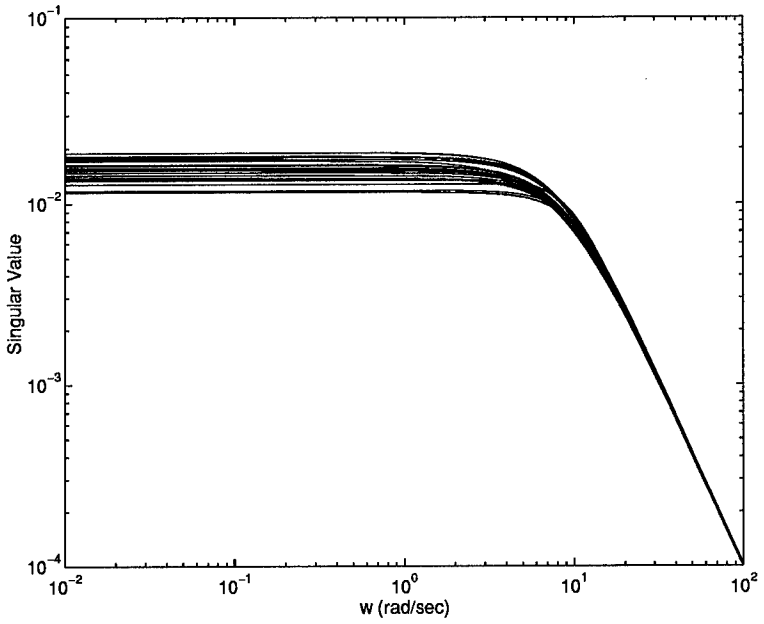


Figure 5.50 Case IVb: $\dot{q}_c$ to α Closed-Loop Dynamics

time responses are shown in Fig. 5.52 and Fig. 5.53 and the resulting scheduled variables are shown in Fig. 5.54. The objective function value is 2.26 with seven intervals.

The results of this optimization are similar to those in Case IVa. The significant difference is in the time response. For the flight conditions evaluated, the time response for all flight conditions was very nearly uniform even though the resulting objective function is not the smallest found so far. This seems to indicate that there is not as close a correlation between the closed loop frequency response and the closed loop time response as previously thought.

5.4.7 Case V. Since there is an indication that the closed loop frequency response error and the closed loop time response are not directly related, this optimization directly optimizes both objectives. The time domain error is measured as the difference between the central controller's time response and the time response at all the other flight conditions. Time time response if evaluated for 5.0 seconds at increments of 0.1 seconds to determine the time

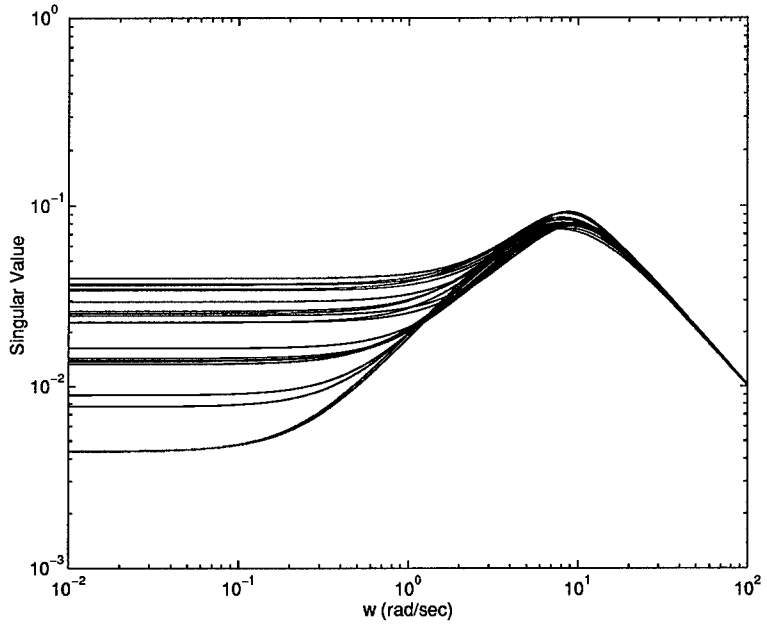


Figure 5.51 Case IVb: $\dot{q}_c$ to q Closed-Loop Dynamics

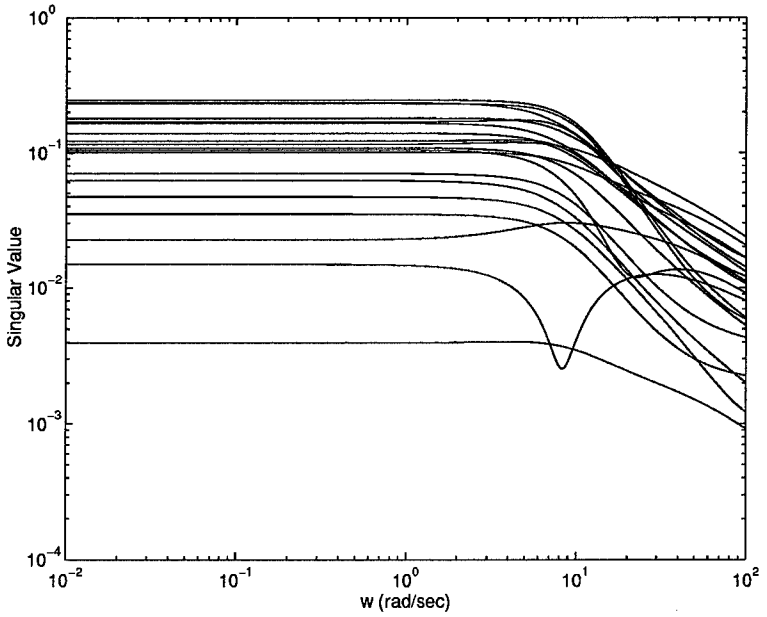


Figure 5.52 Case IVb: Relative Error

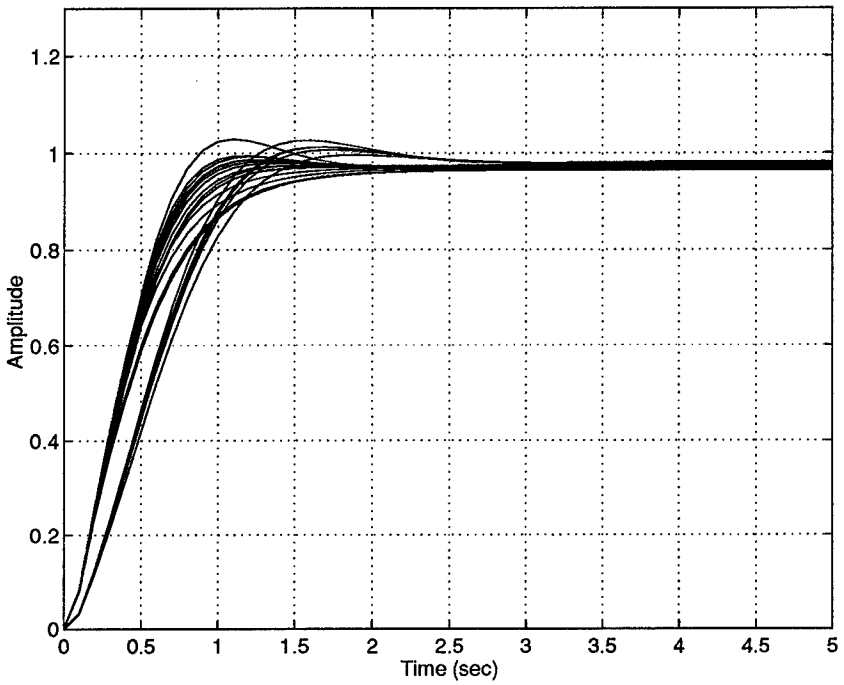


Figure 5.53 Case IVb: Time Response for a Step Input

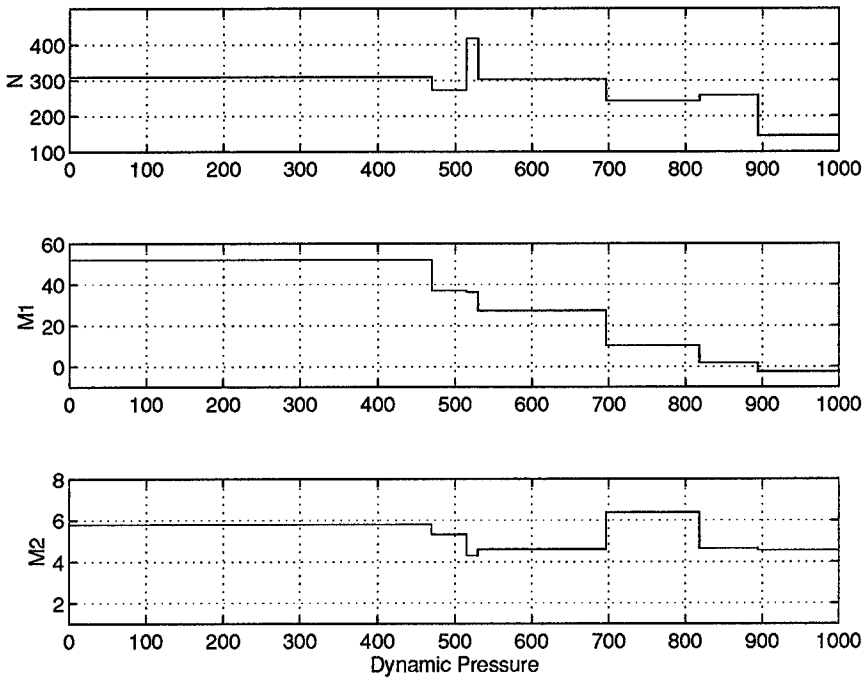


Figure 5.54 Case IVb: Controller Parameter Schedule

error (See Eq (5.19)).

$$e(t) = \sum_{j=1}^{50} y_j(t) - y_0(t) \quad t = 0, 0.1, 0.2, \cdots, 5 \tag{5.19}$$

where $y_0(t)$ is the time response of the closed outer-loop at the central flight condition and $y_j(t)$ is the time response of the closed outer loop at a different flight condition.

The number of design variables is the same as in Case IV.

Five intervals are chosen with an objective function value of 14.22. The closed inner-loop responses are shown in Fig. 5.55 and Fig. 5.56. The relative error and time responses are shown in Fig. 5.57 and Fig. 5.58 and the resulting scheduled variables are shown in Fig. 5.59.

The closed loop frequency responses are similar to those of the previous cases. The relative error is reduced from the baseline but is not near the minimum relative error value found so far. However, the time response is very closely uniform for all of the flight conditions.

5.4.8 Case VI. This optimization duplicates Case IV except that the scheduled variables are now piecewise linear functions of the scheduling variable. The number of design variables is increased by three, since an addition set of scheduled variables is needed for a linear fit The number of design variables now varies between 12 and 40.

Three intervals are chosen with an objective function value of 1.98, the lowest of all cases. The closed inner-loop responses are shown in Fig. 5.60 and Fig. 5.61. The relative error and time responses are shown in Fig. 5.62 and Fig. 5.63 and the resulting scheduled variables are shown in Fig. 5.64.

The significant difference between this case and the previous cases is that the minimum relative error was found sooner with fewer function evaluations. This case also found the smallest relative error of all the optimization cases. The result of this minimal relative error is evident in Fig. 5.60 and Fig. 5.61 where there is a tight grouping of the closed loop response.

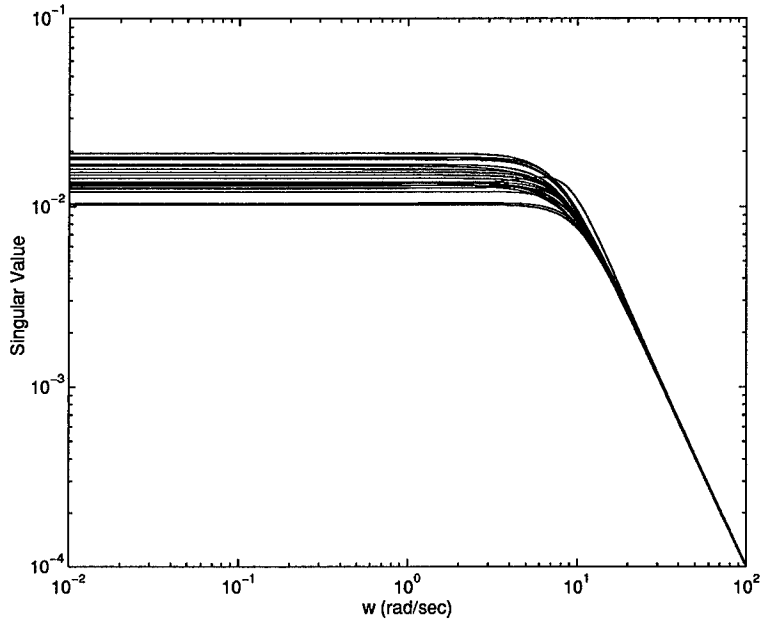


Figure 5.55 Case V: $\dot{q}_c$ to α Closed-Loop Dynamics

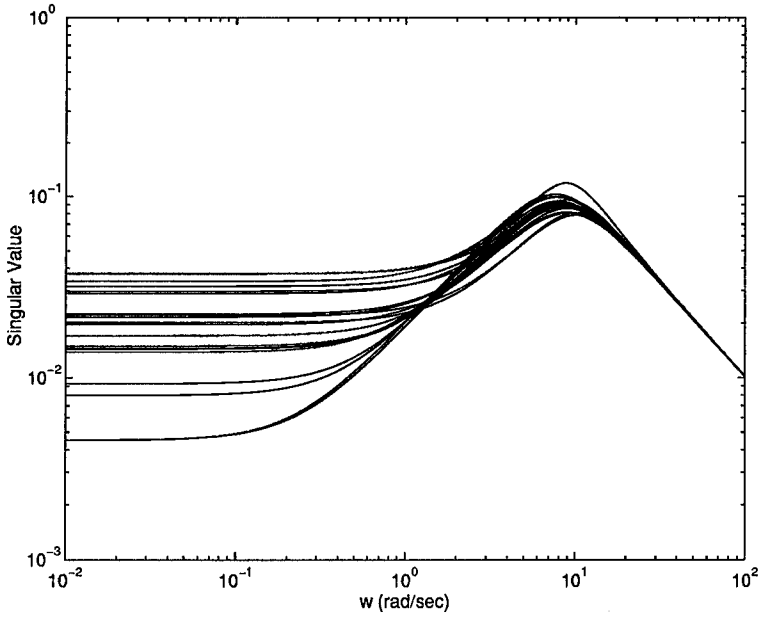


Figure 5.56 Case V: $\dot{q}_c$ to q Closed-Loop Dynamics

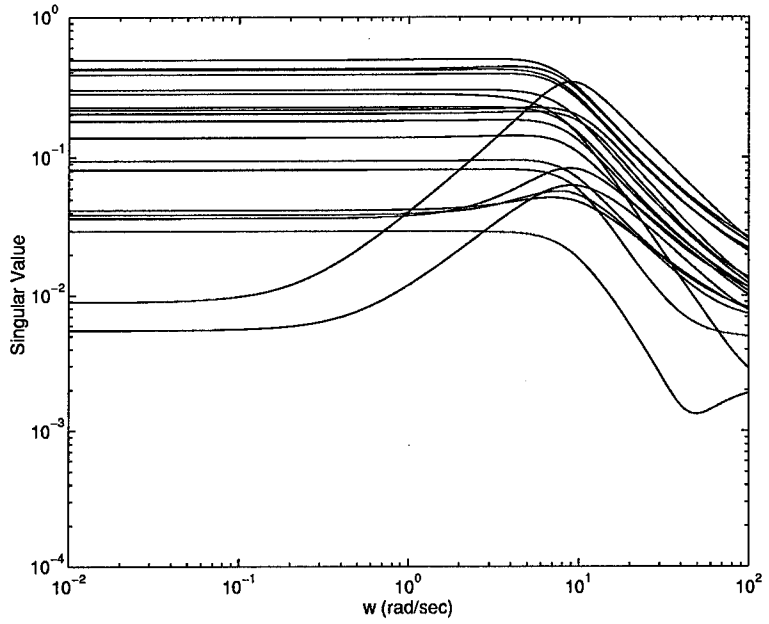


Figure 5.57 Case V: Relative Error

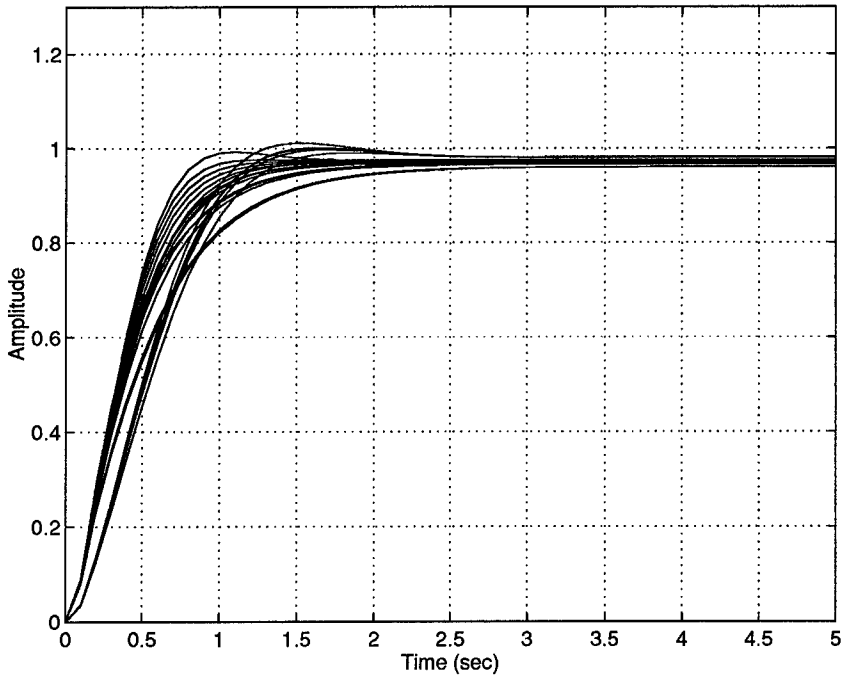


Figure 5.58 Case V: Time Response for a Step Input

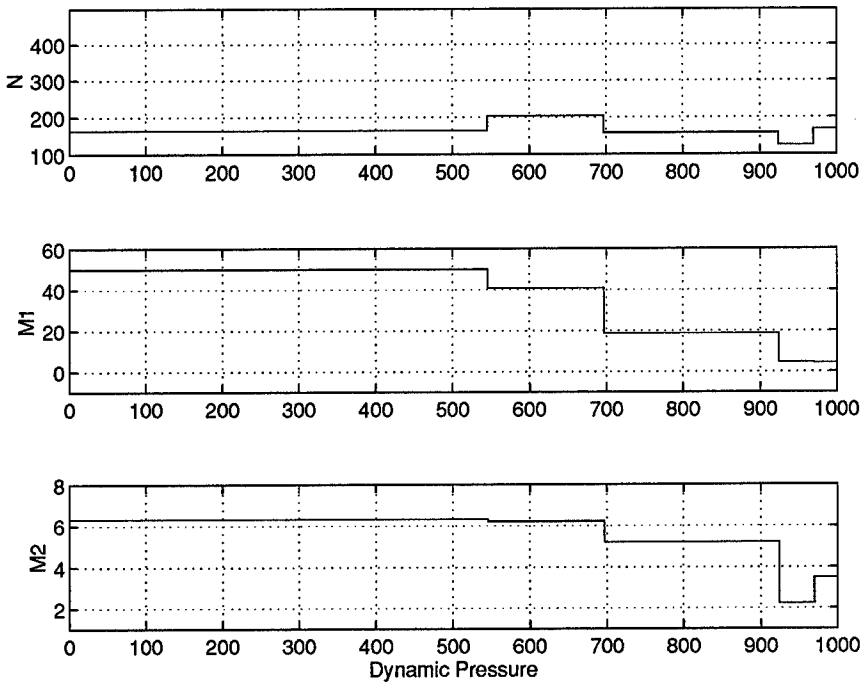


Figure 5.59 Case V: Controller Parameter Schedule

Since the schedule is better, piecewise linear versus piecewise constant, the optimization results are better than the previous cases.

5.4.9 Case VII. This optimization duplicates Case V for piecewise linear scheduled variables. The number of design variables now varies between 12 and 40.

Two intervals are chosen as optimal with no penalty function on the number of intervals. The objective function value is 12.25, which is less than Case V results. The closed inner-loop responses are shown in Fig. 5.65 and Fig. 5.66. The relative error and time responses are shown in Fig. 5.67 and Fig. 5.68 and the resulting scheduled variables are shown in Fig. 5.69.

The closed loop frequency responses and the relative error results are similar to those in Case V, but the time response is not quite as uniform as the time response in Case V. This might seem odd since the value of the objective function is less than that of Case V. This is because each of these cases is using a different central controller.

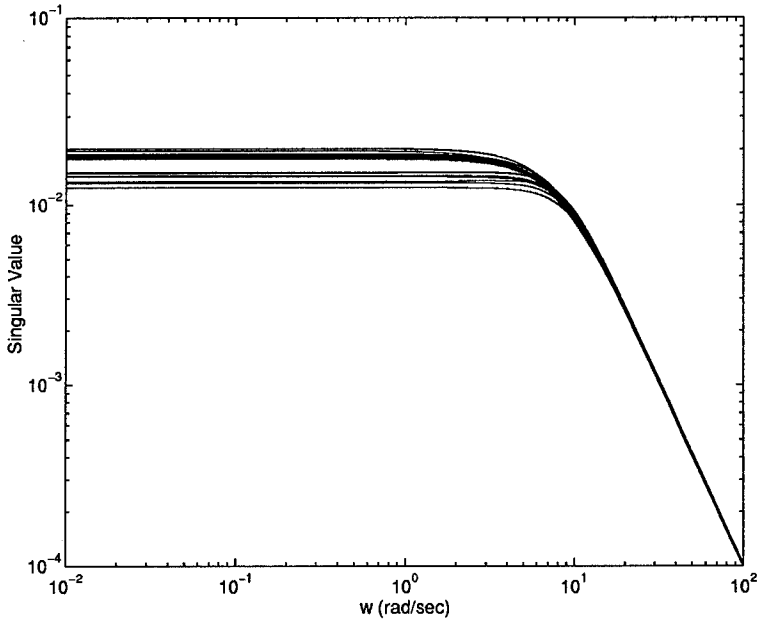


Figure 5.60 Case VI: $\dot{q}_c$ to α Closed-Loop Dynamics

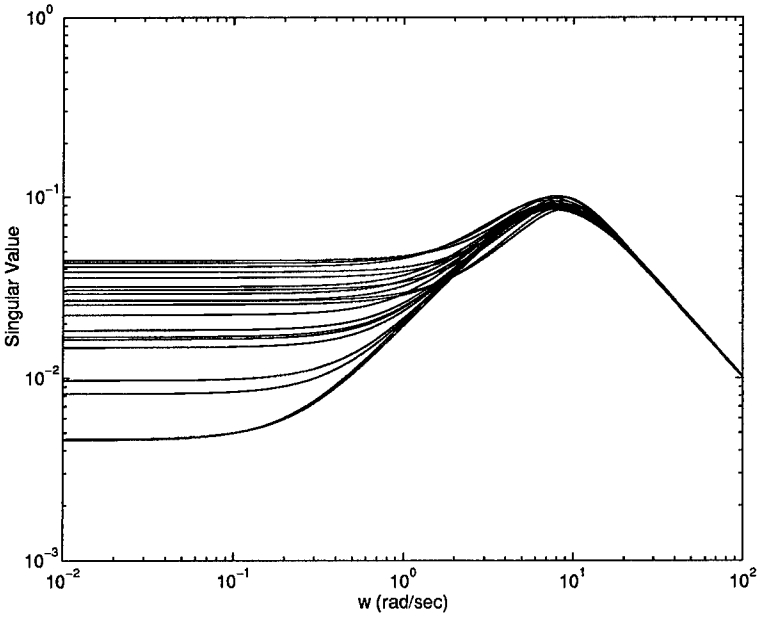


Figure 5.61 Case VI: $\dot{q}_c$ to q Closed-Loop Dynamics

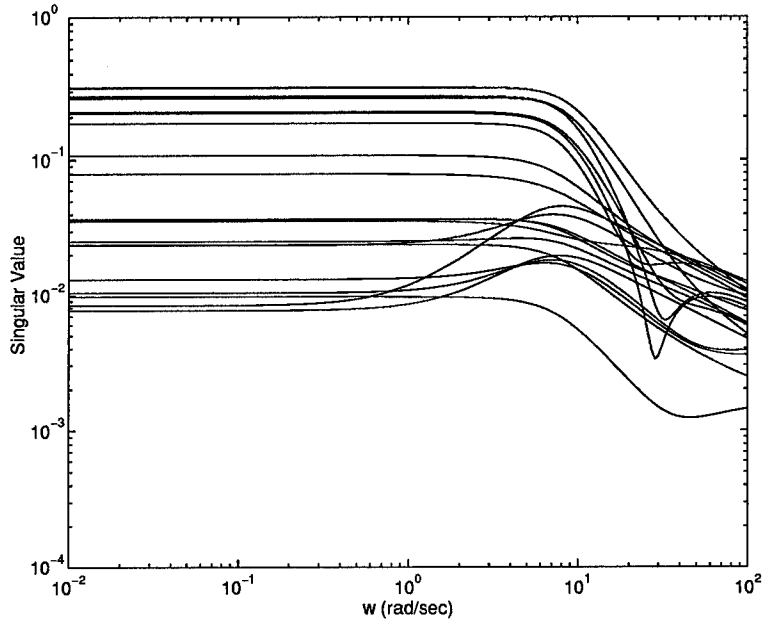


Figure 5.62 Case VI: Relative Error

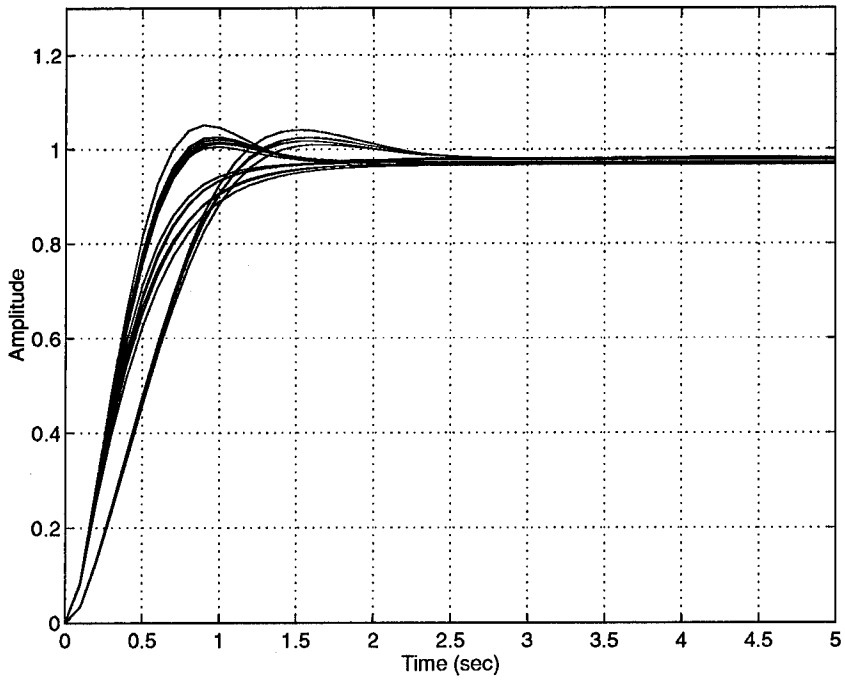


Figure 5.63 Case VI: Time Response for a Step Input

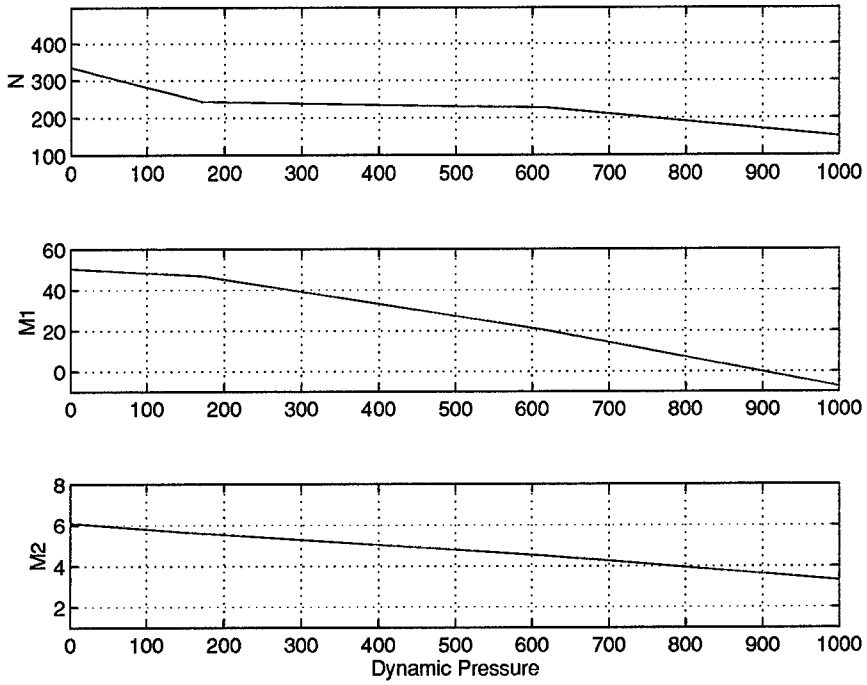


Figure 5.64 Case VI: Controller Parameter Schedule

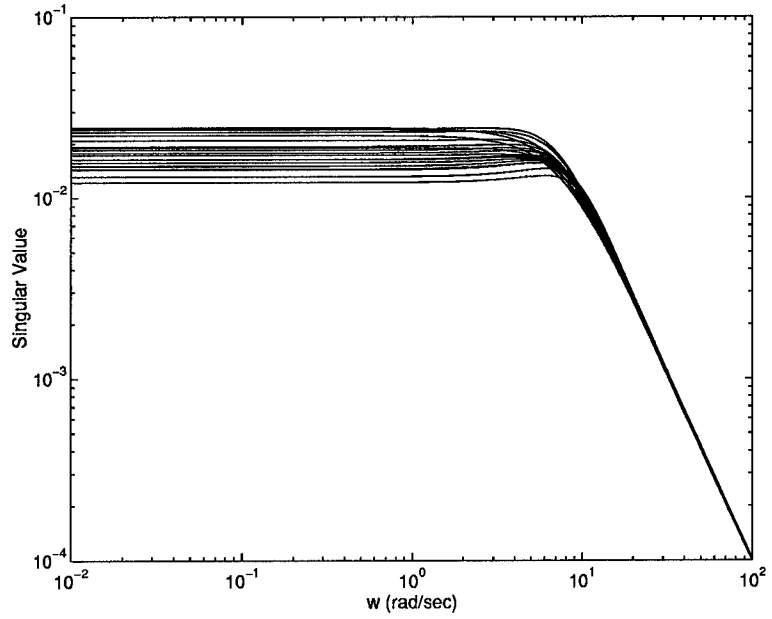


Figure 5.65 Case VII: $\dot{q}_c$ to α Closed-Loop Dynamics

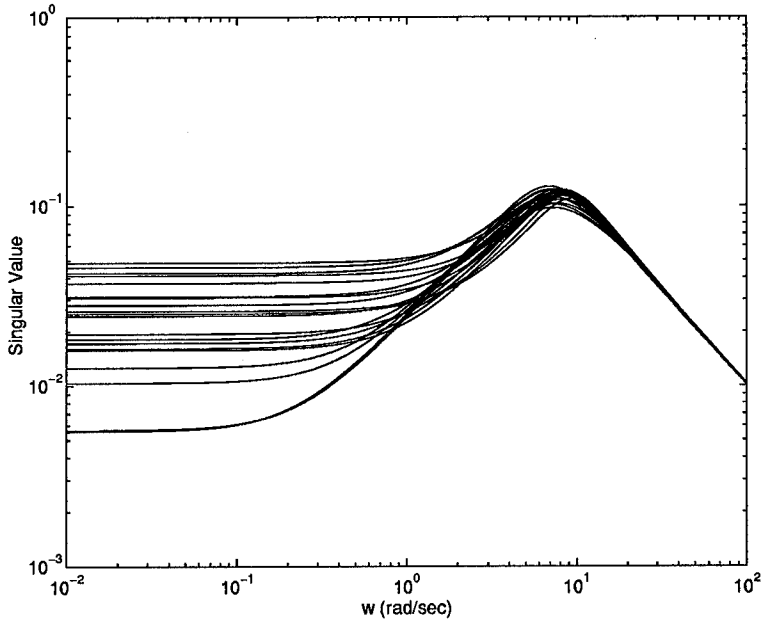


Figure 5.66 Case VII: $\dot{q}_c$ to q Closed-Loop Dynamics

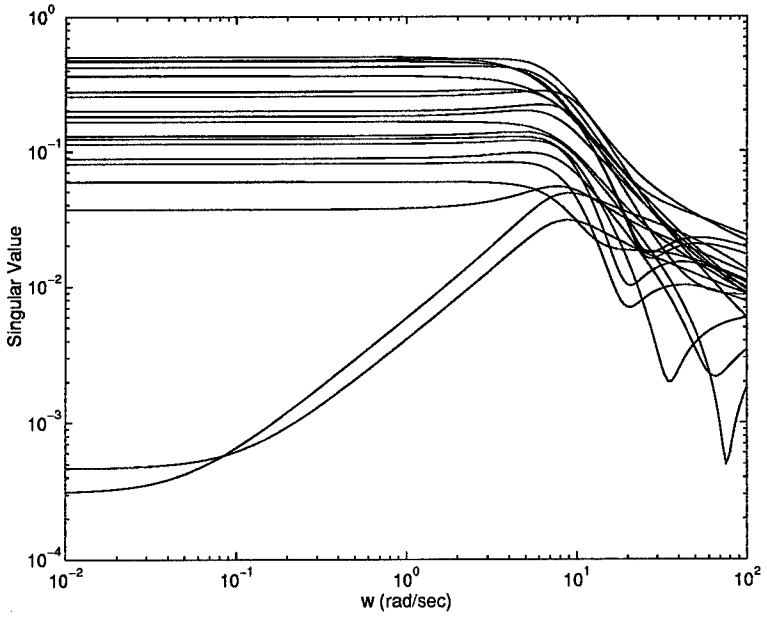


Figure 5.67 Case VII: Relative Error

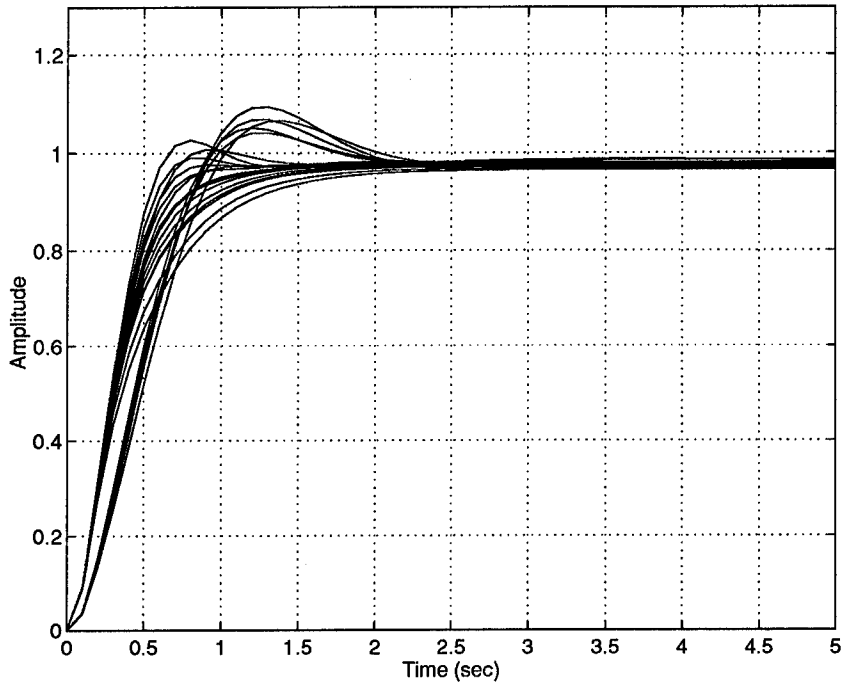


Figure 5.68 Case VII: Time Response for a Step Input

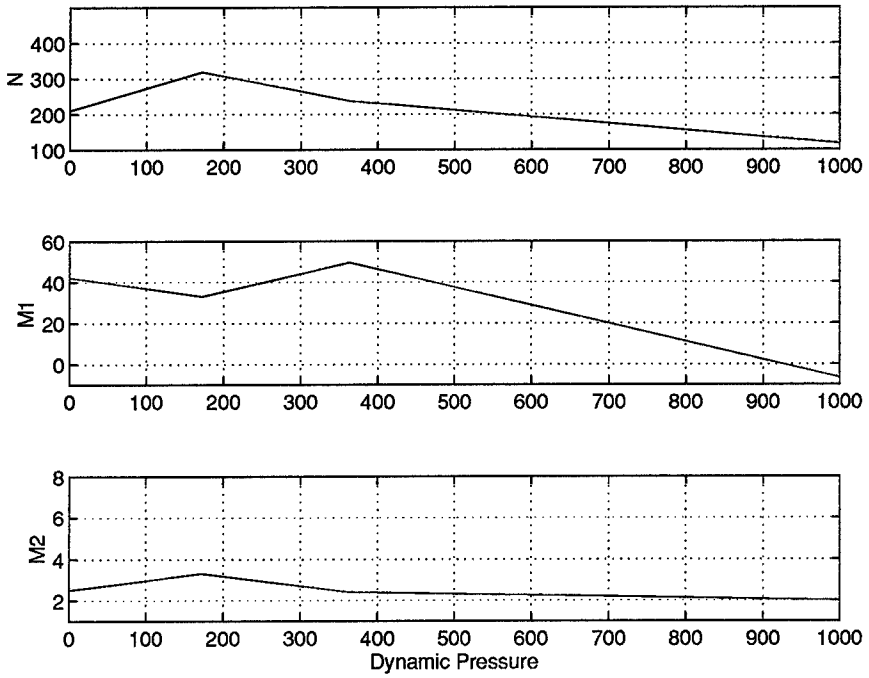


Figure 5.69 Case VII: Controller Parameter Schedule

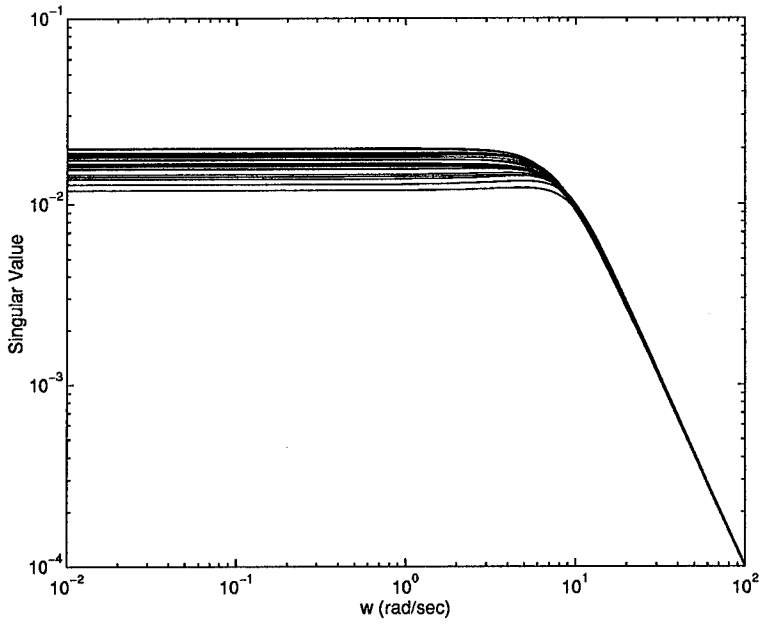


Figure 5.70 Case VIII: $\dot{q}_c$ to α Closed-Loop Dynamics

5.4.10 Case VIII. For this optimization, objective function J_4 is used to more equally weight the time domain error and the frequency domain error. The design variables are the same as in Case VII. Since the relative error is less in magnitude than the time domain error, the time domain error is scaled smaller. Whereas, objective function J_3 placed an emphasis on the time domain error.

Two intervals are again chosen as optimal with no penalty on the number of intervals. The objective function value is 5.16. The closed inner-loop responses are shown in Fig. 5.70 and Fig. 5.71. The relative error and time responses are shown in Fig. 5.72 and Fig. 5.73 and the resulting scheduled variables are shown in Fig. 5.74.

The time response is almost uniform over the operating envelope for the points evaluated. Additionally, the closed loop responses are very uniform for frequencies greater than 5 radians per second. Furthermore, the control parameter schedules are very simple with only two intervals and piecewise linear function of dynamic pressure. Therefore, this optimization is selected as the best. This schedule is validated in section 5.4.12.

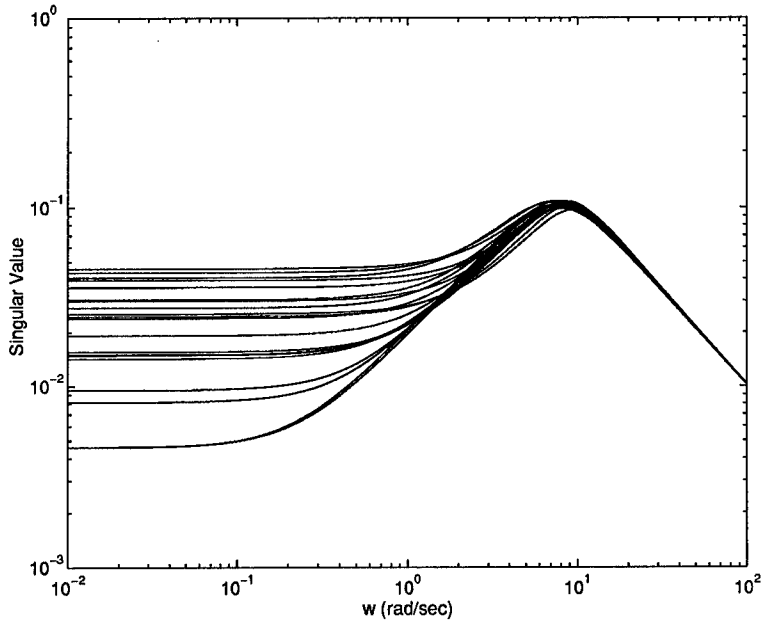


Figure 5.71 Case VIII: $\dot{q}_c$ to q Closed-Loop Dynamics

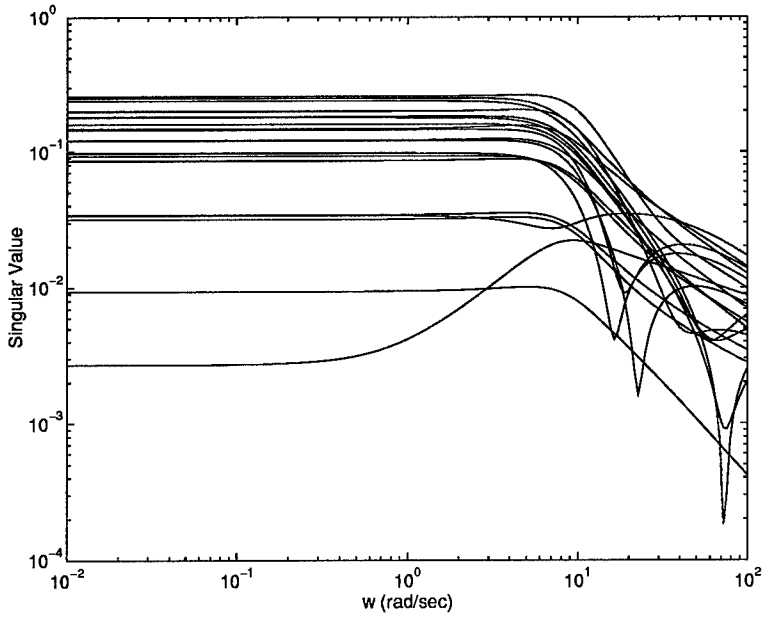


Figure 5.72 Case VIII: Relative Error

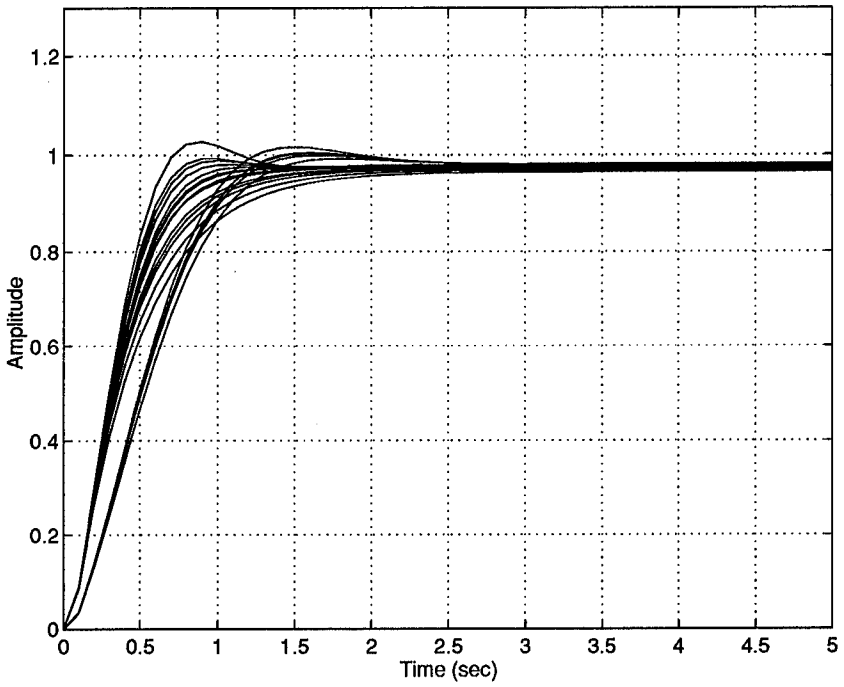


Figure 5.73 Case VIII: Time Response for a Step Input

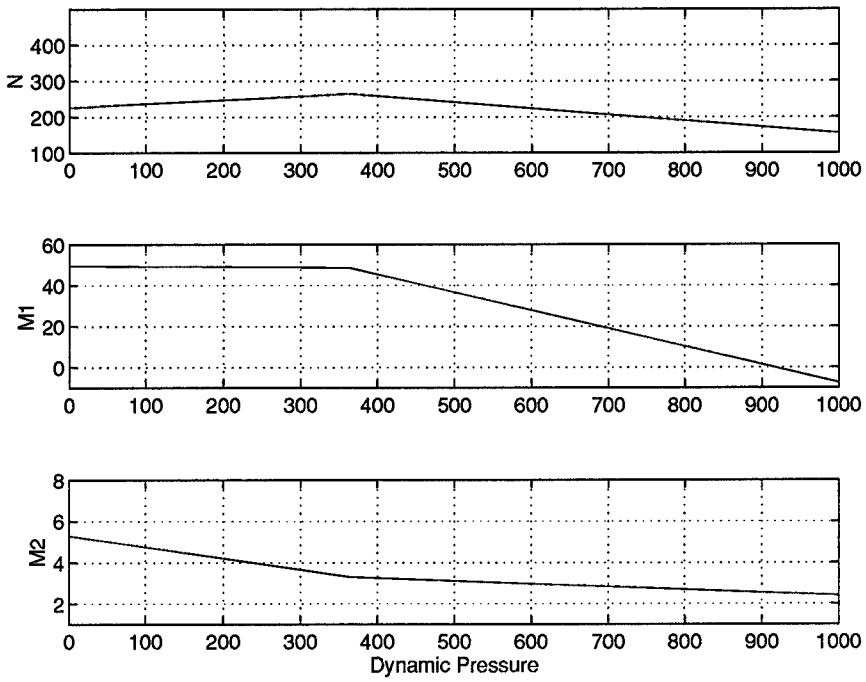


Figure 5.74 Case VIII: Controller Parameter Schedule

5.4.11 Summary of Results. A summary of the optimization results are shown in Table 5.2. For each case the values of all four objective functions are calculated where possible for comparison. The best result for each objective function is emphasized. No single optimization is able to optimize more than one objective simultaneously. Each case demonstrates that the gain schedule can be optimized with respect to the specified objective function. By changing the objective function to meet the desired goal, an optimal gain schedule is achievable.

Every single optimization case tried is able to improve on the baseline design. The resulting optimization has a more consistent time response for the flight conditions evaluated. Additionally, another benefit of the optimization is that the time response was improved in terms of overshoot for most cases.

From these optimizations, a few important facts become evident. First, the closed loop response of the system can be made more uniform across the operating envelope. Second, the most effective gain schedule optimization results when the control parameters are scheduled as piecewise linear functions of the scheduling variable. Additionally, the number and size of the scheduling variable intervals are allowed to vary as design variables. Furthermore, when using a central controller for computation of gain scheduling error, the selection of which controller is central can be an effective design variable in reducing the calculated error.

To see how the central controller varied with the optimization cases see Fig. 5.75. The 'x' marks the central controller selected for the baseline design using engineering judgment. The 'o's' mark the central controllers selected by the optimization. The next section uses the best optimized schedule, Case VIII, and validates the schedule using operating points not in the original design set.

5.4.12 Design Validation. To validate the optimal gain schedule designed in the previous section, six more flight conditions are selected across the flight envelope. These are shown with 'o's' in Fig. 5.76. These operating points are specifically selected to be near the corners of the operating envelope. The relative error and time response of these points

Table 5.2 Comparison of Optimization Results

Case	J_1	J_2	J_3	J_4	N	$f(N)$	P_0
0	6.37		23.24	9.74			M.95h20
I	3.09*		21.89	6.85	4		M.95h20
II	2.51*		22.62	6.53	4		M.95h20
IIIa	3.24	5.24*	20.80	6.75	2	N	M.95h20
IIIb	3.23	7.23*	21.16	6.81	2	N^2	M.95h20
IIIc	3.22	3.45*	21.12	6.80	2	$N/9$	M.95h20
IIId	3.13	3.24*	23.07	7.11	3	$(N/9)^2$	M.95h20
IVa	2.07	**2.07***	26.82	7.02	5		M.4h22
IVb	2.19	2.26*	16.56	**5.06**	7	$N/100$	M.5h10
V	4.09		14.22*	6.22	5		M.6h2
VI	**1.98***		19.42	5.47	3		M.7h18.5
VII	4.15		**12.65***	5.85	2		M.6h2
VIII	2.44		16.05	**5.16***	2		M.6h15

* denotes the objective function used

Note: the bold numbers are the minimum for each objective function

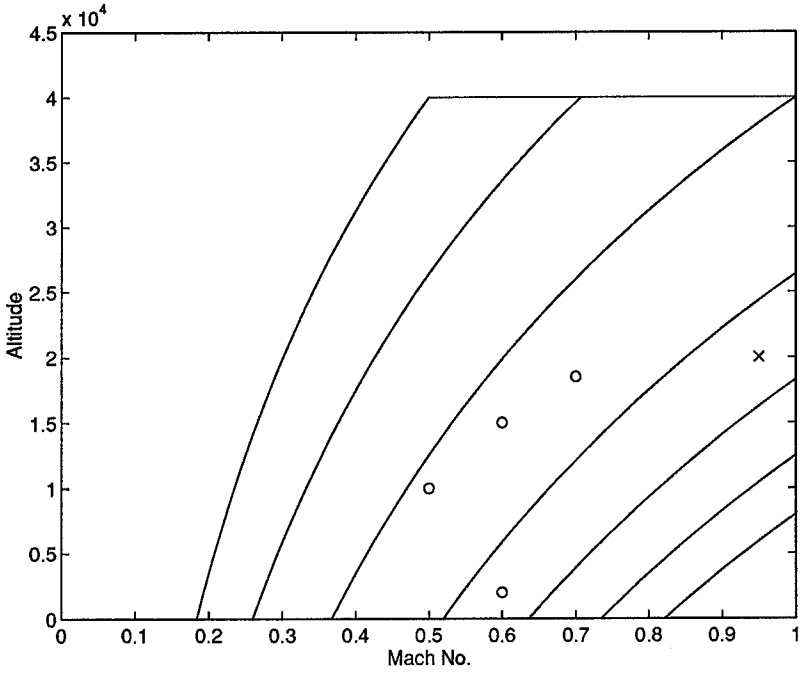


Figure 5.75 Central Controller Variations

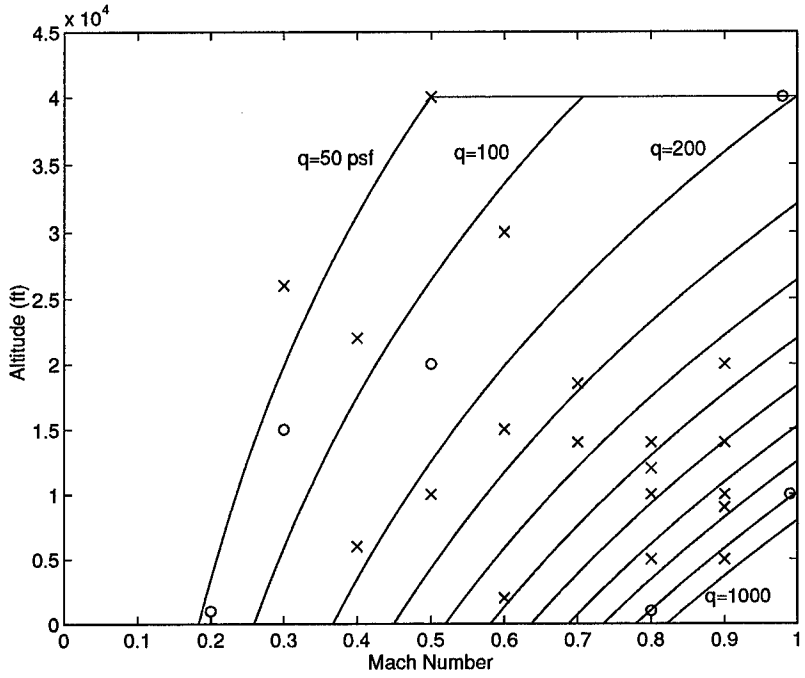


Figure 5.76 F-18 Flight Envelope

using the baseline schedule are shown in Fig. 5.77 and Fig. 5.78. The relative error and time response of these points using the optimized schedule are shown in Fig. 5.79 and Fig. 5.80.

The purpose of this step is to complete the gain scheduling design process for the optimally selected schedules. By inspecting the relative error graphs, Fig. 5.77 and Fig. 5.79, all the relative errors are less than one therefore stability is maintained. The difference is that for the baseline schedule the relative error for these six flight conditions is 2.87, whereas, for the optimal schedule the relative error is only 1.1. For both schedules, the time responses have acceptable characteristics. However, with a lower relative error of the optimally scheduled controller, it is easier to design an outer-loop controller to provide robust performance throughout the operating envelope. From the lessons learned in these optimizations, a general design procedure is developed in the next chapter.

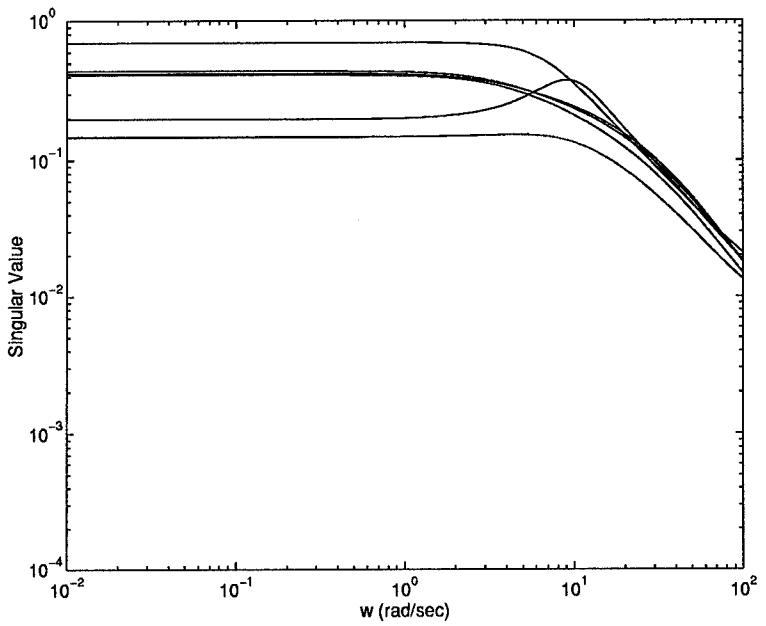


Figure 5.77 Baseline Relative Error Validation

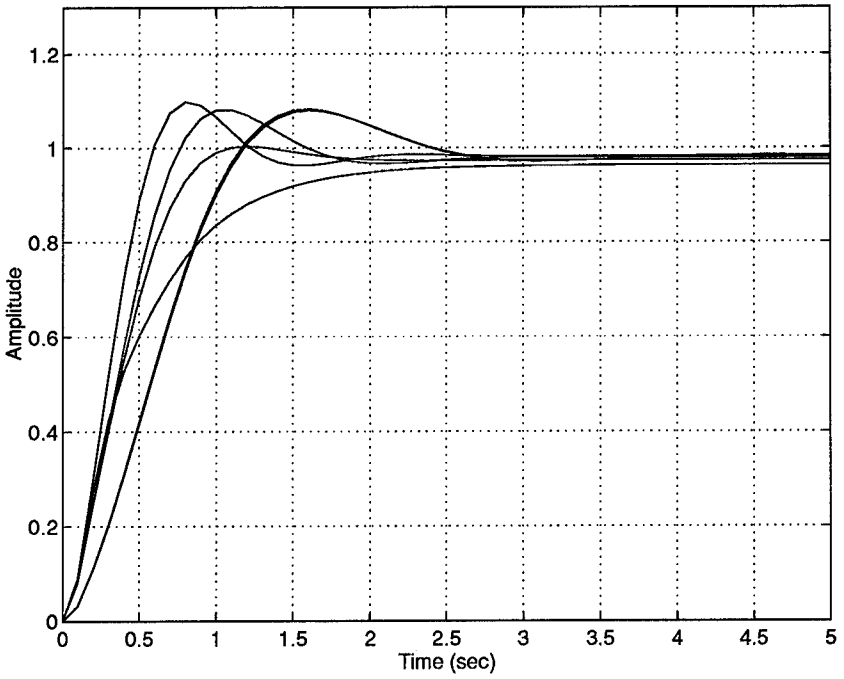


Figure 5.78 Baseline Time Response Validation

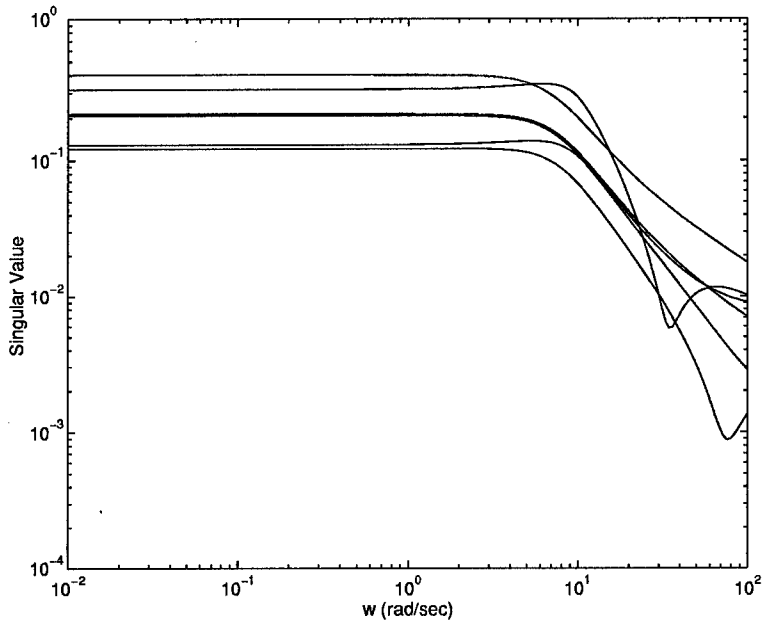


Figure 5.79 Optimial Schedule Relative Error Validation

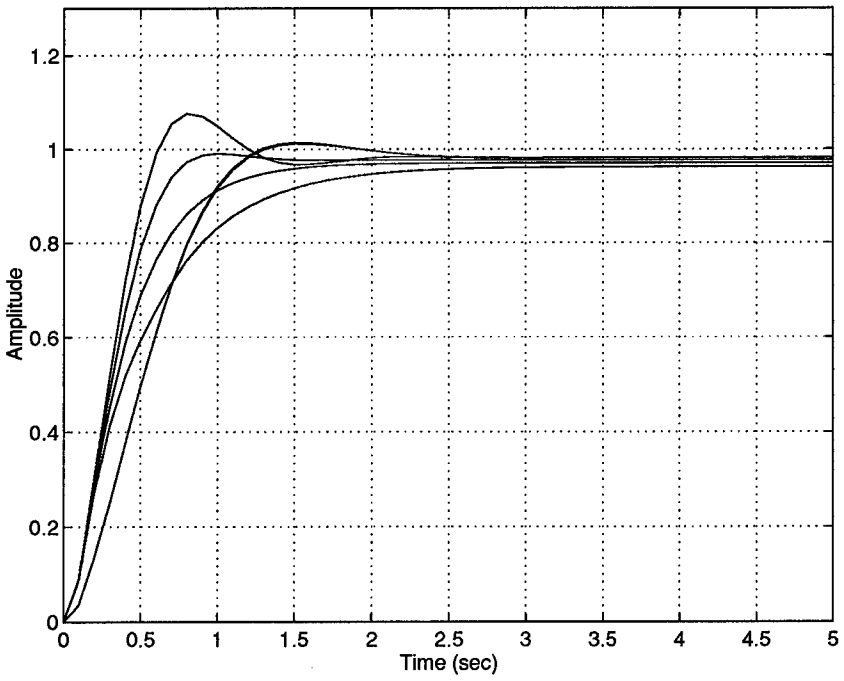


Figure 5.80 Optimal Schedule Time Response Validation

VI. General Design Process

A great deal of experience was gained in designing and optimizing the gain schedules presented in chapters IV and V. From this experience a systematic gain scheduling method has been developed. The next section presents this formal design process.

6.1 Gain Scheduling Design

There are many aspects of the system to consider when designing a controller for a real world system. Traditionally, the method of implementing the designed controller is not considered until the final stages of the design process. Ideally, the designed controller should be robust enough to not require gain scheduling, but more often than not some scheduling is required. The design method presented here differs fundamentally from most design methods in that the scheduling of the controller is considered at the beginning of the design process.

There are four steps in the design process and are as follows:

I. Control Problem Definition First the control designer must select the scheduling variable based on criteria provided in [41, 30, 43, 46]. The system should be evaluated to avoid selecting a poor scheduling variable as outlined in [43, 46]. Second, the family of plants is chosen to represent the extrema of the operating envelope and to sufficiently represent the areas between these extrema. The plants should represent the plant dynamics accurately. Additionally, for areas of the operating envelope where the plant dynamics change rapidly, additional plants should be chosen.

II. Controller Design Next select a general form of the desired controller. After defining the form of the controller, determine the number of controller parameters, n_{scp}, that are to be scheduled. For some controllers all of the parameters are scheduled, such as a full state feedback gain matrix. In other controllers, such as the one in Chapter V, only some of the controller parameters vary. Once the scheduled parameters are identified, design a controller at the extrema of the scheduling variable to meet the requirements of the

system. This provides upper and lower bound of the scheduled variables, although the control designer can extend these bounds to increase the search space. By bounding the parameters the control designer is able to define the space of the optimal gain schedule controller.

III. Gain Schedule Optimization Next the gain schedule is optimized using GAs. The scheduling variable is decomposed into n intervals with each interval having an arbitrary size. The scheduled parameters are bound by the designs performed in Step II and allowed to vary linearly in each interval. The number of intervals is chosen to vary within a selected range, $n \in (n_{min}, n_{max})$. Additionally, the central flight condition is allowed to vary within the family of plants chosen in Step I. For the optimization, the number of design variables varies between $(n_{min} + n_{min} \times n_{scp}), and (n_{max} + n_{max} \times n_{scp})$. An objective function is then defined that measures the gain scheduling error of the family of plants. Now, a GA can be used to optimize the objective function with the design variables described.

IV. Analyze Results Finally, select a family of plants not included in the family of Step I. Simulate the closed loop response of the system with the scheduled controller to insure that the system requirements are achieved.

6.2 Conclusions

The objectives set forth in Chapter I were achieved. First, a simple GA was found that could be implemented with a common computer-aided control design software package. This enables the designer to quickly and efficiently analyze and optimize a gain scheduled controller. Second, a method of evaluating and measuring the effectiveness of a gain schedule design was developed. This evaluation measure allowed for the comparison of various gain schedules. Next, using the gain schedule measurement, a simple gain scheduling problem was optimized to validate the use of GAs as an optimization tool. Then, the gain scheduling optimization method was used to evaluate a gain schedule designed for an F-18 fighter aircraft. It is shown in Chapter V that the gain schedule originally developed could be optimized and

the resulting time response of the system was improved. Finally, the lessons learned from this research were compiled into a formal design method for designing a gain scheduled controller.

6.3 Future Research

Future research efforts could focus on improving the computational efficiency of the genetic algorithm. Additionally, the precision of the GA could be improved by using a floating point chromosome and a variable mutation operator. These improvements focus on the GA itself. The basic approach employed in this effort is to decompose the gain schedule into a piecewise function of the scheduling variable. Future work could analyze the direct implement ion and optimization of a polynomial scheduling function. Further research is also needed in the area of determining global stability and performance for a scheduled controller.

Appendix A. F-18 Design Flight Conditions

Following are the flight conditions and their respective plant matricies that were used in the evaluation of the gain scheduled controller.

Table A.1 Longitudinal Flight Conditions For Optimization

Mach Number	Altitude (ft)	$\overline{q}$ (psf)	α (deg)
0.3	26000	47.4	25.2
0.5	40000	68.5	16.8
0.4	22000	100.1	8.7
0.6	30000	158.4	5.2
0.4	6000	189.9	6.0
0.5	10000	255.0	3.5
0.6	15000	301.1	2.9
0.7	18500	355.0	2.4
0.7	14000	426.4	2.6
0.6	2000	496.0	1.8
0.8	14000	557.0	1.4
0.8	12000	603.0	1.9
0.95	20000	614.4	1.6
0.8	10000	652.0	1.7
0.9	14000	705.0	1.2
0.8	5000	789.1	1.5
0.9	10000	825.2	1.4
0.85	5000	890.8	1.4
0.95	9000	956.0	0.9
0.9	5000	998.7	1.3

Following are the short period longitudinal dynamic plant matricies. The general form of the longitudinal short period dynamics is shown in Eq (A.1).

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \end{bmatrix} = A_{long} \begin{bmatrix} \alpha \\ q \end{bmatrix} + B_{long} \dot{q}_c \tag{A.1}$$

(A.2)

Table A.2 Longitudinal Flight Conditions For Evaluation

Mach Number	Altitude (ft)	$\overline{q}$ (psf)	α (deg)
0.98	40000	263.3	3.1
0.99	10000	998.5	1.0
0.5	20000	170.1	5.2
0.3	15000	75.3	12.2
0.2	1000	57.2	20.8
0.8	1000	914.6	1.0

The notation with the A and B matricies denotes the flight condition. For example, Am3h26 denotes the A matrix at Mach 0.3 and altitude 26000 feet.

$$A_{long}^{m3h26} = \begin{bmatrix} -0.2296 & 0.9931 \\ 0.02436 & -0.2046 \end{bmatrix} \quad B_{long}^{m3h26} = \begin{bmatrix} -0.04034 & -0.01145 \\ -1.73 & -0.517 \end{bmatrix}$$

$$A_{long}^{m5h40} = \begin{bmatrix} -0.2423 & 0.9964 \\ -2.342 & -0.1737 \end{bmatrix} \quad B_{long}^{m5h40} = \begin{bmatrix} -0.0416 & -0.01141 \\ -2.595 & -0.8161 \end{bmatrix}$$

$$A_{long}^{m6h30} = \begin{bmatrix} -0.5088 & 0.994 \\ -1.131 & -0.2804 \end{bmatrix} \quad B_{long}^{m6h30} = \begin{bmatrix} -0.09277 & -0.01787 \\ -6.573 & -1.525 \end{bmatrix}$$

$$A_{long}^{m4h22} = \begin{bmatrix} -0.4285 & 0.9916 \\ -0.7473 & -0.3123 \end{bmatrix} \quad B_{long}^{m4h22} = \begin{bmatrix} -0.0813 & -0.0145 \\ -4.0770 & -0.7978 \end{bmatrix}$$

$$A_{long}^{m4h6} = \begin{bmatrix} -0.8018 & 0.9847 \\ -1.521 & -0.5944 \end{bmatrix} \quad B_{long}^{m4h6} = \begin{bmatrix} -0.1508 & -0.02776 \\ -7.926 & -1.751 \end{bmatrix}$$

$$A_{long}^{m7h14} = \begin{bmatrix} -1.175 & 0.9871 \\ -8.458 & -0.8776 \end{bmatrix} \quad B_{long}^{m7h14} = \begin{bmatrix} -0.194 & -0.04349 \\ -19.29 & -3.803 \end{bmatrix}$$

$$A_{long}^{m5h10} = \begin{bmatrix} -0.8930 & 0.9852 \\ -4.1582 & -0.6873 \end{bmatrix} \quad B_{long}^{m5h10} = \begin{bmatrix} -0.1626 & -0.0261 \\ -10.6454 & -2.0318 \end{bmatrix}$$

$$A_{long}^{m6h15} = \begin{bmatrix} -0.9181 & 0.9872 \\ -6.2419 & -0.6920 \end{bmatrix} \quad B_{long}^{m6h15} = \begin{bmatrix} -0.1599 & -0.0265 \\ -12.5295 & -2.4169 \end{bmatrix}$$

$$A_{long}^{m7h18.5} = \begin{bmatrix} -0.9920 & 0.9888 \\ -7.8450 & -0.7525 \end{bmatrix} \quad B_{long}^{m7h18.5} = \begin{bmatrix} -0.1652 & -0.0274 \\ -16.1838 & -2.8619 \end{bmatrix}$$

$$A_{long}^{m6h2} = \begin{bmatrix} -1.4710 & 0.9808 \\ -11.5022 & -1.0846 \end{bmatrix} \quad B_{long}^{m6h2} = \begin{bmatrix} -0.2419 & -0.0417 \\ -21.5227 & -3.9738 \end{bmatrix}$$

$$A_{long}^{m8h14} = \begin{bmatrix} -1.4406 & 0.9868 \\ -14.2709 & -1.0645 \end{bmatrix} \quad B_{long}^{m8h14} = \begin{bmatrix} -0.2154 & -0.0379 \\ -24.3921 & -4.5141 \end{bmatrix}$$

$$A_{long}^{m8h12} = \begin{bmatrix} -1.562 & 0.9862 \\ -14.94 & -1.132 \end{bmatrix} \quad B_{long}^{m8h12} = \begin{bmatrix} -.2316 & -.04349 \\ -26.48 & -5.323 \end{bmatrix}$$

$$A_{long}^{m95h20} = \begin{bmatrix} -1.905 & 0.9895 \\ -33.88 & -0.9872 \end{bmatrix} \quad B_{long}^{m95h20} = \begin{bmatrix} -0.1867 & -0.03287 \\ -27.22 & -4.573 \end{bmatrix}$$

$$A_{long}^{m8h10} = \begin{bmatrix} -1.675 & 0.9853 \\ -16.16 & -1.212 \end{bmatrix} \quad B_{long}^{m8h10} = \begin{bmatrix} -0.2449 & -0.04649 \\ -28.34 & -5.742 \end{bmatrix}$$

$$A_{long}^{m9h14} = \begin{bmatrix} -2.1163 & 0.9872 \\ -32.6459 & -1.1826 \end{bmatrix} \quad B_{long}^{m9h14} = \begin{bmatrix} -0.2450 & -0.0426 \\ -32.6358 & -5.7862 \end{bmatrix}$$

$$A_{long}^{m8h5} = \begin{bmatrix} -1.994 & 0.9828 \\ -19.44 & -1.427 \end{bmatrix} \quad B_{long}^{m8h5} = \begin{bmatrix} -0.2852 & -0.05567 \\ -33.44 & -6.931 \end{bmatrix}$$

$$A_{long}^{m9h10} = \begin{bmatrix} -2.452 & 0.9856 \\ -38.61 & -1.34 \end{bmatrix} \quad B_{long}^{m9h10} = \begin{bmatrix} -0.2757 & -0.05226 \\ -37.36 & -7.247 \end{bmatrix}$$

$$A_{long}^{m85h5} = \begin{bmatrix} -2.328 & 0.9831 \\ -30.44 & -1.493 \end{bmatrix} \quad B_{long}^{m85h5} = \begin{bmatrix} -0.3012 & -0.05866 \\ -38.43 & -7.815 \end{bmatrix}$$

$$A_{long}^{m95h9} = \begin{bmatrix} -2.8375 & 0.9855 \\ -51.8325 & -1.4037 \end{bmatrix} \quad B_{long}^{m95h9} = \begin{bmatrix} -0.2863 & -0.0454 \\ -42.9285 & -6.6039 \end{bmatrix}$$

$$A_{long}^{m9h5} = \begin{bmatrix} -2.911 & 0.9835 \\ -46.47 & -1.553 \end{bmatrix} \quad B_{long}^{m9h5} = \begin{bmatrix} -0.3161 & -0.06231 \\ -43.65 & -8.752 \end{bmatrix}$$

The following flight conditions are those used to evaluate the optimal gain schedule.

$$A_{long}^{m98h40} = \begin{bmatrix} -0.7055 & 0.9949 \\ -17.5821 & -0.4583 \end{bmatrix} \quad B_{long}^{m98h40} = \begin{bmatrix} -0.0918 & -0.0122 \\ -13.0772 & -1.6363 \end{bmatrix}$$

$$A_{long}^{m99h10} = \begin{bmatrix} -2.6317 & 0.9860 \\ -76.1833 & -1.3868 \end{bmatrix} \quad B_{long}^{m99h10} = \begin{bmatrix} -0.2783 & -0.0390 \\ -44.1706 & -5.8898 \end{bmatrix}$$

$$A_{long}^{m5h20} = \begin{bmatrix} -0.6199 & 0.9900 \\ -1.8909 & -0.4433 \end{bmatrix} \quad B_{long}^{m5h20} = \begin{bmatrix} -0.1149 & -0.0186 \\ -7.1405 & -1.3628 \end{bmatrix}$$

$$A_{long}^{m3h15} = \begin{bmatrix} -0.3664 & 0.9887 \\ 0.0321 & -0.3425 \end{bmatrix} \quad B_{long}^{m3h15} = \begin{bmatrix} -0.0698 & -0.0134 \\ -2.7177 & -0.5925 \end{bmatrix}$$

$$A_{long}^{m2h1} = \begin{bmatrix} -0.3678 & 0.9843 \\ 0.3351 & -0.3022 \end{bmatrix} \quad B_{long}^{m2h1} = \begin{bmatrix} -0.0719 & -0.0146 \\ -1.9761 & -0.4288 \end{bmatrix}$$

$$A_{long}^{m8h1} = \begin{bmatrix} -2.2679 & 0.9803 \\ -22.8644 & -1.6265 \end{bmatrix} \quad B_{long}^{m8h1} = \begin{bmatrix} -0.3193 & -0.0586 \\ -37.6778 & -7.4086 \end{bmatrix}$$

Appendix B. Computer Codes

B.1 Example Problem Code

Following is the *Matlab*© code that was used to evaluate the objective function for the sample control problem in chapter IV. There are four m-files shown. The are the fixed interval objective function, the variable interval size objective function, the variable interval number and size objective function, and the linear schedule objective function, respectively.

```
%****************************************** file fixint.m ****
function [f]=fixint(k);

%  This function is an evaluation routine run from genesis to
% evaluate the distance from the closed loop poles of a system
% with a given gain to the desired closed loop poles over
% a range of a varying parameter c.
% The variable k is a vector of gains for each interval.

% Define plant transfer function in form 1/(s+a)(s^2 + bs + c)
% Controller k is proportional

a=6;b=4;

% The characteristic equation of the closed loop system is
% s^3 + (a+ b) s^2 + (ab+c) s + ac + k

total=0;

interval=[0 2 4 6 8 10];

for i=1:5
  for c = interval(i):0.02:interval(i+1)
    r = roots([1, (a+b), (a*b+c), (a*c+k(i))])';
    temp = (max(imag(r)) - 2)^2 + (max(real(r)) + 2)^2;
    total = total + temp;
  end
end

f=total;

return
```

```
%****************************************** end of file ****

%****************************************** file varint.m ****
function [f]=varint(k);

%  This function is an evaluation routine run from genesis to
%  evaluate the distance from the closed loop poles of a
%  system with a given gain to the desired closed loop
%  poles over a range of a varying parameter c.
%  The variable k is a vector of gains for each interval
%  and the four interval break points.

% Define plant transfer function in form 1/(s+a)(s^2 + bs + c)
% Controller k is proportional

a=6;b=4;

% The characteristic equation of the closed loop system is
% s^3 + (a+ b) s^2 + (ab+c) s + ac + k

total=0;

% sort the interval break points k(1)-k(4) in order from
% lowest to highest

for l=1:3
  for i= 1:3
    if k(i)>k(i+1)
      temp = k(i);
      k(i) = k(i+1);
      k(i+1)=temp;
    end
  end
end

interval=[0 k(1:4) 10];

for i=1:5
  for c = interval(i):0.02:interval(i+1)
    r = roots([1, (a+b), (a*b+c), (a*c+k(4+i))])';
    temp = (max(imag(r)) - 2)^2 + (max(real(r)) + 2)^2;
    total = total + temp;
```

```
  end
end

f=total;

return

%****************************************** end of file ****

%************************************** file varintn.m ****
function [f]=varintn(k);

%  This function is an evaluation routine run from genesis
%  to evaluate the distance from the closed loop poles of
%  a system with a given gain to the desired closed loop
%  poles over a range of a varying parameter c.
%  The variable k is a vector of gains for each interval
%  and the four interval break points. The first element
%  of k is the number of intervals (2:8), the next eight
%  elements are the break points (0:10), and the final
%  nine elements are the gains for each interval (-50:50)

% Define plant transfer function form 1/(s+a)(s^2 + bs + c)
% Controller k is proportional

a=6;b=4;

% The characteristic equation of the closed loop system is
% s^3 + (a+ b) s^2 + (ab+c) s + ac + k

total=0;

% sort the interval break points k(1)-k(4) in order from
% lowest to highest

int=[0 k(2:k(1)) 10];
s=length(int);

for l=1:3
  for i= 1:3
    if k(i)>k(i+1)
      temp = k(i);
```

```
      k(i) = k(i+1);
      k(i+1)=temp;
    end
  end
end

interval=[0 k(1:4) 10];

for i=1:k(1)
  for c = interval(i):0.02:interval(i+1)
    r = roots([1, (a+b), (a*b+c), (a*c+k(9+i))])';
    temp = (max(imag(r)) - 2)^2 + (max(real(r)) + 2)^2;
    total = total + temp;
  end
end

f=total;

return

%****************************************** end of file ****

%************************************** file sampcoef.m ****
function [f]=sampcoef(k);

%  This function is an evaluation routine run from genesis
%  to evaluate the distance from the closed loop poles of
%  a system with a given gain to the desired closed loop
%  poles over a range of a varying parameter c.
%  The variable k is a vector of the coefficients of a
%  polynomial function of the scheduling variable c.

% Define plant transfer function form 1/(s+a)(s^2 + bs + c)
% Controller k is proportional

a=6;b=4;

% The characteristic equation of the closed loop system is
% s^3 + (a+ b) s^2 + (ab+c) s + ac + k

total=0;
```

```
for c = 0:0.02:10
  k=polyval(coef,c);
  r = roots([1, (a+b), (a*b+c), (a*c+k)])';
  temp = (max(imag(r)) - 2)^2 + (max(real(r)) + 2)^2;
  total = total + temp;
  end
end

f=total;

return

%****************************************** end of file ****
```

B.2 F-18 Example Code

The *Matlab*© code for the F-18 example varies for each optimization example. First, the 'C' code that calls the *Matlab*© engine is presented. Next the code for evaluating the baseline design is presented. Next, for simplicity the code for only Cases I, II, V and VII are presented.

```
%************************************** file relerr.c ****
#include  "extern/include/engine.h"
#include  <stdio.h>
#include  <stdlib.h>
#include  <string.h>
#include  "global.h"

double eval(str, length, vect, genes)
char str[]; /* string representation */
int length; /* length of bit string */
double vect[]; /* floating point representation */
int genes; /* number of elements in vect */

{
  Matrix *v, *d;
register int i;
double sum, *Dreal;
sum = 0.0;

/* printf("here");
printf("%g\n",*vect); */
```

```
v = mxCreateFull(1,genes,REAL);
memcpy(mxGetPr(v),vect,genes*sizeof(double));
mxSetName(v,"V");

engPutMatrix(ep1, v);
engEvalString(ep1,"d=relerr(V)");
d = engGetMatrix(ep1,"d");


Dreal = mxGetPr(d);
/* printf("%g\n",*Dreal); */

mxFreeMatrix(v);
mxFreeMatrix(d);

return (*Dreal);


}
%****************************************** end of file ****

%****************************************** file relbase.m ****
% This program  calculates the open and closed loop dynamics of the F-18
% short period longitudinal dynamics and the Relative error of the closed
% inner loop for the chosen central flight condition.

clear numtemp dentemp
clear nump denp

F = -40;  Kf = [1  1];  G = .0247;

% Vector of dynamic pressure for 20 flight conditions
% (Original 12 + 8 more off design)

q=[47.4 68.5 100.1 158.4 189.9 255.0 301.1 355.0 426.4 496.0 557.0 ...
603.0 614.4 652.0 705.0 789.1 825.2 890.8 956.0 998.7]

% Calculate the scheduled parameters of the inner loop controller

N=-.312*q+461
M1=-.058*q+50.5
M2=-.006*q+8.11
```

```
% Calculate the inner loop controller state space form

l=1;
for i=1:20
  Akeq(i)=F-G*N(i);
  Bkeq(i,:)=Kf-G*[M1(i) M2(i)];
  Ckeq(i)=-N(i);
  Dkeq(i,:)=-[M1(i) M2(i)];
end

% The central controller is chosen to be the Am95h20

Akcen=Akeq(13);
Bkcen=Bkeq(13,:);
Ckcen=Ckeq(13);
Dkcen=Dkeq(13,:);

% Input the dynamics at the 20 flight conditons

Am3h26=[-.2296 .9931;.02436 -.2046];
Am5h40=[-.2423 .9964;-2.342 -.1737];
Am4h22=[-.4285 .9916;-.7473 -.3123]; %Am4h22    *3
Am6h30=[-.5088 .994;-1.131 -.2804];
Am4h6=[-.8018 .9847;-1.521 -.5944];
Am5h10=[-.8930 .9852;-4.1582 -.6873]; %Am5h10    *6
Am6h15=[-.9181 .9872;-6.2419 -.6920]; %Am6h15    *7
Am7h18_5=[-.9920 .9888;-7.8450 -.7525]; %Am7h18_5  *8
Am7h14=[-1.175 .9871;-8.458 -.8776];
Am6h2=[-1.4710 .9808;-11.5022 -1.0846]; %Am6h2     *10
Am8h14=[-2.1163 .9872;-32.6459 -1.1826]; %Am8h14    *11
Am8h12=[-1.562 .9862;-14.94 -1.132];
Am95h20=[-1.905 .9895;-33.88 -.9872];
Am8h10=[-1.675 .9853;-16.16 -1.212];
Am9h14=[-2.1163 .9872;-32.6459 -1.1826]; %Am9h14    *15
Am8h5=[-1.994 .9828;-19.44 -1.427];
Am9h10=[-2.452 .9856;-38.61 -1.34];
Am95h9=[-2.8375 .9855;-51.8325 -1.4037]; %Am95h9    *19
Am85h5=[-2.328 .9831;-30.44 -1.493];
Am9h5=[-2.911 .9835;-46.47 -1.553];
Acen=[-1.905 .9895;-33.88 -.9872];
C=[1 0;0 1];
```

```
D=[0;0];
Blong=[0;1];

%  The elevator actuator dynamics

numelev=[1/82.9^2 2*.068/82.9 1];
denelev=conv([1/36.4^2 2*.41/36.4 1],[1/105.3^2 2*.59/105.3 1]);
[aact,bact,cact,dact]=tf2ss(numelev,denelev);
%break
%  Combine  the actuator dynamics in the plant

[Ap1a,Bp1a,Cp1a,Dp1a]=series(aact,bact,cact,dact,Am3h26,Blong,C,D);
[Ap2a,Bp2a,Cp2a,Dp2a]=series(aact,bact,cact,dact,Am5h40,Blong,C,D);
[Ap3a,Bp3a,Cp3a,Dp3a]=series(aact,bact,cact,dact,Am4h22,Blong,C,D);
[Ap4a,Bp4a,Cp4a,Dp4a]=series(aact,bact,cact,dact,Am6h30,Blong,C,D);
[Ap5a,Bp5a,Cp5a,Dp5a]=series(aact,bact,cact,dact,Am4h6,Blong,C,D);
[Ap6a,Bp6a,Cp6a,Dp6a]=series(aact,bact,cact,dact,Am5h10,Blong,C,D);
[Ap7a,Bp7a,Cp7a,Dp7a]=series(aact,bact,cact,dact,Am6h15,Blong,C,D);
[Ap8a,Bp8a,Cp8a,Dp8a]=series(aact,bact,cact,dact,Am7h18_5,Blong,C,D);
[Ap9a,Bp9a,Cp9a,Dp9a]=series(aact,bact,cact,dact,Am7h14,Blong,C,D);
[Ap10a,Bp10a,Cp10a,Dp10a]=series(aact,bact,cact,dact,Am6h2,Blong,C,D);
[Ap11a,Bp11a,Cp11a,Dp11a]=series(aact,bact,cact,dact,Am8h14,Blong,C,D);
[Ap12a,Bp12a,Cp12a,Dp12a]=series(aact,bact,cact,dact,Am8h12,Blong,C,D);
[Ap13a,Bp13a,Cp13a,Dp13a]=series(aact,bact,cact,dact,Am95h20,Blong,C,D);
[Ap14a,Bp14a,Cp14a,Dp14a]=series(aact,bact,cact,dact,Am8h10,Blong,C,D);
[Ap15a,Bp15a,Cp15a,Dp15a]=series(aact,bact,cact,dact,Am9h14,Blong,C,D);
[Ap16a,Bp16a,Cp16a,Dp16a]=series(aact,bact,cact,dact,Am8h5,Blong,C,D);
[Ap17a,Bp17a,Cp17a,Dp17a]=series(aact,bact,cact,dact,Am9h10,Blong,C,D);
[Ap18a,Bp18a,Cp18a,Dp18a]=series(aact,bact,cact,dact,Am85h5,Blong,C,D);
[Ap19a,Bp19a,Cp19a,Dp19a]=series(aact,bact,cact,dact,Am95h9,Blong,C,D);
[Ap20a,Bp20a,Cp20a,Dp20a]=series(aact,bact,cact,dact,Am9h5,Blong,C,D);
[Ap0a,Bp0a,Cp0a,Dp0a]=series(aact,bact,cact,dact,Acen,Blong,C,D);

% Close the inner loop with the corresponding controller

[Ap1,Bp1,Cp1,Dp1]=feedback(Ap1a,Bp1a,Cp1a,Dp1a,...
Akeq(1),Bkeq(1,:),Ckeq(1),Dkeq(1,:),1);
[Ap2,Bp2,Cp2,Dp2]=feedback(Ap2a,Bp2a,Cp2a,Dp2a,...
Akeq(2),Bkeq(2,:),Ckeq(2),Dkeq(2,:),1);
[Ap3,Bp3,Cp3,Dp3]=feedback(Ap3a,Bp3a,Cp3a,Dp3a,...
Akeq(3),Bkeq(3,:),Ckeq(3),Dkeq(3,:),1);
[Ap4,Bp4,Cp4,Dp4]=feedback(Ap4a,Bp4a,Cp4a,Dp4a,...
```

```
Akeq(4),Bkeq(4,:),Ckeq(4),Dkeq(4,:),1);
[Ap5,Bp5,Cp5,Dp5]=feedback(Ap5a,Bp5a,Cp5a,Dp5a,...
Akeq(5),Bkeq(5,:),Ckeq(5),Dkeq(5,:),1);
[Ap6,Bp6,Cp6,Dp6]=feedback(Ap6a,Bp6a,Cp6a,Dp6a,...
Akeq(6),Bkeq(6,:),Ckeq(6),Dkeq(6,:),1);
[Ap7,Bp7,Cp7,Dp7]=feedback(Ap7a,Bp7a,Cp7a,Dp7a,...
Akeq(7),Bkeq(7,:),Ckeq(7),Dkeq(7,:),1);
[Ap8,Bp8,Cp8,Dp8]=feedback(Ap8a,Bp8a,Cp8a,Dp8a,...
Akeq(8),Bkeq(8,:),Ckeq(8),Dkeq(8,:),1);
[Ap9,Bp9,Cp9,Dp9]=feedback(Ap9a,Bp9a,Cp9a,Dp9a,...
Akeq(9),Bkeq(9,:),Ckeq(9),Dkeq(9,:),1);
[Ap10,Bp10,Cp10,Dp10]=feedback(Ap10a,Bp10a,Cp10a,Dp10a,...
Akeq(10),Bkeq(10,:),Ckeq(10),Dkeq(10,:),1);
[Ap11,Bp11,Cp11,Dp11]=feedback(Ap11a,Bp11a,Cp11a,Dp11a,...
Akeq(11),Bkeq(11,:),Ckeq(11),Dkeq(11,:),1);
[Ap12,Bp12,Cp12,Dp12]=feedback(Ap12a,Bp12a,Cp12a,Dp12a,...
Akeq(12),Bkeq(12,:),Ckeq(12),Dkeq(12,:),1);
[Ap13,Bp13,Cp13,Dp13]=feedback(Ap13a,Bp13a,Cp13a,Dp13a,...
Akeq(13),Bkeq(13,:),Ckeq(13),Dkeq(13,:),1);
[Ap14,Bp14,Cp14,Dp14]=feedback(Ap14a,Bp14a,Cp14a,Dp14a,...
Akeq(14),Bkeq(14,:),Ckeq(14),Dkeq(14,:),1);
[Ap15,Bp15,Cp15,Dp15]=feedback(Ap15a,Bp15a,Cp15a,Dp15a,...
Akeq(15),Bkeq(15,:),Ckeq(15),Dkeq(15,:),1);
[Ap16,Bp16,Cp16,Dp16]=feedback(Ap16a,Bp16a,Cp16a,Dp16a,...
Akeq(16),Bkeq(16,:),Ckeq(16),Dkeq(16,:),1);
[Ap17,Bp17,Cp17,Dp17]=feedback(Ap17a,Bp17a,Cp17a,Dp17a,...
Akeq(17),Bkeq(17,:),Ckeq(17),Dkeq(17,:),1);
[Ap18,Bp18,Cp18,Dp18]=feedback(Ap18a,Bp18a,Cp18a,Dp18a,...
Akeq(18),Bkeq(18,:),Ckeq(18),Dkeq(18,:),1);
[Ap19,Bp19,Cp19,Dp19]=feedback(Ap19a,Bp19a,Cp19a,Dp19a,...
Akeq(19),Bkeq(19,:),Ckeq(19),Dkeq(19,:),1);
[Ap20,Bp20,Cp20,Dp20]=feedback(Ap20a,Bp20a,Cp20a,Dp20a,...
Akeq(20),Bkeq(20,:),Ckeq(20),Dkeq(20,:),1);
[Ap0,Bp0,Cp0,Dp0]=feedback(Ap0a,Bp0a,Cp0a,Dp0a,...
Akcen,Bkcen,Ckcen,Dkcen,1);

[Ap1,Bp1,Cp1,Dp1]=feedback(Am3h26,Blong,C,D,...
Akeq(1),Bkeq(1,:),Ckeq(1),Dkeq(1,:),1);
[Ap2,Bp2,Cp2,Dp2]=feedback(Am5h40,Blong,C,D,...
Akeq(2),Bkeq(2,:),Ckeq(2),Dkeq(2,:),1);
[Ap3,Bp3,Cp3,Dp3]=feedback(Am4h22,Blong,C,D,...
Akeq(3),Bkeq(3,:),Ckeq(3),Dkeq(3,:),1);
```

```
[Ap4,Bp4,Cp4,Dp4]=feedback(Am6h30,Blong,C,D,...
Akeq(4),Bkeq(4,:),Ckeq(4),Dkeq(4,:),1);
[Ap5,Bp5,Cp5,Dp5]=feedback(Am4h6,Blong,C,D,...
Akeq(5),Bkeq(5,:),Ckeq(5),Dkeq(5,:),1);
[Ap6,Bp6,Cp6,Dp6]=feedback(Am5h10,Blong,C,D,...
Akeq(6),Bkeq(6,:),Ckeq(6),Dkeq(6,:),1);
[Ap7,Bp7,Cp7,Dp7]=feedback(Am6h15,Blong,C,D,...
Akeq(7),Bkeq(7,:),Ckeq(7),Dkeq(7,:),1);
[Ap8,Bp8,Cp8,Dp8]=feedback(Am7h18_5,Blong,C,D,...
Akeq(8),Bkeq(8,:),Ckeq(8),Dkeq(8,:),1);
[Ap9,Bp9,Cp9,Dp9]=feedback(Am7h14,Blong,C,D,...
Akeq(9),Bkeq(9,:),Ckeq(9),Dkeq(9,:),1);
[Ap10,Bp10,Cp10,Dp10]=feedback(Am6h2,Blong,C,D,...
Akeq(10),Bkeq(10,:),Ckeq(10),Dkeq(10,:),1);
[Ap11,Bp11,Cp11,Dp11]=feedback(Am8h14,Blong,C,D,...
Akeq(11),Bkeq(11,:),Ckeq(11),Dkeq(11,:),1);
[Ap12,Bp12,Cp12,Dp12]=feedback(Am8h12,Blong,C,D,...
Akeq(12),Bkeq(12,:),Ckeq(12),Dkeq(12,:),1);
[Ap13,Bp13,Cp13,Dp13]=feedback(Am95h20,Blong,C,D,...
Akeq(13),Bkeq(13,:),Ckeq(13),Dkeq(13,:),1);
[Ap14,Bp14,Cp14,Dp14]=feedback(Am8h10,Blong,C,D,...
Akeq(14),Bkeq(14,:),Ckeq(14),Dkeq(14,:),1);
[Ap15,Bp15,Cp15,Dp15]=feedback(Am9h14,Blong,C,D,...
Akeq(15),Bkeq(15,:),Ckeq(15),Dkeq(15,:),1);
[Ap16,Bp16,Cp16,Dp16]=feedback(Am8h5,Blong,C,D,...
Akeq(16),Bkeq(16,:),Ckeq(16),Dkeq(16,:),1);
[Ap17,Bp17,Cp17,Dp17]=feedback(Am9h10,Blong,C,D,...
Akeq(17),Bkeq(17,:),Ckeq(17),Dkeq(17,:),1);
[Ap18,Bp18,Cp18,Dp18]=feedback(Am85h5,Blong,C,D,...
Akeq(18),Bkeq(18,:),Ckeq(18),Dkeq(18,:),1);
[Ap19,Bp19,Cp19,Dp19]=feedback(Am95h9,Blong,C,D,...
Akeq(19),Bkeq(19,:),Ckeq(19),Dkeq(19,:),1);
[Ap20,Bp20,Cp20,Dp20]=feedback(Am9h5,Blong,C,D,...
Akeq(20),Bkeq(20,:),Ckeq(20),Dkeq(20,:),1);
[Ap0,Bp0,Cp0,Dp0]=feedback(Acen,Blong,C,D,...
Akcen,Bkcen,Ckcen,Dkcen,1);

w=logspace(-2,2,200);
C1=C(1,:);D1=0;
C2=C(2,:);D2=0;

% Compute open loop dynamics Pitch acceleration to angle of attack
```

```
[sv1o1]=sigma(Am3h26,Blong,C1,D1,w);
[sv2o1]=sigma(Am5h40,Blong,C1,D1,w);
[sv3o1]=sigma(Am4h22,Blong,C1,D1,w);
[sv4o1]=sigma(Am6h30,Blong,C1,D1,w);
[sv5o1]=sigma(Am4h6,Blong,C1,D1,w);
[sv6o1]=sigma(Am5h10,Blong,C1,D1,w);
[sv7o1]=sigma(Am6h15,Blong,C1,D1,w);
[sv8o1]=sigma(Am7h18_5,Blong,C1,D1,w);
[sv9o1]=sigma(Am7h14,Blong,C1,D1,w);
[sv10o1]=sigma(Am6h2,Blong,C1,D1,w);
[sv11o1]=sigma(Am8h14,Blong,C1,D1,w);
[sv12o1]=sigma(Am8h12,Blong,C1,D1,w);
[sv13o1]=sigma(Am95h20,Blong,C1,D1,w);
[sv14o1]=sigma(Am8h10,Blong,C1,D1,w);
[sv15o1]=sigma(Am9h14,Blong,C1,D1,w);
[sv16o1]=sigma(Am8h5,Blong,C1,D1,w);
[sv17o1]=sigma(Am9h10,Blong,C1,D1,w);
[sv18o1]=sigma(Am85h5,Blong,C1,D1,w);
[sv19o1]=sigma(Am95h9,Blong,C1,D1,w);
[sv20o1]=sigma(Am9h5,Blong,C1,D1,w);

% Compute open loop dynamics Pitch acceleration to pitch rate

[sv1o2]=sigma(Am3h26,Blong,C2,D2,w);
[sv2o2]=sigma(Am5h40,Blong,C2,D2,w);
[sv3o2]=sigma(Am4h22,Blong,C2,D2,w);
[sv4o2]=sigma(Am6h30,Blong,C2,D2,w);
[sv5o2]=sigma(Am4h6,Blong,C2,D2,w);
[sv6o2]=sigma(Am5h10,Blong,C2,D2,w);
[sv7o2]=sigma(Am6h15,Blong,C2,D2,w);
[sv8o2]=sigma(Am7h18_5,Blong,C2,D2,w);
[sv9o2]=sigma(Am7h14,Blong,C2,D2,w);
[sv10o2]=sigma(Am6h2,Blong,C2,D2,w);
[sv11o2]=sigma(Am8h14,Blong,C2,D2,w);
[sv12o2]=sigma(Am8h12,Blong,C2,D2,w);
[sv13o2]=sigma(Am95h20,Blong,C2,D2,w);
[sv14o2]=sigma(Am8h10,Blong,C2,D2,w);
[sv15o2]=sigma(Am9h14,Blong,C2,D2,w);
[sv16o2]=sigma(Am8h5,Blong,C2,D2,w);
[sv17o2]=sigma(Am9h10,Blong,C2,D2,w);
[sv18o2]=sigma(Am85h5,Blong,C2,D2,w);
```

```
[sv19o2]=sigma(Am95h9,Blong,C2,D2,w);
[sv20o2]=sigma(Am9h5,Blong,C2,D2,w);

% Closed loop singular value analysis (pitch acceleration to pitch rate)

[sv1p2]=sigma(Ap1,Bp1,Cp1(2,:),Dp1(2,:),w);
[sv2p2]=sigma(Ap2,Bp2,Cp2(2,:),Dp2(2,:),w);
[sv3p2]=sigma(Ap3,Bp3,Cp3(2,:),Dp3(2,:),w);
[sv4p2]=sigma(Ap4,Bp4,Cp4(2,:),Dp4(2,:),w);
[sv5p2]=sigma(Ap5,Bp5,Cp5(2,:),Dp5(2,:),w);
[sv6p2]=sigma(Ap6,Bp6,Cp6(2,:),Dp6(2,:),w);
[sv7p2]=sigma(Ap7,Bp7,Cp7(2,:),Dp7(2,:),w);
[sv8p2]=sigma(Ap8,Bp8,Cp8(2,:),Dp8(2,:),w);
[sv9p2]=sigma(Ap9,Bp9,Cp9(2,:),Dp9(2,:),w);
[sv10p2]=sigma(Ap10,Bp10,Cp10(2,:),Dp10(2,:),w);
[sv11p2]=sigma(Ap11,Bp11,Cp11(2,:),Dp11(2,:),w);
[sv12p2]=sigma(Ap12,Bp12,Cp12(2,:),Dp12(2,:),w);
[sv13p2]=sigma(Ap13,Bp13,Cp13(2,:),Dp13(2,:),w);
[sv14p2]=sigma(Ap14,Bp14,Cp14(2,:),Dp14(2,:),w);
[sv15p2]=sigma(Ap15,Bp15,Cp15(2,:),Dp15(2,:),w);
[sv16p2]=sigma(Ap16,Bp16,Cp16(2,:),Dp16(2,:),w);
[sv17p2]=sigma(Ap17,Bp17,Cp17(2,:),Dp17(2,:),w);
[sv18p2]=sigma(Ap18,Bp18,Cp18(2,:),Dp18(2,:),w);
[sv19p2]=sigma(Ap19,Bp19,Cp19(2,:),Dp19(2,:),w);
[sv20p2]=sigma(Ap20,Bp20,Cp20(2,:),Dp20(2,:),w);

% Reduce the closed loop system to SISO (pitch acceleration to angle of attack)

Cp1=Cp1(1,:);Dp1=Dp1(1,:);
Cp2=Cp2(1,:);Dp2=Dp2(1,:);
Cp3=Cp3(1,:);Dp3=Dp3(1,:);
Cp4=Cp4(1,:);Dp4=Dp4(1,:);
Cp5=Cp5(1,:);Dp5=Dp5(1,:);
Cp6=Cp6(1,:);Dp6=Dp6(1,:);
Cp7=Cp7(1,:);Dp7=Dp7(1,:);
Cp8=Cp8(1,:);Dp8=Dp8(1,:);
Cp9=Cp9(1,:);Dp9=Dp9(1,:);
Cp10=Cp10(1,:);Dp10=Dp10(1,:);
Cp11=Cp11(1,:);Dp11=Dp11(1,:);
Cp12=Cp12(1,:);Dp12=Dp12(1,:);
Cp13=Cp13(1,:);Dp13=Dp13(1,:);
Cp14=Cp14(1,:);Dp14=Dp14(1,:);
```

```
Cp15=Cp15(1,:);Dp15=Dp15(1,:);
Cp16=Cp16(1,:);Dp16=Dp16(1,:);
Cp17=Cp17(1,:);Dp17=Dp17(1,:);
Cp18=Cp18(1,:);Dp18=Dp18(1,:);
Cp19=Cp19(1,:);Dp19=Dp19(1,:);
Cp20=Cp20(1,:);Dp20=Dp20(1,:);
Cp0=Cp0(1,:);Dp0=Dp0(1,:);

% Closed loop singular value analysis (pitch acceleration to angle of attack)

[sv1p]=sigma(Ap1,Bp1,Cp1,Dp1,w);
[sv2p]=sigma(Ap2,Bp2,Cp2,Dp2,w);
[sv3p]=sigma(Ap3,Bp3,Cp3,Dp3,w);
[sv4p]=sigma(Ap4,Bp4,Cp4,Dp4,w);
[sv5p]=sigma(Ap5,Bp5,Cp5,Dp5,w);
[sv6p]=sigma(Ap6,Bp6,Cp6,Dp6,w);
[sv7p]=sigma(Ap7,Bp7,Cp7,Dp7,w);
[sv8p]=sigma(Ap8,Bp8,Cp8,Dp8,w);
[sv9p]=sigma(Ap9,Bp9,Cp9,Dp9,w);
[sv10p]=sigma(Ap10,Bp10,Cp10,Dp10,w);
[sv11p]=sigma(Ap11,Bp11,Cp11,Dp11,w);
[sv12p]=sigma(Ap12,Bp12,Cp12,Dp12,w);
[sv13p]=sigma(Ap13,Bp13,Cp13,Dp13,w);
[sv14p]=sigma(Ap14,Bp14,Cp14,Dp14,w);
[sv15p]=sigma(Ap15,Bp15,Cp15,Dp15,w);
[sv16p]=sigma(Ap16,Bp16,Cp16,Dp16,w);
[sv17p]=sigma(Ap17,Bp17,Cp17,Dp17,w);
[sv18p]=sigma(Ap18,Bp18,Cp18,Dp18,w);
[sv19p]=sigma(Ap19,Bp19,Cp19,Dp19,w);
[sv20p]=sigma(Ap20,Bp20,Cp20,Dp20,w);

%  Convert state space to transfer function for computation of relative error

[nump(1,:),denp(1,:)]=ss2tf(Ap1,Bp1,Cp1,Dp1);
[nump(2,:),denp(2,:)]=ss2tf(Ap2,Bp2,Cp2,Dp2);
[nump(3,:),denp(3,:)]=ss2tf(Ap3,Bp3,Cp3,Dp3);
[nump(4,:),denp(4,:)]=ss2tf(Ap4,Bp4,Cp4,Dp4);
[nump(5,:),denp(5,:)]=ss2tf(Ap5,Bp5,Cp5,Dp5);
[nump(6,:),denp(6,:)]=ss2tf(Ap6,Bp6,Cp6,Dp6);
[nump(7,:),denp(7,:)]=ss2tf(Ap7,Bp7,Cp7,Dp7);
[nump(8,:),denp(8,:)]=ss2tf(Ap8,Bp8,Cp8,Dp8);
[nump(9,:),denp(9,:)]=ss2tf(Ap9,Bp9,Cp9,Dp9);
```

```
[nump(10,:),denp(10,:)]=ss2tf(Ap10,Bp10,Cp10,Dp10);
[nump(11,:),denp(11,:)]=ss2tf(Ap11,Bp11,Cp11,Dp11);
[nump(12,:),denp(12,:)]=ss2tf(Ap12,Bp12,Cp12,Dp12);
[nump(13,:),denp(13,:)]=ss2tf(Ap13,Bp13,Cp13,Dp13);
[nump(14,:),denp(14,:)]=ss2tf(Ap14,Bp14,Cp14,Dp14);
[nump(15,:),denp(15,:)]=ss2tf(Ap15,Bp15,Cp15,Dp15);
[nump(16,:),denp(16,:)]=ss2tf(Ap16,Bp16,Cp16,Dp16);
[nump(17,:),denp(17,:)]=ss2tf(Ap17,Bp17,Cp17,Dp17);
[nump(18,:),denp(18,:)]=ss2tf(Ap18,Bp18,Cp18,Dp18);
[nump(19,:),denp(19,:)]=ss2tf(Ap19,Bp19,Cp19,Dp19);
[nump(20,:),denp(20,:)]=ss2tf(Ap20,Bp20,Cp20,Dp20);
[nump0,denp0]=ss2tf(Ap0,Bp0,Cp0,Dp0);

% Reduce the transfer fuction to essential parts (chop off added zeros)
nump=nump(:,3:4)
nump0=nump0(3:4)
nump0=nump(13,:);denp0=denp(13,:);
% Compute Relative Error  (Po-P)Po^-1

for i=1:20

  numtemp(i,:)=conv(nump(i,:),denp0)-conv(nump0,denp(i,:));
  dentemp(i,:)=conv(denp(i,:),nump0);

end


% Convert back to state space for singular value analysis

[a1,b1,c1,d1]=tf2ss(numtemp(1,:),dentemp(1,:));
[a2,b2,c2,d2]=tf2ss(numtemp(2,:),dentemp(2,:));
[a3,b3,c3,d3]=tf2ss(numtemp(3,:),dentemp(3,:));
[a4,b4,c4,d4]=tf2ss(numtemp(4,:),dentemp(4,:));
[a5,b5,c5,d5]=tf2ss(numtemp(5,:),dentemp(5,:));
[a6,b6,c6,d6]=tf2ss(numtemp(6,:),dentemp(6,:));
[a7,b7,c7,d7]=tf2ss(numtemp(7,:),dentemp(7,:));
[a8,b8,c8,d8]=tf2ss(numtemp(8,:),dentemp(8,:));
[a9,b9,c9,d9]=tf2ss(numtemp(9,:),dentemp(9,:));
[a10,b10,c10,d10]=tf2ss(numtemp(10,:),dentemp(10,:));
[a11,b11,c11,d11]=tf2ss(numtemp(11,:),dentemp(11,:));
[a12,b12,c12,d12]=tf2ss(numtemp(12,:),dentemp(12,:));
[a13,b13,c13,d13]=tf2ss(numtemp(13,:),dentemp(13,:));
```

```
[a14,b14,c14,d14]=tf2ss(numtemp(15,:),dentemp(15,:));
[a15,b15,c15,d15]=tf2ss(numtemp(15,:),dentemp(15,:));
[a16,b16,c16,d16]=tf2ss(numtemp(16,:),dentemp(16,:));
[a17,b17,c17,d17]=tf2ss(numtemp(17,:),dentemp(17,:));
[a18,b18,c18,d18]=tf2ss(numtemp(18,:),dentemp(18,:));
[a19,b19,c19,d19]=tf2ss(numtemp(19,:),dentemp(19,:));
[a20,b20,c20,d20]=tf2ss(numtemp(20,:),dentemp(20,:));

% Singular value analysis

[sv1]=sigma(a1,b1,c1,d1,w);
[sv2]=sigma(a2,b2,c2,d2,w);
[sv3]=sigma(a3,b3,c3,d3,w);
[sv4]=sigma(a4,b4,c4,d4,w);
[sv5]=sigma(a5,b5,c5,d5,w);
[sv6]=sigma(a6,b6,c6,d6,w);
[sv7]=sigma(a7,b7,c7,d7,w);
[sv8]=sigma(a8,b8,c8,d8,w);
[sv9]=sigma(a9,b9,c9,d9,w);
[sv10]=sigma(a10,b10,c10,d10,w);
[sv11]=sigma(a11,b11,c11,d11,w);
[sv12]=sigma(a12,b12,c12,d12,w);
[sv13]=sigma(a13,b13,c13,d13,w);
[sv14]=sigma(a14,b14,c14,d14,w);
[sv15]=sigma(a15,b15,c15,d15,w);
[sv16]=sigma(a16,b16,c16,d16,w);
[sv17]=sigma(a17,b17,c17,d17,w);
[sv18]=sigma(a18,b18,c18,d18,w);
[sv19]=sigma(a19,b19,c19,d19,w);
[sv20]=sigma(a20,b20,c20,d20,w);


j=max(sv1)+max(sv2)+max(sv3)+max(sv4)+max(sv5)+max(sv6)+max(sv7)+max(sv8)+...
max(sv9)+max(sv10)+max(sv11)+max(sv12)+max(sv13)+max(sv14)+max(sv15)+...
max(sv16)+max(sv17)+max(sv18)+max(sv19)+max(sv20)
toc

figure(1)
loglog(w,sv1,w,sv2,w,sv3,w,sv4,w,sv5,w,sv6,w,sv7,w,sv8,w,sv9,w,sv10,w,sv11,...
w,sv12,w,sv13,w,sv14,w,sv15,w,sv16,w,sv17,w,sv18,w,sv19,w,sv20)
%title('Baseline Closed Inner Loop Relative Error')
xlabel('w (rad/sec)')
```

```
ylabel('Singular Value')
axis([.01 100 .0001 1])
print -deps relbase

figure(2)
loglog(w,sv1p,w,sv2p,w,sv3p,w,sv4p,w,sv5p,w,sv6p,w,sv7p,w,sv8p,w,sv9p,...
w,sv10p,w,sv11p,w,sv12p,w,sv13p,w,sv14p,w,sv15p,w,sv16p,w,sv17p,w,sv18p,...
w,sv19p,w,sv20p)
xlabel('w (rad/sec)')
ylabel('Singular Value')
%title('Pitch Acceleration to Angle of Attack Closed Inner-Loop Dynamics')
axis([.01 100 .0001 .1])
print -deps basecla

figure(3)
loglog(w,sv1o1,w,sv2o1,w,sv3o1,w,sv4o1,w,sv5o1,w,sv6o1,w,sv7o1,w,sv8o1,...
w,sv9o1,w,sv10o1,w,sv11o1,w,sv12o1,w,sv13o1,w,sv14o1,w,sv15o1,w,sv16o1,...
w,sv17o1,w,sv18o1,w,sv19o1,w,sv20o1)
xlabel('w (rad/sec)')
ylabel('Singular Value')
%title('Pitch Acceleration to Angle of Attack Open Inner-Loop Dynamics')
print -deps olalpha

figure(4)
loglog(w,sv1o2,w,sv2o2,w,sv3o2,w,sv4o2,w,sv5o2,w,sv6o2,w,sv7o2,w,sv8o2,...
w,sv9o2,w,sv10o2,w,sv11o2,w,sv12o2,w,sv13o2,w,sv14o2,w,sv15o2,w,sv16o2,...
w,sv17o2,w,sv18o2,w,sv19o2,w,sv20o2)
xlabel('w (rad/sec)')
ylabel('Singular Value')
%title('Pitch Acceleration to Pitch Rate Open Inner-Loop Dynamics')
print -deps olpitch

figure(5)
loglog(w,sv1p2,w,sv2p2,w,sv3p2,w,sv4p2,w,sv5p2,w,sv6p2,w,sv7p2,w,sv8p2,...
w,sv9p2,w,sv10p2,w,sv11p2,w,sv12p2,w,sv13p2,w,sv14p2,w,sv15p2,w,sv16p2,...
w,sv17p2,w,sv18p2,w,sv19p2,w,sv20p2)
xlabel('w (rad/sec)')
ylabel('Singular Value')
%title('Pitch Acceleration to Pitch Rate Closed Inner-Loop Dynamics')
axis([.01 100 .001 1])
print -deps baseclp
```

```
% Load the outer loop controller
longout

% Combine the outer loop controller and the Closed inner loop

[Aoo1,Boo1,Coo1,Doo1]=series(Akoutlow,Bkoutlow,Ckoutlow,Dkoutlow,...
Ap1,Bp1,Cp1,Dp1);
[Aoo2,Boo2,Coo2,Doo2]=series(Akoutlow,Bkoutlow,Ckoutlow,Dkoutlow,...
Ap2,Bp2,Cp2,Dp2);
[Aoo3,Boo3,Coo3,Doo3]=series(Akoutlow,Bkoutlow,Ckoutlow,Dkoutlow,...
Ap3,Bp3,Cp3,Dp3);
[Aoo4,Boo4,Coo4,Doo4]=series(Akoutlow,Bkoutlow,Ckoutlow,Dkoutlow,...
Ap4,Bp4,Cp4,Dp4);
[Aoo5,Boo5,Coo5,Doo5]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap5,Bp5,Cp5,Dp5);
[Aoo6,Boo6,Coo6,Doo6]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap6,Bp6,Cp6,Dp6);
[Aoo7,Boo7,Coo7,Doo7]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap7,Bp7,Cp7,Dp7);
[Aoo8,Boo8,Coo8,Doo8]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap8,Bp8,Cp8,Dp8);
[Aoo9,Boo9,Coo9,Doo9]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap9,Bp9,Cp9,Dp9);
[Aoo10,Boo10,Coo10,Doo10]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap10,Bp10,Cp10,Dp10);
[Aoo11,Boo11,Coo11,Doo11]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap11,Bp11,Cp11,Dp11);
[Aoo12,Boo12,Coo12,Doo12]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap12,Bp12,Cp12,Dp12);
[Aoo13,Boo13,Coo13,Doo13]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap13,Bp13,Cp13,Dp13);
[Aoo14,Boo14,Coo14,Doo14]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap14,Bp14,Cp14,Dp14);
[Aoo15,Boo15,Coo15,Doo15]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap15,Bp15,Cp15,Dp15);
[Aoo16,Boo16,Coo16,Doo16]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap16,Bp16,Cp16,Dp16);
[Aoo17,Boo17,Coo17,Doo17]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap17,Bp17,Cp17,Dp17);
[Aoo18,Boo18,Coo18,Doo18]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap18,Bp18,Cp18,Dp18);
[Aoo19,Boo19,Coo19,Doo19]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
```

```
Ap19,Bp19,Cp19,Dp19);
[Aoo20,Boo20,Coo20,Doo20]=series(Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh,...
Ap20,Bp20,Cp20,Dp20);

% Close the outer loop with negative feedback

[Ao1,Bo1,Co1,Do1]=cloop(Aoo1,Boo1,Coo1,Doo1,-1);
[Ao2,Bo2,Co2,Do2]=cloop(Aoo2,Boo2,Coo2,Doo1,-1);
[Ao3,Bo3,Co3,Do3]=cloop(Aoo3,Boo3,Coo3,Doo3,-1);
[Ao4,Bo4,Co4,Do4]=cloop(Aoo4,Boo4,Coo4,Doo4,-1);
[Ao5,Bo5,Co5,Do5]=cloop(Aoo5,Boo5,Coo5,Doo5,-1);
[Ao6,Bo6,Co6,Do6]=cloop(Aoo6,Boo6,Coo6,Doo6,-1);
[Ao7,Bo7,Co7,Do7]=cloop(Aoo7,Boo7,Coo7,Doo7,-1);
[Ao8,Bo8,Co8,Do8]=cloop(Aoo8,Boo8,Coo8,Doo8,-1);
[Ao9,Bo9,Co9,Do9]=cloop(Aoo9,Boo9,Coo9,Doo9,-1);
[Ao10,Bo10,Co10,Do10]=cloop(Aoo10,Boo10,Coo10,Doo10,-1);
[Ao11,Bo11,Co11,Do11]=cloop(Aoo11,Boo11,Coo11,Doo11,-1);
[Ao12,Bo12,Co12,Do12]=cloop(Aoo12,Boo12,Coo12,Doo12,-1);
[Ao13,Bo13,Co13,Do13]=cloop(Aoo13,Boo13,Coo13,Doo13,-1);
[Ao14,Bo14,Co14,Do14]=cloop(Aoo14,Boo14,Coo14,Doo14,-1);
[Ao15,Bo15,Co15,Do15]=cloop(Aoo15,Boo15,Coo15,Doo15,-1);
[Ao16,Bo16,Co16,Do16]=cloop(Aoo16,Boo16,Coo16,Doo16,-1);
[Ao17,Bo17,Co17,Do17]=cloop(Aoo17,Boo17,Coo17,Doo17,-1);
[Ao18,Bo18,Co18,Do18]=cloop(Aoo18,Boo18,Coo18,Doo18,-1);
[Ao19,Bo19,Co19,Do19]=cloop(Aoo19,Boo19,Coo19,Doo19,-1);
[Ao20,Bo20,Co20,Do20]=cloop(Aoo20,Boo20,Coo20,Doo20,-1);

% Get step response of closed outer loop
t=0:.1:5;
[y1,x1]=step(Ao1,Bo1,Co1,Do1,1,t);
[y2,x2]=step(Ao2,Bo2,Co2,Do2,1,t);
[y3,x3]=step(Ao3,Bo3,Co3,Do3,1,t);
[y4,x4]=step(Ao4,Bo4,Co4,Do4,1,t);
[y5,x5]=step(Ao5,Bo5,Co5,Do5,1,t);
[y6,x6]=step(Ao6,Bo6,Co6,Do6,1,t);
[y7,x7]=step(Ao7,Bo7,Co7,Do7,1,t);
[y8,x8]=step(Ao8,Bo8,Co8,Do8,1,t);
[y9,x9]=step(Ao9,Bo9,Co9,Do9,1,t);
[y10,x10]=step(Ao10,Bo10,Co10,Do10,1,t);
[y11,x11]=step(Ao11,Bo11,Co11,Do11,1,t);
[y12,x12]=step(Ao12,Bo12,Co12,Do12,1,t);
[y13,x13]=step(Ao13,Bo13,Co13,Do13,1,t);
```

```
[y14,x14]=step(Ao14,Bo14,Co14,Do14,1,t);
[y15,x15]=step(Ao15,Bo15,Co15,Do15,1,t);
[y16,x16]=step(Ao16,Bo16,Co16,Do16,1,t);
[y17,x17]=step(Ao17,Bo17,Co17,Do17,1,t);
[y18,x18]=step(Ao18,Bo18,Co18,Do18,1,t);
[y19,x19]=step(Ao19,Bo19,Co19,Do19,1,t);
[y20,x20]=step(Ao20,Bo20,Co20,Do20,1,t);
e=sum(abs(sum([ (y1-y13),(y1-y13),(y2-y13),(y3-y13),(y4-y13),(y5-y13),(y6-y13),(y7-
(y8-y13),(y9-y13),(y10-y13),(y11-y13),(y12-y13),(y14-y13),(y15-y13),...
(y16-y13),(y17-y13),(y18-y13),(y19-y13),(y20-y13)])))
figure(6)
plot(t,y1,t,y2,t,y3,t,y4,t,y5,t,y6,t,y7,t,y8,t,y9,t,y10,t,y11,t,y12,...
t,y13,t,y14,t,y15,t,y16,t,y17,t,y18,t,y19,t,y20)
xlabel('Time (sec)')
ylabel('Amplitude')
grid
axis([0 5 0 1.3])
print -deps timebase
%****************************************** end of file ****

%****************************************** file relerr.m ****
function j=relerr(k)

% This function will take the given minimal order Hinf
% controller Keq and calculate the relative error to the
% chosen central controller

clear numtemp dentemp nump denp
SV=[];j=0;
F = -40;  Kf = [1  1];  G = .0247;

l=1;
for i=1:4
  Akeq(i)=F-G*k(l);
  Bkeq(i,:)=Kf-G*[k(l+1) k(l+2)];
  Ckeq(i)=-k(l);
  Dkeq(i,:)=-[k(l+1) k(l+2)];
  l=l+3;
end

Akcen=Akeq(3);
Bkcen=Bkeq(3,:);
```

```
Ckcen=Ckeq(3);
Dkcen=Dkeq(3,:);

% Input the dynamics at the 20 flight conditions

numSYS=20;
Along=[47.4,0;[-.2296 .9931;.02436 -.2046];   %Am3h26
68.5,0;[-.2423 .9964;-2.342 -.1737]; %Am5h40
100.1,0;[-.4285 .9916;-.7473 -.3123]; %Am4h22    *3
158.4,0;[-.5088 .994;-1.131 -.2804]; %Am6h30
189.9,0;[-.8018 .9847;-1.521 -.5944]; %Am4h6
255.0,0;[-.8930 .9852;-4.1582 -.6873]; %Am5h10    *6
301.1,0;[-.9181 .9872;-6.2419 -.6920]; %Am6h15    *7
355.0,0;[-.9920 .9888;-7.8450 -.7525]; %Am7h18_5  *8
426.4,0;[-1.175 .9871;-8.458 -.8776]; %Am7h14
496.0,0;[-1.4710 .9808;-11.5022 -1.0846]; %Am6h2     *10
557.0,0;[-2.1163 .9872;-32.6459 -1.1826]; %Am8h14    *11
603.0,0;[-1.562 .9862;-14.94 -1.132]; %Am8h12;
614.4,0;[-1.905 .9895;-33.88 -.9872]; %Am95h20;
652.0,0;[-1.675 .9853;-16.16 -1.212]; %Am8h10;
705.0,0;[-2.1163 .9872;-32.6459 -1.1826]; %Am9h14    *15
789.1,0;[-1.994 .9828;-19.44 -1.427]; %Am8h5;
825.2,0;[-2.452 .9856;-38.61 -1.34]; %Am9h10;
890.8,0;[-2.328 .9831;-30.44 -1.493] %Am85h5;
956.0,0;[-2.8375 .9855;-51.8325 -1.4037]; %Am95h9    *19
998.7,0;[-2.911 .9835;-46.47 -1.553]; %Am9h5;
inf,0];
C=[1 0;0 1];
D=[0;0];
Blong=[0;1];

w=logspace(-2,2,50);
int=[250 500 750 1000];
r=1;i=1;count=1;
for  i = 1:4
  while Along(r,1) < int(i)
    [Ap,Bp,Cp,Dp]=feedback(Along(r+1:r+2,:),Blong,C,D,Akeq(i),..
Bkeq(i,:),Ckeq(i),Dkeq(i,:),1);
    [nump(count,:),denp(count,:)]=ss2tf(Ap,Bp,Cp(1,:),Dp(1,:));
    [svcla]=sigma(Ap,Bp,Cp(1,:),Dp(1,:),w);
    SVcla=[SVcla;svcla];
    [svclp]=sigma(Ap,Bp,Cp(2,:),Dp(2,:),w);
```

```
    SVclp=[SVclp;svclp];
    r=r+3;
    count=count+1;
  end;
  i=i+1;
end;


% Reduce the transfer function to essential parts
% (chop off added zeros)
nump=nump(:,3:4);
nump0=nump(13,:);
denp0=denp(13,:);

% Compute Relative Error  (Po-P)Po^-1

for z=1:20
  numtemp=conv(nump(z,:),denp0)-conv(nump0,denp(z,:));
  dentemp=conv(denp(z,:),nump0);
  [a,b,c,d]=tf2ss(numtemp,dentemp);
  [sv]=sigma(a,b,c,d,w);
  SV=[SV;sv];
  j=j+max(sv);
end

return
%****************************************** end of file ****

%************************************** file relend.m ****
function j=relend(k)

% This function will take the given minimal order Hinf
% controller Keq and calculate the relative error to the
% chosen central controller

numtemp=[];dentemp=[];numSYS=0;Ap=[];Bp=[];Cp=[];Dp=[];j=[0];

n=45;
F = -40;  Kf = [1  1];  G = .0247;
l=4;
for i=1:4
  Akeq(i)=F-G*k(l);
```

```
  Bkeq(i,:)=Kf-G*[k(l+1) k(l+2)];
  Ckeq(i)=-k(l);
  Dkeq(i,:)=-[k(l+1) k(l+2)];
  l=l+3;
end

%  Following are the flight conditions used in the
%  evaluation of relative error.

A=[47.4,0;[-.2296 .9931;.02436 -.2046];   %Am3h26
68.5,0;[-.2423 .9964;-2.342 -.1737]; %Am5h40
100.1,0;[-.4285 .9916;-.7473 -.3123]; %Am4h22
158.4,0;[-.5088 .994;-1.131 -.2804]; %Am6h30
189.9,0;[-.8018 .9847;-1.521 -.5944]; %Am4h6
255.0,0;[-.8930 .9852;-4.1582 -.6873]; %Am5h10
301.1,0;[-.9181 .9872;-6.2419 -.6920]; %Am6h15
355.0,0;[-.9920 .9888;-7.8450 -.7525]; %Am7h18_5
426.4,0;[-1.175 .9871;-8.458 -.8776]; %Am7h14
496.0,0;[-1.4710 .9808;-11.5022 -1.0846]; %Am6h2
557.0,0;[-2.1163 .9872;-32.6459 -1.1826]; %Am8h14
603.0,0;[-1.562 .9862;-14.94 -1.132]; %Am8h12;
614.4,0;[-1.905 .9895;-33.88 -.9872]; %Am95h20;
652.0,0;[-1.675 .9853;-16.16 -1.212]; %Am8h10;
705.0,0;[-2.1163 .9872;-32.6459 -1.1826]; %Am9h14
789.1,0;[-1.994 .9828;-19.44 -1.427]; %Am8h5;
825.2,0;[-2.452 .9856;-38.61 -1.34]; %Am9h10;
890.8,0;[-2.328 .9831;-30.44 -1.493] %Am85h5;
956.0,0;[-2.8375 .9855;-51.8325 -1.4037]; %Am95h9
998.7,0;[-2.911 .9835;-46.47 -1.553]; %Am9h5;
inf,0];

C=[1 0;0 1];
D=[0;0];
Blong=[0;1];

% note i = interval number

int=[k(1:3) 1000];
s=length(int);

for l=1:(s-1)
  for i= 1:(s-1)
```

```
    if int(i)>int(i+1)
      temp = int(i);
      int(i) = int(i+1);
      int(i+1)=temp;
    end
  end
end


r=1;i=1;
while i <= 4
  while A(r,1)< int(i)
    numSYS=numSYS+1;
    [Ap,Bp,Cp,Dp]=feedback(A(r+1:r+2,:),Blong,C,D,...
Akeq(i),Bkeq(i,:),Ckeq(i),Dkeq(i,:),1);
    [nump(numSYS,:),denp(numSYS,:)]=ss2tf(Ap,Bp,Cp(1,:),Dp(1,:));
    r=r+3;
  end;
  i=i+1;
end;

w=logspace(-2,2,n);

nump0=nump(13,:);
denp0=denp(13,:);

for z=1:numSYS
  numtemp=conv(nump(z,:),denp0)-conv(nump0,denp(z,:));
  dentemp=conv(denp(z,:),nump0);
  numtemp=numtemp(:,3:7);
  dentemp=dentemp(:,3:7);
  [a,b,c,d]=tf2ss(numtemp,dentemp);
  [sv]=sigma(a,b,c,d,w);
  j=j+max(sv);
end

return
%****************************************** end of file ****

%****************************************** file relnum.m ****
function j=relnum(k)
```

```
% This function will take the given minimal order Hinf
% controller Keq and calculate the relative error to
% the chosen central controller

numtemp=[];dentemp=[];numSYS=0;j=[0];

n=45;
F = -40;  Kf = [1  1];  G = .0247;

% Calculate the controller for each interval
% Note i = interval number

l=10;
for i=1:k(1)
  Akeq(i)=F-G*k(l);
  Bkeq(i,:)=Kf-G*[k(l+1) k(l+2)];
  Ckeq(i)=-k(l);
  Dkeq(i,:)=-[k(l+1) k(l+2)];
  l=l+3;
end

% Following are the flight conditions used in the evaluation
% of relative error.

A=[47.4,0;[-.2296 .9931;.02436 -.2046];   %Am3h26
68.5,0;[-.2423 .9964;-2.342 -.1737]; %Am5h40
100.1,0;[-.4285 .9916;-.7473 -.3123]; %Am4h22
158.4,0;[-.5088 .994;-1.131 -.2804]; %Am6h30
189.9,0;[-.8018 .9847;-1.521 -.5944]; %Am4h6
255.0,0;[-.8930 .9852;-4.1582 -.6873]; %Am5h10
301.1,0;[-.9181 .9872;-6.2419 -.6920]; %Am6h15
355.0,0;[-.9920 .9888;-7.8450 -.7525]; %Am7h18_5
426.4,0;[-1.175 .9871;-8.458 -.8776]; %Am7h14
496.0,0;[-1.4710 .9808;-11.5022 -1.0846]; %Am6h2
557.0,0;[-2.1163 .9872;-32.6459 -1.1826]; %Am8h14
603.0,0;[-1.562 .9862;-14.94 -1.132]; %Am8h12;
614.4,0;[-1.905 .9895;-33.88 -.9872]; %Am95h20;
652.0,0;[-1.675 .9853;-16.16 -1.212]; %Am8h10;
705.0,0;[-2.1163 .9872;-32.6459 -1.1826]; %Am9h14
789.1,0;[-1.994 .9828;-19.44 -1.427]; %Am8h5;
825.2,0;[-2.452 .9856;-38.61 -1.34]; %Am9h10;
890.8,0;[-2.328 .9831;-30.44 -1.493] %Am85h5;
```

```
956.0,0;[-2.8375 .9855;-51.8325 -1.4037]; %Am95h9
998.7,0;[-2.911 .9835;-46.47 -1.553]; %Am9h5;
inf,0];

C=[1 0;0 1];
D=[0;0];
Blong=[0;1];

% Sort the interval break points from lowest to highest

int=[k(2:k(1)),1000];
s=length(int);

for l=1:(s-1)
  for i= 1:(s-1)
    if int(i)>int(i+1)
      temp = int(i);
      int(i) = int(i+1);
      int(i+1)=temp;
    end
  end
end

% Check the dynamic pressure of the flight condition and
% close the inner loop with the appropriate controller
% for the interval.
% Then convert from state space to transfer function form.

r=1;i=1;
while i <= k(1)
  while A(r,1)< int(i)
    numSYS=numSYS+1;
    [Ap,Bp,Cp,Dp]=feedback(A(r+1:r+2,:),Blong,C,D,..
                Akeq(i),Bkeq(i,:),Ckeq(i),Dkeq(i,:),1);
    [nump(numSYS,:),denp(numSYS,:)]=ss2tf(Ap,Bp,Cp(1,:),Dp(1,:));
    r=r+3;
  end;
  i=i+1;
end;

% Select the central controller and calculate the relative error
w=logspace(-2,2,n);
```

```
nump0=nump(k(37),:);
denp0=denp(k(37),:);

for z=1:numSYS
  numtemp=conv(nump(z,:),denp0)-conv(nump0,denp(z,:));
  dentemp=conv(denp(z,:),nump0);
  numtemp=numtemp(:,3:7);
  dentemp=dentemp(:,3:7);
  [a,b,c,d]=tf2ss(numtemp,dentemp);
  [sv]=sigma(a,b,c,d,w);
  j=j+max(sv);
end

j=j

return
%****************************************** end of file ****

%************************************* file relinter.m ****
function j=relinter(k)

% This function will take the given minimal order Hinf
% controller Keq and calculate the relative error to
% the chosen central controller

e=[];numSYS=0;j=[0];

n=45;
F = -40;  Kf = [1  1];  G = .0247;

%  Following are the flight conditions used in the
%  evaluation of relative error.

A=[47.4,0;[-.2296 .9931;.02436 -.2046];   %Am3h26
68.5,0;[-.2423 .9964;-2.342 -.1737]; %Am5h40
100.1,0;[-.4285 .9916;-.7473 -.3123]; %Am4h22
158.4,0;[-.5088 .994;-1.131 -.2804]; %Am6h30
189.9,0;[-.8018 .9847;-1.521 -.5944]; %Am4h6
255.0,0;[-.8930 .9852;-4.1582 -.6873]; %Am5h10
301.1,0;[-.9181 .9872;-6.2419 -.6920]; %Am6h15
355.0,0;[-.9920 .9888;-7.8450 -.7525]; %Am7h18_5
426.4,0;[-1.175 .9871;-8.458 -.8776]; %Am7h14
```

```
496.0,0;[-1.4710 .9808;-11.5022 -1.0846]; %Am6h2
557.0,0;[-2.1163 .9872;-32.6459 -1.1826]; %Am8h14
603.0,0;[-1.562 .9862;-14.94 -1.132]; %Am8h12;
614.4,0;[-1.905 .9895;-33.88 -.9872]; %Am95h20;
652.0,0;[-1.675 .9853;-16.16 -1.212]; %Am8h10;
705.0,0;[-2.1163 .9872;-32.6459 -1.1826]; %Am9h14
789.1,0;[-1.994 .9828;-19.44 -1.427]; %Am8h5;
825.2,0;[-2.452 .9856;-38.61 -1.34]; %Am9h10;
890.8,0;[-2.328 .9831;-30.44 -1.493] %Am85h5;
956.0,0;[-2.8375 .9855;-51.8325 -1.4037]; %Am95h9
998.7,0;[-2.911 .9835;-46.47 -1.553]; %Am9h5;
inf,0];

C=[1 0;0 1];
D=[0;0];
Blong=[0;1];

% Sort the interval break points from lowest to highest

int=[45,k(2:k(1)),1000];
s=length(int);

for l=1:(s-1)
  for i= 1:(s-1)
    if int(i)>int(i+1)
      temp = int(i);
      int(i) = int(i+1);
      int(i+1)=temp;
    end
  end
end

% Check the dynamic pressure of the flight condition to
% determine which interval endpoints to use.  Calculate
% the corresponding values of N and M.
% Form the controller Keq, close the inner loop and
% convert to transfer function form for the relative error
% calculation.

ni=10;m1i=20;m2i=30;
r=1;i=1;
while i <= k(1)
```

```
  while A(r,1)< int(i+1)
    numSYS=numSYS+1;
    N=k(ni)+(k(ni+1)-k(ni))*((A(r,1)-int(i))/(int(i+1)-int(i)));
    M1=k(m1i)+(k(m1i+1)-k(m1i))*((A(r,1)-int(i))/(int(i+1)-int(i)));
    M2=k(m2i)+(k(m2i+1)-k(m2i))*((A(r,1)-int(i))/(int(i+1)-int(i)));
    Akeq=F-G*N;
    Bkeq=Kf-G*[M1 M2];
    Ckeq=-N;
    Dkeq=-[M1 M2];
    [Ap,Bp,Cp,Dp]=feedback(A(r+1:r+2,:),Blong,C,D,...
Akeq,Bkeq,Ckeq,Dkeq,1);
    [nump(numSYS,:),denp(numSYS,:)]=ss2tf(Ap,Bp,Cp(1,:),Dp(1,:));
    r=r+3;
  end;
  i=i+1;ni=ni+1;m1i=m1i+1;m2i=m2i+1;
end;

% Select the central controller and calculate the relative error

w=logspace(-2,2,n);
nump=nump(:,3:4);
nump0=nump(k(40),:);
denp0=denp(k(40),:);

for z=1:numSYS
  numtemp=conv(nump(z,:),denp0)-conv(nump0,denp(z,:));
  dentemp=conv(denp(z,:),nump0);
  [a,b,c,d]=tf2ss(numtemp,dentemp);
  [sv]=sigma(a,b,c,d,w);
  j=j+max(sv);
end

% Close the outer loop and calculate the step response

longout  % a file containing the outer loop controller

for z=1:5
  [nums(z,:),dens(z,:)]=series(numoutlow,denoutlow,...
nump(z,:),denp(z,:));
  [numcl(z,:),dencl(z,:)]=cloop(nums(z,:),dens(z,:),-1);
end
for z=6:numSYS
```

```
  [nums(z,:),dens(z,:)]=series(numouthigh,denouthigh,...
nump(z,:),denp(z,:));
  [numcl(z,:),dencl(z,:)]=cloop(nums(z,:),dens(z,:),-1);
end

t=0:.1:5;


for z=1:numSYS
  [y,x]=step(numcl(z,:),dencl(z,:),t);
  Y=[Y,y];X=[X,x];
end

% Calculate the error of the step response

for z=1:numSYS
  e(:,z)=Y(:,z)-Y(:,k(40));
end
e=sum(abs(sum(e)));

j=j+e

return
%****************************************** end of file ****

%************************************** file longout.m ****
%  Outer loop longitudinal controller

Kouthigh=1.1995e2;
zouthigh=[-2.418e3,-5.4039+7.2572i,-5.4039-7.2572i]';
pouthigh=[-7.1836e1+4.0476e1i,-7.1836e1-4.0476e1i,-7.2319e-2,-2.0509e1]';

Koutlow=1.2951e2;
zoutlow=[-6.0168e2,-3.8407+6.2507i,-3.8407-6.2507i]';
poutlow=[-5.9135e1+5.1963e1i,-5.9135e1-5.1963e1i,-3.6321e-2,-6.6426]';

[numouthigh,denouthigh]=zp2tf(zouthigh,pouthigh,Kouthigh);
[numoutlow,denoutlow]=zp2tf(zoutlow,poutlow,Koutlow);
[Akoutlow,Bkoutlow,Ckoutlow,Dkoutlow]=zp2ss(zoutlow,poutlow,Koutlow);
[Akouthigh,Bkouthigh,Ckouthigh,Dkouthigh]=zp2ss(zouthigh,pouthigh,Kouthigh);

%****************************************** end of file ****
```

Bibliography

1. *Proceedings of the 1992 IEEE Conference on Decision and Control*, Dec 1992.
2. *Proc. IEEE National Aerospace and Electronics Conference*, May 1994.
3. *Proceedings of the First IEEE Conference on Evolutionary Computation*, June 1994.
4. Flight Dynamics Branch, Flight Control Division. *An Introduction to Multivariable Flight Control System Design.* Technical Report WL-TR-92-3110. 1992.
5. Apkarian, Pierre and Pascal Gahinet. "A Convex Characterization of Gain Scheduled H_∞ Controllers." Submitted to IEEE Transactions on Automatic Control.
6. Apkarian, Pierre, et al. "Self Scheduled H_∞ Control of Linear Parameter-Varying Systems: A Design Example." Submitted to Automatica.
7. Bauer. *Genetic Algorithms and Investment Strategies.* New York: John Wiley and Sons, 1994.
8. Blakelock, John. *Automatic Control of Aircraft and Missiles, 2nd Ed..* John Wiley and Sons, Inc., 1991.
9. Bramlette and Cusin. "A Comparative Evaluation of Search Methods Applied to Parametric Design," *Proceedings of the Third International Conference on Genetic Algorithms*, 213–218 (1989).
10. Davis, L., editor. *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991.
11. Dejong, K. A. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD dissertation, University of Michigan, 1975.
12. Dejong, K. A. "Adaptive System Design: a Genetic Approach," *IEEE Transactions on Systems, Man, and Cybernetics, 10*(9):566–574 (Sept 1980).
13. Department of Defense. *Military Standard – Flying Qualities of Piloted Aircraft.* Technical Report MIL-STD-1797A. 30 Jan 1990.
14. Doyle, J.C. "Structured Uncertainty in Control System Design," *Proceedings of the 24th IEEE Conference on Decision and Control* (December 1985).
15. Doyle, J.C., et al. *Feedback Control Theory.* New York: Macmillian Publishing Company, 1992.
16. Dymec, Andrew. *Design and Implementation of Parallel Genetic Algorithms for Solving Large Scale Optimization Problems.* MS thesis, Air Force Institute of Technology, Wright-Patterson AFB OH, March 1992.
17. Filho, José L. Ribeiro, et al. "Genetic Algorithms: A Survey," In *Computer* [49].
18. Gahinet, Pascal, et al. "Parameter-Dependent Lyapunov Functions for Real Parametric Uncertainty." Submitted to IEEE Transactions on Automatic Control.

19. Gelb, Arthur, editor. *Applied Optimal Estimation.* Cambridge, Massachusettes: M.I.T. Press, 1992.

20. Goldberg, David. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, 1989.

21. Goldberg, David. "Messy Genetic Algorithms: Motivations, Analysis And First Results," *Complex Systems*, *3*:493–530 (1989).

22. Goldberg, David. "Messy Genetic algorithms revisited: Studies in mixed size and scale," *Complex Systems*, *4*:415–444 (1990).

23. Grefenstette, John. "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-16*(1):122–128 (Jan/Feb 1986).

24. Guo, Di and Wilson J. Rugh. "An Approach to Gain Scheduling on Fast Variables." In *Proceedings of the 1992 IEEE Conference on Decision and Control* [1].

25. Holland, John. *Adaptation in Natural and Artificial Systems.* PhD dissertation, University of Michigan, 1975.

26. Kirkpatrick, S., et al. "Optimization by Simulated Annealing," *Science*, *220*(4598):671–681 (May 1983).

27. Krishnakumar, K. and D. Goldberg. "Control System Optimization Using Genetic Algorithms," *Journal of Guidance, Control, and Dynamics*, *15*(3):735–739 (May-June 1992).

28. Lawrence, Douglas. "On A Nonlinear Observer With Pseudo-Linearized Error Dynamics." In *Proceedings of the 1992 IEEE Conference on Decision and Control* [1].

29. Lawrence, Douglas A. and Wilson J. Rugh. "On a Stability Theorem for Nonlinear Systems with Slowly-Varying Inputs," *IEEE Trans. on Automatic Control*, *AC-35*(7):860–864 (1990).

30. Lawrence, Douglas A. and Wilson J. Rugh. "Gain Scheduling Dynamic Linear Controllers for a Nonlinear Plant." *Proceedings of the 32nd IEEE Conference on Decision and Control.* 1024–1029. 1993.

31. Mathworks. *Matlab©User's Manual.* Mathworks, 1993.

32. McGregor, D.R., et al. "Adaptive Control of a Dynamic System Using Genetic-Based Methods," *Proceedings of the 1992 IEEE Symposium on Intelligent Control*

33. Merkle, Laurence D. *Parallelization and Analysis of Standard and Messy Genetic Algorithms.* MS thesis, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1992.

34. Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1992.

35. Michalewicz, Zbigniew, et al. "A Modified Genetic Algorithm of Optimal Control Problems," *Computers, Mathematics, and Applications*, *23*(12):83–94 (1992).

36. Ortega, R. "New techniques to improve perfomance of simple control loops," *Low Cost Automation 1992, Techniques, Components and Instruments, and Applications. Selected papers from the 3rd IFAC Symposium*, 1–6 (Sept 1992).

37. Packard, Andy. "Gain Scheduling via Linear Fractional Transformations," *Systems and Control Letters*, *22*:79–92 (Feb 1994).

38. Porter, B. and D.L. Hicks. "Genetic Robustification of Digital Model-Following Flight Control Systems." In *Proc. IEEE National Aerospace and Electronics Conference* [2].

39. Porter, B. and D.L. Hicks. "Perfomance Measures in the Genetic design of Digital Model-Following Flight Control Systems." In *Proc. IEEE National Aerospace and Electronics Conference* [2].

40. Reorda, Matteo Sonza and M. Rebaudengo. "A Genetic Algorithm for Floor Plan Area Optimization." In *Proceedings of the First IEEE Conference on Evolutionary Computation* [3].

41. Rugh, Wilson. "Analytical Framework for Gain Scheduling," *IEEE Control Systems*, 79–84 (1991).

42. Safonov, M.G. and R.Y. Chiang. "Model Reduction for Robust Control: A Schur Relative Error Method," *International Journal of Adaptive Control and Signal Processing*, *2*:259–272 (1988).

43. Shamma, Jeff. *Analysis and Design of Gain Scheduled Control Systems*. Ph.D. dissertation, Massachusetts Institute of Technology, 1988.

44. Shamma, Jeff and Mike Athans. "Analysis of Gain Scheduled Control for Nonlinear Plants," *IEEE Trans. on Automatic Control*, *35*(8):898–907 (August 1990).

45. Shamma, Jeff and Mike Athans. "Guaranteed Properties of Gain Scheduled Control for Linear Parameter-varying Plants," *Automatica*, *27*(3):559–564 (1991).

46. Shamma, Jeff and Mike Athans. "Gain Scheduling: Potential Hazards and Possible Remedies," *IEEE Control Systems Magazine*, *12*(3):101–107 (June 1992).

47. Shamma, Jeff and James Cloutier. "Gain Scheduled Missile Auto Pilot Design Using Linear Parameter Varying Transformations," *Journal of Guidance, Control, and Dynamics*, *16*(2):256–263 (Mar-Apr 1993).

48. Srinivas, M. and L. M. Patnaik. "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, *24*(4):656–667 (Apr 1994).

49. Srinivas, M. and Lalit M. Patnaik. "Genetic Algorithms: A Survey," *Computer*, *27*(6) (June 1994).

50. Vanderplaats, Garret. *Numerical Optimization Techniques for Engineering Design.* McGraw-Hill, Inc., 1984.

51. White, et al. "Missile autopilot design using gain scheduling techniques," *Proceedings of the 26th Southeastern Symposium on System Theory*, 606–610 (Mar 1994).

52. Yeh, H.H., et al. "Robust Control Design with Real Parameter Uncertainties." *Proc. 1992 American Control Conference.* June 1992.

Vita

Robert C. Martin IV [redacted]. He recieved a B.S. in aeronautical engineering from the University of Cincinnati in 1993. After recieving an officer commission in the US Air Force, he attended the Air Force Institute of Technology where he will complete a M.S degree in aeronuatical engineering in Dec. 1994.

Permanent address: [redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Master's Thesis

4. TITLE AND SUBTITLE
A GAIN SCHEDULING OPTIMIZATION METHOD USING GENETIC ALGORITHMS

5. FUNDING NUMBERS

6. AUTHOR(S)
Robert C. Martin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GAE/ENY/94D-3

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Gain scheduling, the traditional method of providing adaptive control to a nonlinear system, has long been an ad hoc design process. Until recently, little theoretical guidance directed this practioners' art. For this reason, a systematic study of this design process and its potential for optimization has never been accomplished. Additionally, the nonlinearities and the large search space involved in gain scheduling also precluded such an optimization study. Traditionally, the gain scheduling process has been some variation of a linear interpolation between discrete design points. By using powerful non-traditional optimization tools such as genetic algorithms there are ways of improving this design process.

This thesis utilizes the power of genetic algorithms to optimally design a gain schedule. First, a design methodology is validated on a simple pole placement problem, then demonstrated for an F-18 Super-maneuverable Fighter. From this experience, a general gain scheduling design process is developed and presented.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Scheduling, Flight Control Systems, Algorithms	152
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL