AFIT/GAE/ENY/91D-7

AD-A244 267

DTIC
ELECTE
JAN 08 1992
S B D

AN ALGORITHM FOR ROBUST EIGENSTRUCTURE ASSIGNMENT USING THE LINEAR QUADRATIC REGULATOR

THESIS

Thomas C. Huckabone, Capt, USAF

AFIT/GAE/ENY/91D-7

Approved for public release; distribution unlimited

92-00113

AFIT/GAE/ENY/91D-7

AN ALGORITHM FOR ROBUST EIGENSTRUCTURE ASSIGNMENT USING THE LINEAR QUADRATIC REGULATOR

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

by

Thomas C. Huckabone, B.S.M.E.

Captain, USAF

December 1991

Approved for public release; distribution unlimited

Acknowledgements

I would like to express my deepest gratitude to my wife Karen for her support throughout this thesis effort. Her editing help and her patience throughout the birth of our daughter Kara were vital to me. I would also like to thank Dr. Brad Liebst for his advice and indispensable help during this research effort.

DTIC COPY INSPECTED

Accession For

A-1

TABLE OF CONTENTS

	Page
List of Figures	v
List of Tables	vi
List of Symbols	vii
Abstract	xi
I. Introduction	1
Background	2
Problem Statement	4
Methodology	6
Organization	7
II. Theory	9
Eigenstructure Assignment	10
Linear Quadratic Regulator	14
Stability Robustness	16
Linear Quadratic Regulator Stability Margins	17
III. Robust Eigenstructure Assignment Algorithm	23
Algorithm Equations	23
Programing Considerations	28
Existing Subroutines	28
Newly Developed Suboutines	28
Program Flow	29
Using the Program	35
IV. Examples	37
Example 1 - F-4 Lateral Inner Loop	37
Comparison to Robinson's Results	39
Comparison to Harvey and Stein Results	41
Example 2 - YAH-64 Helicopter Control System	52
Desired Eigenstructure	54
Garrard and Liebst Results	56
Robust Eigenstructure Assignment Algorithm Results	57
Comparison to Garrard and Liebst Results	65
V. Conclusions and Recommendations	69
Appendix A - Subroutine Descriptions	72

Page

Appendix B - Subroutine Source Codes 78

Bibliography 104

Vita 106

List of Figures

Figure		Page
1	Block Diagram of the Linear System	10
2	Simplified Block Diagram for Closed Loop System	18
3	Computer Program Hierarchy	30
4	F-4 Example Minimum Singular Values for Variations of **R** Matrix	49
5	Helicopter Example Minimum Singular Values for Unity Weighting Case with **R**>0	60
6	Helicopter Example Minimum Singular Values for the Three Weighting Cases	66
7	Comparison of Minimum Singular Values of Return Difference Matrix for Garrard and Liebst Design and for Robust Eigenstructure Assignment Algorithm Design	67

List of Tables

Table		Page
1	Constants Added to XGUESS to Build Initial Simplex	32
2	Comparison to Robinson Pole Placement Results, F-4 Example, $\mathbf{R}=\rho I$	40
3	Desired Eigenvectors for F-4 Lateral Inner Loop	42
4	Results from Robust Eigenstructure Assignment Algorithm for F-4 Example, $\mathbf{R}=\rho I$, Open Loop Actuator Poles of -10 and -5	43
5	Results from Harvey and Stein Method for F-4 Example	44
6	Results from Robust Eigenstructure Assignment Algorithm for F-4 Example ($\mathbf{R}=\rho I$), Open Loop Actuator Poles of -20 and -10	46
7	Results from Robust Eigenstructure Assignment Algorithm for F-4 Example with Heavier Weighting on Dutch Roll Poles ($\mathbf{R}=\rho I$)	47
8	Results from Robust Eigenstructure Assignment Algorithm for F-4 Example with $\mathbf{R}$=[diagonal]	48
9	Results from Robust Eigenstructure Assignment Algorithm for F-4 Example with $\mathbf{R}>0$	50
10	Open Loop Eigenstructure for Helicopter Example	54
11	Desired Eigenstructure for Helicopter Example	55
12	Garrard and Liebst Helicopter Example Results for Full State Feedback	57
13	Robust Eigenstructure Assignment Results for Helicopter Example with Unity Weighting and $\mathbf{R}>0$	58
14	Helicopter Example Results with $\mathbf{R}>0$, First Weighting Variation	62
15	Helicopter Example Results with $\mathbf{R}>0$, Second Weighting Variation	64
16	Independent Stability Margins for Three Weighting Cases, Helicopter Example	65

List of Symbols

α	minimum singular value of return difference matrix
α_r	constant for calculation of reflected vector
β	side slip
β_c	constant for calculation of contracted vector
γ_e	constant for calculation of expanded vector
δ_a	aileron deflection
δ_{ac}	commanded aileron deflection
δ_r	rudder deflection
δ_{rc}	commanded rudder deflection
θ	pitch angle
Θ	diagonal matrix of eigenvector diff. min. parameter
θ_i	eigenvector difference minimization parameter
λ	eigenvalue (pole) of **A** matrix
$\bar{J}_\lambda$	algorithm performance index - pole contribution
λ_a	achievable pole
Λ_a	diagonal matrix of achievable poles
λ_d	desired pole
Λ_d	diagonal matrix of desired poles
ρ	constant multiplied by **I** to form **R**
ϕ	roll angle
ϕ_i	angle for eigenvector difference minimization parameter
$\bar{\sigma}$	maximum singular value
$\underline{\sigma}$	minimum singular value

a	constant used in building initial simplex
$\mathbf{A}$	state matrix
$\mathbf{A}_{nd}$	non-dimensionalized state matrix
$\mathbf{B}$	control matrix
$\mathbf{B}_{nd}$	non-dimensionalized control matrix
$\mathbf{C}$	output matrix
c_i	constants used in building initial simplex
E_a	plant additive uncertainty matrix
E_i	i^{th} eigenvector difference error
E_m	plant multiplicative uncertainty matrix
EJ	minimization convergence criteria
$f_{\lambda i}$	algorithm weight for i^{th} pole
$\mathbf{Fe}$	algorithm pole weighting matrix input to program
$\mathbf{Fv}$	algorithm eigenvector weighting matrix input to program
$\mathbf{Fv}_i$	algorithm weighting matrix for i^{th} eigenvector
$\mathbf{G}$	nominal plant matrix
$\mathbf{G}_p$	perturbed plant matrix
$\mathbf{H}$	symmetric matrix used to build $\mathbf{Q}$
$\mathbf{I}$	identity matrix
j	square root of negative one
J	LQR performance index
$\bar{J}$	algorithm performance index
$j\omega$	frequency domain parameter
J_{es}	performance index used by Garrard and Liebst
k	SISO feedback gain

$\mathbf{K}$	feedback gain matrix
$\mathbf{K}_{nd}$	non-dimensionalized feedback gain matrix
$\mathbf{M}$	symmetric matrix used to build $\mathbf{R}$
nx	size of minimization vector
p	roll rate
$\mathbf{P}$	Riccati equation solution matrix
$\mathbf{P}_i$	Garrard and Liebst eigenvector weighting matrix
p_s	stability axis roll rate
q	pitch rate
$\mathbf{Q}$	LQR state weighting matrix
r	yaw rate
$\mathbf{R}$	LQR control weighting matrix
rcode	code specifying type of R matrix to be used
r_s	stability axis yaw rate
T_1	state non-dimensionalization transformation matrix
T_2	control non-dimensionalization transformation matrix
TF_{CL}	closed loop SISO transfer function
tol	convergence tolerance for algorithm
$\mathbf{u}$	control vector
u	forward velocity
$\mathbf{u}_{nd}$	non-dimensionalized control vector
v	lateral velocity (ft/sec)
$\mathbf{V}$	modal matrix
v_a	achievable eigenvector
$\mathbf{V}_a$	achievable modal matrix
v_{aI}	imaginary part of achievable eigenvector

v_{aR}	real part of achievable eigenvector
v_d	desired eigenvector
$\mathbf{V_d}$	desired modal matrix
v_{dI}	imaginary part of desired eigenvector
v_{dR}	real part of desired eigenvector
$\bar{J}_{vec}$	algorithm performance index - eigenvector contribution
v_i	i^{th} eigenvector
w	downward velocity (ft/sec)
$\mathbf{W}$	matrix of w_i's
w_i	i^{th} eigenvector multiplied by feedback gain matrix
$\mathbf{x}$	state vector
X	vector used to minimize performance index
$\bar{X}$	average of simplex vectors
$X_{contracted}$	contracted vector
$X_{expansion}$	expansion vector
XGUESS	initial guess vector for minimization routine
X_j	j^{th} column of simplex
$\mathbf{x_{nd}}$	non-dimensionalized state vector
$X_{reflected}$	reflected vector
X_{worst}	vector producing worst cost function value
$\mathbf{y}$	output vector

AFIT/GAE/ENY/91D-7

Abstract

The Linear Quadratic Regulator (LQR) can guarantee a robust closed loop eigenstructure for full state feedback. The algorithm developed here takes advantage of the stability guarantees of LQR to achieve an eigenstructure close to desired but within the allowable region of LQR. The algorithm selects the LQR weighting matrices, **Q** and **R**, that minimize the distance between the elements of the desired and LQR achievable eigenstructures. The minimization is accomplished by using a simplex based optimization routine. Specific weightings placed on the elements of the desired eigenstructure define the relative importance of each element. The algorithm is programed in FORTRAN and is designed to be run from the software package MATLAB. Two examples are examined to illustrate the use of the program, including a helicopter flight control system. The results show that this algorithm is a valid technique for achieving robust eigenstructrue assignment with full state feedback.

AN ALGORITHM FOR ROBUST EIGENSTRUCTURE ASSIGNMENT USING THE LINEAR QUADRATIC REGULATOR

I. Introduction

Many current aerospace systems require very complex control systems to provide the desired performance and system stability simultaneously. The dynamics of these system are approximated by mathematical models which are then used to develop feedback control laws. Most of these system models require multiple inputs and have multiple outputs (MIMO). These MIMO models can be broken down into a series of single input/single output (SISO) subsystems, but this can make determining the cross coupling effects between the various SISO subsystems extremely difficult. Classical control system design techniques, developed prior to the availability of today's computer capability, often require multiple input systems to be broken down in this way. These techniques do not lend themselves to automation through relatively simple computer programming because many subjective decisions are required throughout the design process to achieve the best mix between desired performance and system stability. The requirement for numerous subjective decisions makes the control system design for

complex systems excessively cumbersome and time consuming when using classical design techniques.

Eigenstructure assignment and the Linear Quadratic Regulator (LQR) are two control system design techniques that have been developed to produce stable MIMO systems with good performance characteristics. Both techniques can be used to develop feedback controls for MIMO systems without breaking the systems down into SISO subsystems. Eigenstructure assignment provides the advantage of allowing great flexibility in shaping the closed loop system response by allowing specification of closed loop poles and eigenvectors, but has the disadvantage that stability robustness is not guaranteed. The LQR technique assures stability robustness with full state feedback but does not provide the flexibility of eigenstructure assignment in placing closed loop poles and eigenvectors. This thesis develops a method that has the flexibility of eigenstructure assignment within the stability constraints of LQR.

Background

Eigenstructure assignment is a technique that allows a control system designer to specify the desired closed loop performance characteristics of a MIMO system. These performance characteristics are specified through desired eigenvalues and eigenvectors, i.e. the desired closed loop eigenstructure. Moore [1] showed how to exploit the fact that specifying the closed loop eigenvalues of a MIMO system

does not result in a unique solution. Moore further demonstrated that a specific number of elements of each eigenvector of a closed loop MIMO system can be freely assigned. Other papers, including Sobel, Yu, and Lallman [2] and Garrard and Liebst [3,4], have developed algorithms to achieve the desired eigenstructures for closed loop systems. A helicopter flight control system example used by Garrard and Liebst [3] is used as one example in Chapter IV of this report. The validity of eigenstructure assignment as a flight control system design tool has been experimentally investigated by Calico [5].

LQR is an optimal control design method that results in a system with guaranteed robustness. Anderson and Moore [6] have shown that use of the LQR design method results in gain and phase margins of at least $(-6,\infty)$ db and $(-60,60)$ degrees respectively. The state and control weighting matrices for the LQR cost function must be selected by the designer in an attempt to achieve the desired system performance. The desired system performance characteristics (or the desired eigenstructure) often do not lie within the achievable solution space of LQR. A paper by Innocenti and Stanziola [7] compares the robustness achievable through eigenstructure assignment to that guaranteed by the LQR method.

Other researchers have developed eigenstructure assignment methods that use LQR. Wilson and Cloutier [8]

developed a technique that minimized a cost function that provides a tradeoff between desired eigenvalues and eigenvectors, allowing some flexibility in the number of elements assigned in the eigenstructure. The algorithm developed by Broussard [9] minimizes a cost function involving the feedback gain matrix; however, this method requires that the gain matrix associated with the desired closed loop eigenstructure be calculated first. Harvey and Stein [10] developed a method that uses the asymptotic properties of LQR to place eigenvalues and uses a linear projection to determine the achievable eigenvectors. The F-4 aircraft example used by Harvey and Stein is the first example used in Chapter IV of this thesis.

This thesis shows the development of an algorithm that allows a control system designer to achieve closed loop eigenstructure close to desired within the constraints of the LQR stability margins. This report is an extension of the work of Robinson [11]. Robinson developed an algorithm using the software package MATLAB to provide eigenvalue placement using the LQR. This thesis enhances Robinson's work by adding eigenvector assignment as well.

Problem Statement

Consider the state space representation of the multivariate, linear, time-invariate feedback system,

$$\dot{x}=Ax+Bu \tag{1}$$

and

$$y=Cx \tag{2}$$

where

x = n dimensional state vector

u = m dimensional control vector

y = output vector

A, B, C = constant matrices of appropriate dimensions

If linear feedback of all of the state variables is provided in the form,

$$u=-Kx \tag{3}$$

and (**A**,**B**) is controllable, a feedback gain matrix, **K**, can be found that shifts the closed loop eigenvalues to any desired location. LQR can be used with a system of this form. Anderson and Moore [6] have shown that LQR can be used to assure closed loop stability robustness by minimizing the quadratic cost function

$$J=\int_0^\infty (x^TQx+u^TRu)\,dt \tag{4}$$

where,

Q = designer specified state weighting matrix

R = designer specified control weighting matrix

Letting

$$K=R^{-1}B^TP \tag{5}$$

provides the optimal solution where **P** is a positive definite solution to the algebraic Riccati equation,

$$PA+A^TP+Q-PBR^{-1}B^TP=0 \tag{6}$$

For any positive semidefinite, symmetric **Q** and positive definite **R** selected, a feedback gain matrix **K** can be determined that will result in an eigenstructure that provides stability robustness for the closed loop system. More simply stated, the resulting eigenstructure will always fall within an allowable LQR region. It is important to note that the eigenstructure desired by the designer may not fall within the allowable LQR region.

The problem is then to place the achievable eigenstructure close to desired while maintaining the stability robustness characteristics of the LQR.

Methodology

The solution to this problem is accomplished by introducing a second quadratic cost function,

$$\bar{J}=\sum_{i=1}^{n}\left[f_{\lambda_i}(\lambda_{d_i}-\lambda_{a_i})^2+(v_{d_i}-\theta_i v_{a_i})^* F_{v_i}(v_{d_i}-\theta_i v_{a_i})\right] \tag{7}$$

where

n = number of states

$f_{\lambda i}$ = weighting on the ith eigenvalue

$\lambda_{d\,i}$ = i^{th} desired eigenvalue

$\lambda_{a\,i}$ = i^{th} achievable eigenvalue

$v_{d\,i}$ = i^{th} desired eigenvector

$v_{a\,i}$ = i^{th} achievable eigenvector

F_{vi} = diagonal weighting matrix for the ith eigenvector

θ_i = real or complex constant that minimizes $(v_{di}-\theta_i v_{di})$

Equation (7) is minimized over the elements of Q and R, subject to equations (4)-(6). As $\bar{J}$ becomes small, the system comes close to providing the desired closed loop eigenstructure. This method also allows for individual weightings to be placed on each element of the eigenstructure. The weightings can be used to allow an element to move freely (by setting the weighting to zero) or to place greater emphasis on an individual element (by setting the weighting to a larger value than other weightings).

The algorithm is programmed in FORTRAN and interfaces with the software package MATLAB. Input parameters are specified in MATLAB and then the compiled FORTRAN program is called and run from inside MATLAB.

Organization

This report is organized as follows:

- Chapter II contains the theory involved in the development of the robust eigenstructure assignment algorithm. Discussions on eigenstructure assignment and the LQR method are followed by definitions of stability robustness and a discussion on LQR stability margins.

- Chapter III develops the robust eigenstructure assignment algorithm and provides a discussion on the implementation of the algorithm. Included is a discussion on the minimization routine used.
- Chapter IV provides examples of the use of the algorithm. Two examples are shown, including the examples used by Harvey and Stein [10] and Garrard and Liebst [3] with their eigenstructure assignment methods.
- Chapter V gives conclusions resulting from this study and recommendations for further work in the area of eigenstructure assignment using the LQR.
- Appendix A contains detailed information on the FORTRAN subroutines developed by the author that implement this algorithm.
- Appendix B provides a listing of the source code for the main FORTRAN program and the subroutines developed by the author. Also included is a listing of the MATLAB m-file that interfaces with the FORTRAN program.

II. Theory

This chapter introduces the theory used in the development of the robust eigenstructure assignment algorithm. Included are discussions on eigenstructure assignment and the LQR method. The chapter is concluded with discussions on stability robustness and LQR stability margins.

All of the discussions in this chapter center around the state space representation of a multivariable, linear, time-invariate feedback system of the form already given in equations (1)-(3). These equations are repeated here as equation (8)-(10) for convenience.

$$\dot{x}=Ax+Bu \tag{8}$$

$$y=Cx \tag{9}$$

where,

x = n dimensional state vector

u = m dimensional control vector

y = output vector

A, B, C = constant matrices of appropriate dimensions

Linear feedback of the state variables is assumed in the form,

$$u=-Kx \tag{10}$$

Eigenstructure Assignment [1,12]

Again consider the system of equations (8)-(10) with full state feedback. Assume the matrix **B** is of full column rank and the system is both controllable and observable. This system can be represented by the block diagram of Figure 1.

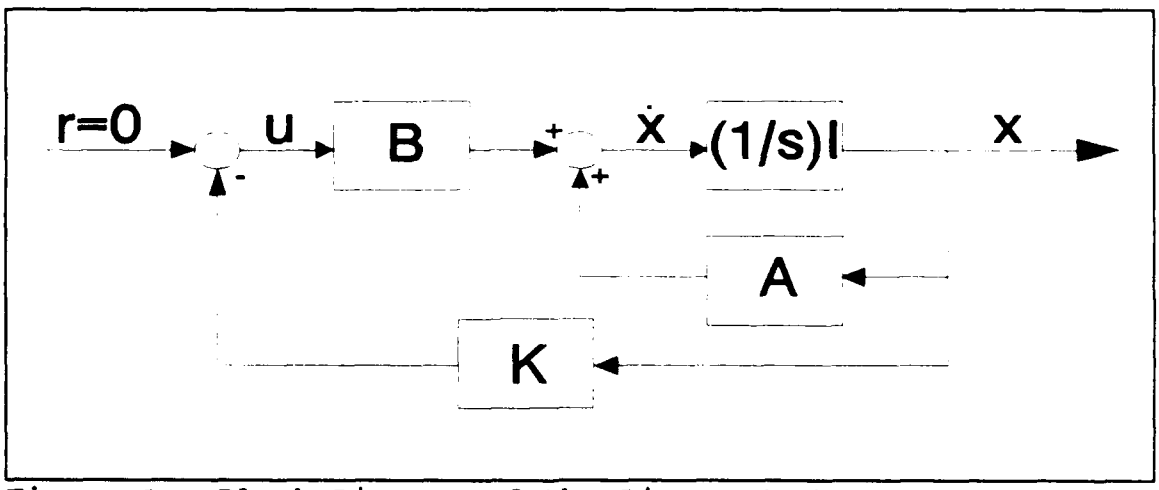

Figure 1 - Block Diagram of the Linear System

The state space representation of the closed loop system becomes

$$\dot{x}=(A-BK)x \tag{11}$$

The closed loop eigenvalues can be determined from the roots of the characteristic equation

$$|\lambda_i I-(A-BK)|=0 \tag{12}$$

Examination of equation (12) shows that, if **B** is of full column rank, each closed loop eigenvalue can be influenced by changing the feedback gain matrix **K**, regardless of the number of inputs. If a designer has complete flexibility to

assign any values to the elements of **K**, the closed loop poles can be placed anywhere in the complex plane.

The closed loop right eigenvectors can be determined from the set of equations given by equation (13) where $\mathbf{v}_i$ is the ith closed loop right eigenvector

$$[\lambda_i I-(A-BK)]v_i=0 \tag{13}$$

Equation (13) can be multiplied by any arbitrary constant θ without influencing the closed loop system. This arbitrary constant can be real or complex. Equation (13) can be rewritten to provide insight into the ability to assign values to elements of the eigenvectors.

$$(\lambda_i I-A)v_i+BKv_i=0 \tag{14}$$

$$v_i=-(\lambda_i I-A)^{-1}B[Kv_i] \tag{15}$$

In equation (15), the matrix $(\lambda_i \mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$ is nxm dimensional. If **B** is of full column rank, the ability to assign values to elements of the eigenvectors is then limited by the subspace spanned by $(\lambda_i \mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$ and only m elements of each eigenvector can be assigned. It is not possible to assign values to an eigenvector that will cause it to lie outside of this subspace. The ability to assign specific values to elements of the eigenvectors is limited in part by the column dimension of the **B** matrix, i.e. the number of inputs to the system. In addition, when one element of the eigenvector associated with a complex pole has been

specified, the corresponding element of the eigenvector associated with the complex conjugate of that pole has also been specified.

The eigenstructure assignment design technique is to specify the eigenvalues and associated eigenvectors and to let

$$w_i = K v_i \tag{16}$$

Equation (15) may now be rewritten

$$(\lambda_i I - A)\, v_i = B w_i \tag{17}$$

As already shown, all of the eigenvalues can be placed exactly, so as long as the desired eigenvectors lie within the achievable subspace the only unknowns in equation (17) are the elements of each $\mathbf{w}_i$. Equation (17) can be solved for the elements of the $\mathbf{w}_i$'s. Once the elements of each $\mathbf{w}_i$ have been calculated, the gain matrix K can be determined by combining the set of n equations from equation (16) into a single matrix equation. Define the matrices **W** and **V** as

$$W = [w_1 \;\; w_2 \ldots w_n] \tag{18}$$

$$V = [v_1 \;\; v_2 \ldots v_n] \tag{19}$$

The matrix **V** containing the right eigenvectors is often refered to as the modal matrix. Combining equations (16) yields

$$W = KV \tag{20}$$

Since the eigenvectors are linearly independent, the **V** matrix is nonsingular and equation (20) becomes

$$K = WV^{-1} \tag{21}$$

In practice, the desired eigenvectors are often not achievable, not lying within the subspace spanned by $(\lambda_i \mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$. This means that a solution for **K** that will yield a closed loop system that has the desired eigenvectors is not possible. One method to get around this problem is to project the desired eigenvectors onto the achievable subspace, minimizing the difference between the desired and achievable vectors. Other researchers have developed methods to accomplish this. Liebst, Garrard, and Adams [12] achieve this by introducing a quadratic cost function J_{es_i} to be minimized subject to equation (17), where

$$J_{es_i} = (v_{ai} - v_{di}) \cdot P_i (v_{ai} - v_{di}) \tag{22}$$

and where P_i = diagonal weighting matrix for the ith eigenvector. Equations (17)-(22) are then manipulated to solve for **W** and **K** matrices that come close to providing the desired eigenstructure. The eigenstructure assignment method, then, provides a means to specifically place eigenvalues and optimally place eigenvectors for a control system.

The algorithm developed in Chapter III of this report will introduce a cost function that includes that of equation (22), but will ensure robust stability by introducing the LQR as a constraint.

<u>Linear Quadratic Regulator</u> [13:section 6.1]

Refer again to the full state feedback system with state equation

$$\dot{x}=Ax+Bu \tag{23}$$

and linear feedback of the state variables defined by

$$u=-Kx \tag{24}$$

The **B** matrix is assumed to be of full column rank. As already discussed, there is a great deal of flexibility in designating the closed loop eigenstructure, but the stability robustness of the closed loop system cannot be assured. LQR provides a means of guaranteeing closed loop stability robustness. Stability robustness is guaranteed by introducing the LQR quadratic performance index

$$J=\int_0^{\infty}(x^TQx+u^TRu)\,dt \tag{25}$$

where **Q** and **R** are symmetric, non-negative weighting matrices designated by the control system designer to place relative importance on the states and controls. Minimizing the LQR performance index, J, will ensure that the deviations of the

states from nominal will be kept small without using excessive control actions.

To ensure that the LQR performance index is finite, and therefore has a minimum, it is required that all unstable states can be made stable and that these states are reflected in the LQR performance index. Requiring that the pair [**A**,**B**] be stabilizable will guarantee that all unstable states can be made stable. Requiring **Q** to be positive semi-definite by letting **H** be any nxn symmetric matrix such that

$$Q=H^TH \tag{26}$$

and requiring that the pair [**A**,**H**] be detectable ensures that all open loop unstable modes will be seen in the LQR performance index. Meeting both of these conditions guarantees the closed loop system will always be stable.

The LQR performance index is minimized when

$$K=R^{-1}B^TP \tag{27}$$

where **P** is the unique positive semidefinite solution to the algebraic Riccati equation given by

$$PA+A^TP+Q-PBR^{-1}B^TP=0 \tag{28}$$

To use equations (25), (27) and (28), the **R** matrix must be invertible and non-negative, and therefore **R** must be positive definite (**R**>0). Defining a symmetric matrix **M** such that

$$R=M^TM$$

will ensure that **R** is positive definite. It can also be shown that the minimum value of the LQR performance index is given by

$$J_{min}=x^T(0)Px(0) \tag{30}$$

Kwakernaak and Sivan [14] provide further discussion on the algebraic Riccati equation and the gain matrix for optimization of the LQR performance index.

Use of the LQR optimal solution for the feedback gain forces constraints on the closed loop eigenstructure of the system. The closed loop eigenstructure is forced into a particular "region" subject to equations (27) and (28). The achievable LQR region is dependent on the **A** and **B** matrices of a given system. A change to the elements of **A** or **B** will result in a different achievable LQR region. For MIMO systems a closed form solution for the achievable LQR region is normally not possible. Thus, when selecting a desired eigenstructure, a designer cannot be assured that a particular selection will lie entirely within the achievable LQR region.

Stability Robustness [13:section 3.2]

A closed loop system is said to possess stability robustness if it remains stable when uncertainties are

present. These uncertainties can be modeled in many ways, including as additive or multiplicative in nature.

Letting **G(s)** represent a nominal open loop plant and $\mathbf{G_p(s)}$ represent the perturbed plant, additive perturbations can be modeled as

$$E_a(s) = G_p(s) - G(s) \tag{31}$$

and multiplicative perturbations can be modeled as

$$E_m(s) = G^{-1}(s)\,[G_p(s) - G(s)] \tag{32}$$

The ability of a system to remain stable in the face of these uncertainties can be estimated by using two robustness tests

$$\bar{\sigma}[E_m(s)] < \underline{\sigma}[I + G(s)] \tag{33}$$

and

$$\bar{\sigma}[E_a(s)] < \underline{\sigma}[I + G^{-1}(s)] \tag{34}$$

where $\bar{\sigma}[\,]$ represents the maximum singular value of the enclosed matrix and $\underline{\sigma}[\,]$ represents the minimum singular value of the enclosed matrix.

Linear Quadratic Regulator Stability Margins [13:chapter 7]

The development thus far has demonstrated that LQR will ensure a stable system. The stability robustness characteristics of the LQR closed loop system will now be addressed. To investigate the stability robustness

characteristics of LQR, refer once more to the full state feedback system with state equation

$$\dot{x}=Ax+Bu \tag{35}$$

and linear feedback of the state variables defined by

$$u=-Kx \tag{36}$$

Figure 2 shows a simplified block diagram representation of this full state feedback system. Before continuing with the discussion of LQR stability, the concepts of the return difference, independent gain and phase margins, and the relationship of the Kalman Inequality must be introduced.

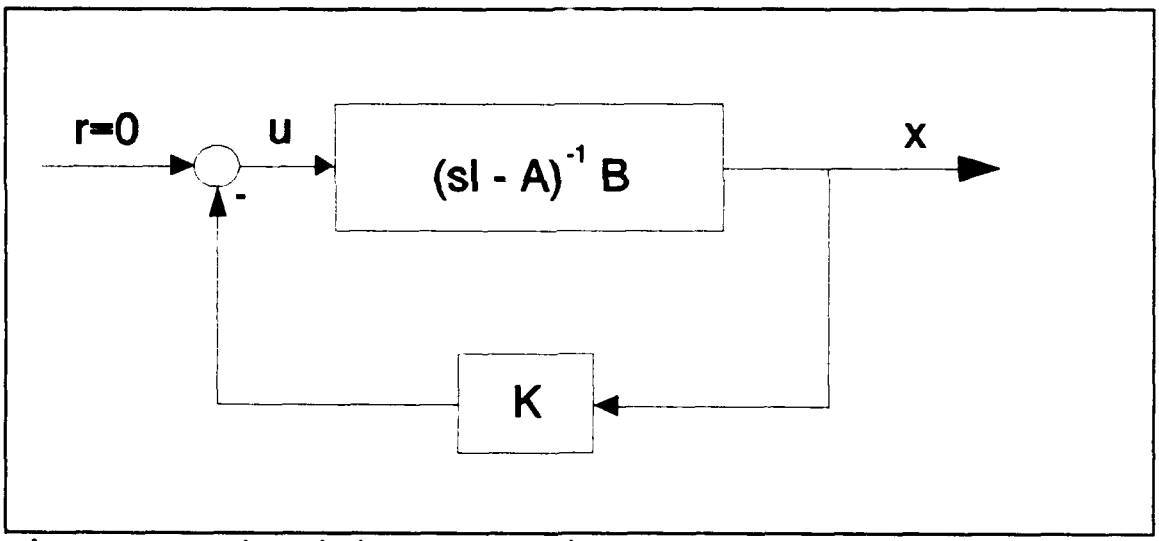

Figure 2 - Simplified Block Diagram for Closed Loop System

For the single input case, the closed loop transfer function for the system in Figure 2 is

$$TF_{CL} = \frac{(j\omega I - A)^{-1} b}{1 + k(j\omega I - A)^{-1} b} \tag{37}$$

The $1+\mathbf{k}(j\omega\mathbf{I}-\mathbf{A})^{-1}\mathbf{b}$ term is referred to as the return difference function, i.e. the output multiplicative difference returned to the input of the plant. Stability for single input systems is typically measured using gain and phase margins [6:section 5.4]. The gain margin is defined as the amount the gain **k** can be changed (increased or decreased) before the closed loop system becomes unstable. The system becomes unstable when the value of the return difference is zero. Phase margin is the amount of phase shift that can be tolerated before the closed loop system becomes unstable.

For multiple input systems, the return difference becomes the matrix $\mathbf{I}+\mathbf{K}(j\omega\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}$. The traditional definitions of gain and phase margin cannot be applied to mulitiple input systems, so some other measure of system stability is required. The concepts of independent gain and phase margins are introduced at this point to provide this measure of stability. Ridgley and Banda [13] give the following definitions.

> Independent gain margins (IGM) are limits within which the gains of all feedback loops may vary independently at the same time without destabilizing the system, while the phase angles remain at their nominal values. Independent phase margins (IPM) are limits within which the phase angles of all loops may vary independently at the same time without destabilizing the system, while gains remain at their nominal values.[13:3-73]

Ridgley and Banda [13:chapter 3] also derive the following equations for IGM and IPM

$$\frac{1}{1+\alpha} < IGM < \frac{1}{1-\alpha} \tag{38}$$

and

$$-2\sin^{-1}\left(\frac{\alpha}{2}\right) < IPM < 2\sin^{-1}\left(\frac{\alpha}{2}\right) \tag{39}$$

where α is the minimum singular value of the return difference matrix given by

$$\alpha = \inf_{\omega} \underline{\sigma}[I+K(j\omega I-A)^{-1}B]$$

and $\alpha \leq 1$. It should be noted that these equations for the MIMO stability margins are based on errors that are multiplicative in nature and they are conservative. A system may be able to accept more gain and phase change than the IGM and IPM reflect.

The following relationship, known as the Kalman Inequality, can be derived using the state equation (8), the optimal gain equation (27), the algebraic Riccati equation (28), and the restriction that $\mathbf{Q} \geq 0$

$$[I+R^{\frac{1}{2}}K(j\omega I-A)^{-1}BR^{-\frac{1}{2}}]^{*}[I+R^{\frac{1}{2}}K(j\omega I-A)^{-1}BR^{-\frac{1}{2}}] \geq I \tag{41}$$

A detailed derivation of the Kalman Inequality is given by Ridgely and Banda [13: chapter 7].

Up to this point the only restriction on **R** is that it be positive definite. First consider the special case where

$\mathbf{R}=\rho I$, where ρ is any scalar value. The Kalman Inequality for this case now reduces to

$$[I+\rho^{\frac{1}{2}}K(j\omega I-A)^{-1}B\rho^{-\frac{1}{2}}]^{*}[I+\rho^{\frac{1}{2}}K(j\omega I-A)^{-1}B\rho^{-\frac{1}{2}}]\geq I \tag{42}$$

Because ρ is a scalar, it cancels out of the equation leaving

$$[I+K(j\omega I-A)^{-1}B]^{*}[I+K(j\omega I-A)^{-1}B]\geq I \tag{43}$$

This inequality is true if and only if equation (44) is true where $\underline{\sigma}[\,]$ indicates the minimum singular value of the matrix inside the brackets which is the return difference matrix in this case

$$\alpha=\underline{\sigma}[I+K(j\omega I-A)^{-1}B]\geq 1 \tag{44}$$

This equation is of the form of equation (33), and therefore is a test for stability robustness. Recalling that the derivations for IGM and IPM are valid for $\alpha \leq 1$, $\alpha = 1$ can be substituted into equations (38) and (39) to determine the minimum limits of the LQR stability margins.

$$\frac{1}{2}<IGM<\infty \tag{45}$$

$$-60^{\circ}<IPM<60^{\circ} \tag{46}$$

Safonov and Athans [15] have shown that any diagonal **R** will result in the same stability margins as long as the perturbations in each channel occur independently of one another. The perturbations can be considered independent as

long as the diagonal elements of **R** have the same relative magnitudes. Equations (45) and (46) are the guaranteed stability margins for LQR with independent perturbations and **R** diagonal.

For the case of any general **R**, the independent stability margins, as calculated by equations (38) and (39), cannot be guaranteed and often will go outside of these bounds. However, as previously mentioned, the equations for IGM and IPM provide conservative values. While the choice of any general **R** may not provide the guaranteed stability margins of equations (45) and (46), the system may still provide acceptable stability characteristics.

III. Robust Eigenstructure Assignment Algorithm

The algorithm developed in this section provides a designer the capability to achieve a closed loop eigenstructure close to desired. The resulting system will also have an eigenstructure within the achievable LQR region. The algorithm places the eigenstructure by minimizing the combined distance between the elements of the desired and LQR achievable eigenstructures. If the desired eigenstructure is not within the allowable LQR region, the algorithm will find the gain matrix, **K**, that achieves a closed loop eigenstructure close to desired. The designer must provide a weighting for each element of the eigenstructure to designate the relative importance of achieving each element. The algorithm is programed in FORTRAN and is designed to be run from the software package MATLAB.

Algorithm Equations

A quadratic performance index, $\bar{J}$, is introduced to be minimized, where

$$\bar{J}=\sum_{i=1}^{n}\left[f_{\lambda_i}(\lambda_{d_i}-\lambda_{a_i})^2+(v_{d_i}-\theta_i v_{a_i})^{*}F_{v_i}(v_{d_i}-\theta_i v_{a_i})\right] \tag{47}$$

and,

n = number of states

f_{λ_i} = weighting on the ith eigenvalue

$\lambda_{d\,i}$ = i^{th} desired eigenvalue

$\lambda_{a\,i}$ = i^{th} achievable eigenvalue

$v_{d\,i}$ = i^{th} desired eigenvector

$v_{a\,i}$ = i^{th} achievable eigenvector

$F_{v\,i}$ = diagonal weighting matrix for the ith eigenvector

θ_i = real or complex constant that minimizes $(v_{d\,i}-\theta_i v_{a\,i})$

Minimizing $\bar{J}$ will minimize the combined distances between the elements of the desired and LQR achievable eigenstructure. The weightings, $f_{\lambda i}$ and $\mathbf{F}_{vi}$, allow the designer to specify the relative importance of achieving individual elements of the eigenstructure. Assigning a weighting of zero to any desired element will leave the algorithm free to place that element to any necessary value.

As already discussed in Chapter II, the closed loop system of Figure 1 can be written as

$$\dot{x}=(A-BK)\,x \tag{48}$$

Using the optimal LQR gain of

$$K=R^{-1}B^{T}P \tag{49}$$

the closed loop system becomes

$$\dot{x}=(A-BR^{-1}B^{T}P)\,x \tag{50}$$

The achievable closed loop poles, λ_i, are the eigenvalues of

$$A_{CL}=(A-BR^{-1}B^{T}P) \tag{51}$$

and the achievable right eigenvectors are given by the

solution of

$$[\lambda_i I-(A-BR^{-1}B^TP)]\,v_i=0 \tag{52}$$

The **A** and **B** matrices are fixed for a given system, so the designer can only influence the closed loop eigenstructure by varying **R** and **P**. However, **P**, the positive definite solution of the algebraic Riccati equation, is a function of both **R** and **Q**. Recall that to ensure that **R**>0 and **Q**≥0, the symmetric matrices **M** and **H** were introduced such that

$$R=M^TM \quad , \quad Q=H^TH \tag{53}$$

The designer's ability to influence the closed loop eigenstructure using LQR design is then limited to varying the symmetric matrices **M** and **H**. Because **M** and **H** are symmetric, the number of paramaters available to be varied is limited to the upper triangular portion of each matrix. If **R** is restricted to ρI or diagonal then the number of paramaters is reduced further.

One further relationship was derived for use with this algorithm. It was shown in Chapter II that the eigenvectors of a matrix can only be determined up to an unknown constant and that constant may be either complex (for complex eigenvectors) or real (for real eigenvectors). Let θ_i represent this unknown constant for the ith eigenvector. Normalizing the eigenvectors to a length of one reduces the possibilities for θ_i to ±1 for the case where the eigenvectors are real. One value will minimize the

eigenvector contribution to $\bar{J}$ and other will maximize the contribution. The normalized eigenvector becomes

$$v_{i_{norm}} = \frac{v_i}{\sqrt{\sum_{k=1}^{n} (v_{i_k})^2}} \tag{54}$$

Complex eigenvectors must also be normed to unit length, but the unknown constant can take on any complex value with a magnitude of one. If the value of θ_i is not determined the algorithm may spend excessive time converging to, or not achieving, the desired eigenvector when it possibly has already found an equivalent vector. The sum of the squared error between the elements of the ith desired and achievable eigenvector can be defined as E_i, given by

$$E_i = [\sum_{k=1}^{n} |v_{d_i} - \theta_i v_{a_i}|^2] \tag{55}$$

where the eigenvector is normalized to a length of one and

$$\theta_i = \pm e^{j\phi_i} = \pm (\cos\phi_i + j\sin\phi_i) \tag{56}$$

To decrease the computational time, it is desirable to minimize E_i with respect to ϕ_i since E_i can be written

$$E_i = \sum_{k=1}^{n} |v_{d_i} - (\cos\phi_i + j\sin\phi_i)\, v_{a_i}|^2 \tag{57}$$

The error, E_i, is an extremum with respect to ϕ_i when its partial derivative is equal to zero, or

$$\frac{\partial E_i}{\partial \phi_i}=0 \tag{58}$$

Seperating each element of the ith desired and achievable eigenvectors into real and imaginary components (denoted by subscripts R and I, the error can be rewritten

$$E_i=\sum_{k=1}^{n}\left[(v_{d_R}-v_{a_R}\cos\phi_i+v_{a_I}\sin\phi_i)^2+(v_{d_I}-v_{a_I}\cos\phi_i-v_{a_R}\sin\phi_i)^2\right] \tag{59}$$

The partial derivative of E_i with respect to ϕ_i can then be evaluated

$$\frac{\partial E_i}{\partial \phi_i}=0=\sum_{k=1}^{n}\left[\sin\phi_i(v_{a_R}v_{d_R}+v_{a_I}v_{d_I})_k+\cos\phi_i(v_{a_I}v_{d_R}-v_{a_R}v_{d_I})_k\right] \tag{60}$$

A solution is now possible for ϕ_i and is given by

$$\phi_i=\tan^{-1}\left[\frac{\sum_{k=1}^{n}(v_{a_R}v_{d_I}-v_{a_I}v_{d_R})_k}{\sum_{k=1}^{n}(v_{a_R}v_{d_R}+v_{a_I}v_{d_I})_k}\right] \tag{61}$$

Equation (56) is then used to calculate θ_i. Both the positive and negative values of θ_i must be checked to determine which minimizes and which maximizes the eigenvector's contribution to $\bar{J}$. The constant θ_i must be calculated for each eigenvector.

All of the required equations are now in place to program the algorithm.

Programing Considerations

The robust eigenstructure assignment algorithm was programmed in FORTRAN (double precision) using both existing and newly developed routines. The main program is called EIGSPACE. The FORTRAN program interfaces with the software package MATLAB through an m-file. The routines used are briefly discussed here.

Existing Subroutines - Several subroutines that were already in existance were used in programming this algorithm. These routines are called by the subroutine EAFUNC. These routines are part of a package of subroutines called LQGLIB on the AFIT computers. The LQGLIB routines are adaptations of routines from Alphatec, Inc and are documented in [16]. The subroutine REG is called by EAFUNC to calculate the LQR optimal gain matrix **K**. The subroutine EIGVV is used to calculate the closed loop eigenstructure. Subroutine MMUL is used to carry out matrix multiplication.

Newly Developed Suboutines - Each routine developed for this algorithm is briefly described here. Appendix A contains more detailed information on each routine, including a listing of each.

- EAFUNC - This subroutine, called by FMINS, calculates the value of the performance index $\bar{J}$. To achieve this other routines are called to solve for the closed loop eigenstructure for a given **M** and **H**.

- FMINS - This subroutine is called by the main program and is based on the Nelder-Mead simplex algorithm [17:298-308]. It calculates a vector **X** that minimizes a given function. The function to be minimized is subroutine EAFUNC in this case.
- FMINSTEP - This subroutine is called by FMINS and is a part of the Nelder-Mead simplex algorithm. It calculates each subsequent iteration in the search to minimize $\bar{J}$.
- MAKEQR - This subroutine is called by EAFUNC to build the positive semidefinite **Q** and positive definite **R** matrices from the vector **X**. The elements of **X** are the upper triangular portions of the symmetric matrices **M** and **H**.
- SORT - This subroutine is called by the main program and EAFUNC. It sorts the eigenvalues in ascending order of magnitude then puts the weighting matrices **Fe** and **Fv** and the modal matrix **W** in the same order.

<u>Program Flow</u> - The program can be broken down into the hierarchical structure shown in Figure 3. The general flow of each level of the program is discussed here.

LEVEL 1 - The function LQREA.m provides the interface between MATLAB and the executable FORTRAN file. Function LQREA.m reads user input from MATLAB, generates an input file, runs the executable FORTRAN file, and reads the FORTRAN output.

```
LEVEL 1 - MATLAB Interface, LQREA.m
    |
    LEVEL 2 - Main FORTRAN Program, EIGSPACE
        |
        LEVEL 3 - Minimization Routine, FMINS
            |
            LEVEL 4 - Minimization Iteration Step
            |         FMINSTEP
            |
            LEVEL 5 - Calculate Performance Index
                      EAFUNC
```

Figure 3 - Computer Program Hierarchy

LEVEL 2 - FORTRAN program EIGSPACE and the subroutines it calls contain the robust eigenstructure assignment algorithm. The program reads the input data file generated by LQREA.m. To simplify the minimization routine, the upper triangular portions of **M** and **H** are put into a single vector **X**. The initial guess, **XGUESS**, is made equivalent to **R**=I and **Q**=I. An initial call is made to FMINS. Because FMINS may find only a local minimum, **XGUESS** is set equal to the returned **X** vector and FMINS is called again. In many cases the second pass through FMINS produces a significant reduction in $\bar{J}$. The improvement on subsequent passes to FMINS is possible because FMINS uses a simplex search technique that does not rely on the calculation of gradients. This will be discussed further in the discussion of LEVEL 3 of the

program. The calls to FMINS continue until the reduction in $\bar{J}$ is within the user specified tolerance. Subroutine EAFUNC is then called using the best **X** vector and the output file is written.

LEVEL 3 - As already mentioned, FMINS is a simplex based minimization routine. An initial simplex is created based on **XGUESS**. The first column of the simplex is **XGUESS**. The other columns are determined by adding an increment to each element of **XGUESS** such that

$$X_j = XGUESS + c_i \qquad j=2,3\ldots(nx+1) \quad , \quad i=1,2 \qquad \textbf{(62)}$$

where nx is the dimension of **X** and c_i is calculated from one of the two following equations

$$c_1 = \frac{a}{nx\sqrt{nx+1}}(\sqrt{nx+1}+nx-1) \quad , \quad c_2 = \frac{a}{nx\sqrt{nx+1}}(\sqrt{nx+1}-1) \qquad \textbf{(63)}$$

In equations (63), a=constant (0.5 for this report). The choice of c_i is defined by Table 1. The values of c_1 and c_2 remain constant for any problem size (nx) so the search in FMINS always starts over a wide range of **X** vectors. The routine converges to a local minimum for the space spanned during the search. Changing XGUESS begins the search over a different set of vectors, possibly spanning additional space, so it is often possible to find a minimum other than the one

TABLE 1
Constants Added to XGUESS to Build Initial Simplex

k \ j	2	3	.	.	.	nx	nx+1
1	c_1	c_2	.	.	.	c_2	c_2
2	c_2	c_1	c_2	.	.	c_2	c_2
.	.	c_2				.	.
.	.	.				.	.
.	.	.				c_2	.
nx-1	c_2	c_2	.	.	c_2	c_1	c_2
nx	c_2	c_2	.	.	c_2	c_2	c_1

achieved during the previous passes through FMINS. It is hoped that enough space is spanned to find the global minimum, but this is not guaranteed. Subroutine EAFUNC is called to calculate the value of the performance index for each column of the simplex. Calls are then made to subroutine FMINSTEP until the convergence criteria of Equation (64) is within the user specified tolerance, tol. The vector $\bar{\mathbf{X}}$ in equation (64) is the average of the **X** vectors in the simplex not including the one yielding the worst performance index and is calculated using equation (65).

$$EJ=\sqrt{\frac{\left[\sum_{j=1}^{nx+1}(X_j-\bar{X})^2\right]-(X_{worst}-\bar{X})^2}{nx}}<tol \qquad \textbf{(64)}$$

$$\overline{X}=\frac{(\sum_{j=1}^{nx+1} X_j)-X_{worst}}{nx} \tag{65}$$

The **X** vector producing the smallest $\bar{J}$ is returned to the main program.

LEVEL 4 - Subroutine FMINSTEP calculates each subsequent itteration in the search to minimize $\bar{J}$. A reflected point is calculated first using the following equation

$$X_{reflected}=\overline{X}+\alpha_r(\overline{X}-X_{worst}) \tag{66}$$

where α_r is a positive constant (a value of 1.0 was used for this report). If $X_{reflected}$ produces the smallest of the current values of $\bar{J}$, an expansion point is calculated

$$X_{expansion}=\overline{X}+\gamma_e(X_{reflected}-\overline{X}) \tag{67}$$

where γ_e is a positive constant (a value of 2.0 was used for this report). The current vector producing the worst value of the performance index is then replaced with the vector (expanded or reflected) producing the lowest value of $\bar{J}$. If the reflected point did not produce the lowest performance index then a contracted point is calculated in one of two ways:

1) If the reflected point produced a $\bar{J}$ higher

than all of the current values

$$X_{contracted}=\overline{X}-\beta_c(\overline{X}-X_{worst}) \tag{68}$$

2) If the reflected point produced a $\bar{J}$ better than the highest value, but not better than the lowest then a contracted point is calculated in the following way

$$X_{contracted}=\overline{X}-\beta_c(\overline{X}-X_{reflected}) \tag{69}$$

The constant β_c must lie between zero and one (a value of 0.5 was used for this report). If the reflected or contracted points produce an improvement in $\bar{J}$ over any of the current points, the vector producing the worst value is replaced by the better of $X_{reflected}$ and $X_{contracted}$. If no improvement has been acheived, then all of the points are moved one half the distance to the best vector.

LEVEL 5 - Subroutine EAFUNC calculates the value of the performance index $\bar{J}$ for a given vector **X**. Recall that **X** contains the upper triangular protions of the symmetric matrices **M** and **H**. The subroutine MAKEQR is called to convert **X** to the matrices **Q** and **R**. The subroutine REG from LQGLIB is called to calculate the optimal feedback gain matrix **K** for the current **R** and **Q** matrices. The subroutine

EIGVV, also from LQGLIB, is then called to calculate the closed loop eigenstructure. The value of $\bar{J}$ is then calculated.

Using the Program

The program is run by using the function LQREA from within MATLAB. The command takes the following form

$$[\boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{P}, \Lambda_a, \boldsymbol{V_a}, \Theta, \bar{J}] = LQREA(\boldsymbol{A}, \boldsymbol{B}, \Lambda_d, \boldsymbol{Fe}, \boldsymbol{V_d}, \boldsymbol{Fv}, tol, rcode)$$

The user must provide the following inputs by defining them in MATLAB

- **A** and **B**
- Λ_d - diagonal matrix containing the desired eigenvalues
- **Fe** - a diagonal matrix containing the weightings for each eigenvalue
- $\mathbf{V}_d$ - the desired modal matrix
- **Fv** - a matrix containing the eigenvector weightings; columns of **Fv** correspond to columns of $\mathbf{V}_d$
- tol - convergence tolerance (default is 10^{-3}).
- rcode - a code to specify what type of **R** matrix to use

  rocde=1, **R**=ρI

  rcode=2, **R**=[diagonal]

  rcode=other, **R**>0

Available outputs from the program are:

- **Q** - LQR state weighting matrix
- **R** - LQR control weighting matrix

- $\mathbf{P}$ - unique positive definite solution to the algebraic Riccati equation
- Λ_a - diagonal matrix containing the acheivable closed loop eigenvalues
- Θ - diagonal matrix containing the eigenvector difference minimization parameter
- $\bar{J}$ - The final value of the performance index

The program normalizes the eigenvectors to one, so to avoid division by zero each column of the desired modal matrix must have at least one nonzero element. The program is configured to handle a system with up to 10 states and up to 10 inputs.

IV. Examples

Two aircraft control sytem examples are examined in this chapter to demonstrate the capability of the robust eigenstructure assignment algorithm. Example 1 is a F-4 lateral inner loop used by Harvey and Stein [10] and later by Robinson [11:554-564]. The algorithm results are compared to results obtained by both Harvey and Stein and by Robinson. Example 2 is a YAH-64 helicopter flight control system used by Garrard and Liebst [3]. The algorithm results are compared to the results obtained by Garrard and Liebst, whose method acheived exact pole placement and good eigenvector assignment but without the stability robustness guarantees of the LQR.

Example 1 - F-4 Lateral Inner Loop

As already mentioned, the algorithm developed in this report is an extension of the work done by Robinson. Robinson compared the results of his pole placement algorithm to results from a method developed by Harvey and Stein using an F-4 example. This example is used here for two purposes: 1) validate the pole placement portion of the FORTRAN program in this report through comparison to Robinson's results, and 2) compare this algorithm's results for complete eigenstructure assignment to Harvey and Stein's complete eigenstructure assignment results. The states and

controls for the F-4 lateral-directional inner loop, as defined in the paper by Harvey and Stein, are

$$\boldsymbol{x} = \begin{bmatrix} p_s \text{ - } \textit{stability axis roll rate} \\ r_s \text{ - } \textit{stability axis yaw rate} \\ \beta \text{ - } \textit{sideslip angle} \\ \phi \text{ - } \textit{bank angle} \\ \delta_r \text{ - } \textit{rudder deflection} \\ \delta_a \text{ - } \textit{aileron deflection} \end{bmatrix}$$

$$\boldsymbol{u} = \begin{bmatrix} \delta_{rc} \text{ - } \textit{rudder command} \\ \delta_{ac} \text{ - } \textit{aileron command} \end{bmatrix}$$

The **A** and **B** matrices are

$$\boldsymbol{A} = \begin{bmatrix} -.746 & .387 & -12.9 & 0 & .952 & 6.05 \\ .024 & -.174 & 4.31 & 0 & -1.76 & -.416 \\ .006 & -.999 & -.0578 & .0369 & .0092 & -.0012 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -20 & 0 \\ 0 & 0 & 0 & 0 & 0 & -10 \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 20 & 0 \\ 0 & 10 \end{bmatrix}$$

Harvey and Stein used the asymptotic properties of LQR to achieve eigenstructure assignment with the control weighting matrix restricted to $\mathbf{R}=\rho\mathrm{I}$. The **Q** matrix is selected and the value of ρ is allowed to approach zero. This forces the feedback gains to become large, approaching infinity. This

is a disadvantage of their method because m of the poles (the actuator poles in this case) approach infinity. They monitor the eigenstructure as ρ tends to zero (or gain tends to infinity) and select the system that produces an eigenstructure close to desired. In the example by Harvey and Stein, they set the open loop actuator poles artifically low (-10 and -5) to compensate for this disadvantage. Except where noted, the actuator poles here are moved to the more accurate values of -20 and -10 to provide a better representation of the actual open loop system.

Comparison to Robinson's Results

The pole placement portion of the algorithm developed in this report differed from Robinson's algorithm only in that this work is programed in FORTRAN. Robinson's algorithm was programed as a MATLAB m-file. Table 2 shows a summary of the pole placement results from this F-4 example for both programs with $\mathbf{R}=\rho\mathbf{I}$. As expected, the results from this algorithm (with eigenvector weighting Fv=[0]) essentially duplicated the results obtained by Robinson. The FORTRAN program was able to acheive a slight improvement in the performance index $\bar{J}$ for this case. All of the other examples reported by Robinson were also run with the FORTRAN program (again, with **Fv**=[0]). The FORTRAN program either duplicated Robinson's results or made a slight improvement in $\bar{J}$ for each case.

TABLE 2
Comparison to Robinson Pole Placement Results
F-4 Example, $R=\rho I$

MODE	λ_d	Robinson MATLAB m-file	FORTRAN Program
Roll Subs.	-4.0	-3.998	-4.004
Dutch Roll	-.63 ± j2.42	-.669 ± j2.365	-.653 ± j2.375
Spiral	-.05	-.091	-.059
Rudder Act.	-20	-20.053	-20.027
Aileron Act.	-10	-10.025	-10.007
$\bar{J}$	--	.014	.006

K MATLAB m-file	-.306	-1.389	.729	.039	.107	-.089
	.409	.858	-.060	.035	-.044	.239
K FORTRAN Porgram	-.304	-1.356	.592	.049	.103	-.077
	.399	.781	.181	.021	-.038	.236

The FORTRAN program produced significant improvements over Robinson's MATLAB based routine in the required computational time. Both routines were run on a VAX computer. For this F-4 example the FORTRAN program took approximately 19 minutes (clock time) to produce the results in Table 2 compared to a clock time in excess of 3 hours using the MATLAB based routine. The FORTRAN also made similiar time improvements on other examples of various size. In general, the FORTRAN program improved required computational time over the MATLAB based routine by a factor of approximately ten.

Comparison to Harvey and Stein Method

Robinson provided a comparison between his pole placement results and the poles achieved with the Harvey and Stein method [10]. Harvey and Stein also attempted to shape eigenvectors with their method. The desired eigenvectors used by Harvey and Stein were not in the achievable LQR region. As a result, a tradeoff is required between the desired poles and the desired eigenvectors. The addition of eigenstructure assignment to the Robinson algorithm now allows for a more valid comparison between this method and the Harvey and Stein method.

Harvey and Stein determined the achievable eigenvectors through a linear projection of the desired vectors onto the achievable subspace. With their method, as the value of ρ approaches zero, the closed loop eigenvectors approach the achievable projection of the desired vectors. The final design is selected as the gain matrix that yields the closed loop poles closest to the desired values.

Some comments are warranted on the selection of desired eigenvectors. In selecting desired eigenvectors, Harvey and Stein split the two eigenvectors for the complex conjugate dutch roll poles into one real vector and one imaginary vector with different elements allowed to float in each vector. For example, the real part of one desired eigenvector element may be allowed to float while the corresponding imaginary part of that element is assigned a

fixed value. The algorithm developed in this report requires that an eigenvector remain intact with the real and imaginary parts of each element either both designated or both allowed to float. As the following results will indicate, this does not proves to be a disadvantage. With these comments in mind, Table 3 shows the desired eigenvectors for this F-4 example.

TABLE 3
Desired Eigenvectors for F-4 Lateral Inner Loop

State	Roll Subs.	Dutch Roll	Spiral	Rudder[2] Actuator	Aileron[2] Actuator
p_s	1	0	x	x	x
r_s	0	.707[3] ± j0	x	x	x
β	0	.707 ± j0[3]	0	x	x
ϕ	x	0	1	x	x
δ_r	x	x	x	x	x
δ_a	x	x	x	x	x

Notes: 1) x's denote elements that are free to float. These values were set to zero in most cases and given zero weighting in all cases to run the FORTRAN program (see note 2)
2) Actuator eigenvectors were given weightings of zero in all cases. Values specified for these eigenvectors are to prevent division by zero in the algorithm.
3) Harvey and Stein allow these elements to float.

Harvey and Stein selected these desired eigenvectors for the following reasons. It is desirable to have the roll subsidence mode show up predominatly in roll and not in yaw, so the elements corresponding to yaw rate and sideslip angle are set to zero. For the dutch roll mode, Harvey and Stein

desired no oscillatory motion in the roll axis, thus the desired elements corresponding to roll angle and roll rate are set to zero. Harvey and Stein desired no sideslip in the spiral mode, so that element was set to zero for the spiral mode eigenvector.

Tables 4 and 5 summarize the results obtained using the robust eigenstructure assignment algorithm and the results from the Harvey and Stein method.

TABLE 4
Results from Robust Eigenstructure Assignment Algorithm for F-4 Example ($R=\rho I$)
Open Loop Actuator Poles of -10 and -5

	Roll Subs.	Dutch Roll	Spiral	Rudder Actuator	Aileron Actuat.
λ_a	-3.994	-.678±j2.383	-.051	-20.001	-10.001
Achievable Eigenvectors					
p_s	.850	-.020±j.005	-.051	.072	-.518
r_s	-.017	.606∓j.098	.037	-.089	-.002
β	-.004	.009±j.228	.008	-.004	-.000
ϕ	-.213	.002±j.008	.998	-.004	-.052
δ_r	.078	.251∓j.422	.015	-.990	-.212
δ_a	-.463	.132±j.550	.007	-.078	.827
$\bar{J}_{vec}$=.874		$\bar{J}_{\lambda}$=.007		$\bar{J}$=.881	

$$K = \begin{matrix} -.014 & -.714 & 1.126 & -.001 & .548 & .072 \\ .525 & .406 & -2.934 & .026 & .065 & .846 \end{matrix}$$

For this comparison, the original Harvey and Stein open loop actuator poles of -10 and -5 were used. The desired poles remain as shown in Table 2. Unity weightings were assigned

TABLE 5
Results from Harvey and Stein Method for F-4 Example

	Roll Subs.	Dutch Roll	Spiral	Rudder Actuator	Aileron Actuat.
λ_a	-3.810	-.727±j2.358	-.049	-22.44	-10.43
Acheivable Eigenvectors					
p_s	.857	-.087∓j.081	-.049	-.035	-.523
r_s	-.005	.610∓j.068	.037	.078	.027
β	-.001	.093±j.231	.000	.003	.003
ϕ	-.226	-.021±.043	.998	.002	.050
δ_r	.105	.295∓j.400	-.001	.996	-.044
δ_a	-.451	.145±j.526	-.007	-.029	.849
$\bar{J}_{vec}$=.94		$\bar{J}_{\lambda}$=6.17		$\bar{J}$=7.11	
K	.132	.882	-1.576	-.026	-.681	.026
	-.524	-.420	2.827	-.021	.031	-.860

Notes: 1) All eigenvectors normed to one.
2) Dutch roll eigenvectors are multiplied by θ_i as calculated by equations (56) and (61).
3) Performance index calculations done using equations (47) using the same weightings used for the robust eigenstructure assignment results.

to each desired element with a fixed value. Desired elements of the eigenvectors that were allowed to float were assigned weightings of zero. Thus, the following weightings were used in the robust eigenstructure assingment algorithm

$$\boldsymbol{Fe} = [1\ 1\ 1\ 1\ 1\ 1] \qquad \boldsymbol{Fv} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Comparison of the results from the two methods shows that the robust eigenstructure assignment algorithm yielded

values much closer to desired than the Harvey and Stein method. In particular, the robust eigenstructure assignment algorithm achieved poles and eigenvectors much closer to desired than the Harvey and Stein results. The values of $\bar{J}_{vec}$ in Tables 4 and 5 are the eigenvector contributions to the performance index $\bar{J}$ in equation (47). Likewise, the values of $\bar{J}_{\lambda}$ are the closed loop pole contributions to equation (47). The large value of $\bar{J}_{\lambda}$ in Table 5 is mainly due to the large difference between the Harvey and Stein desired and achievable actuator poles.

The use of the more realistic open loop actuator poles of -20 and -10 in the Harvey and Stein method would likely result in extremely large values for the closed loop actuator poles. The robust eigenstructure assignment method developed in this thesis does not suffer from the same deficiency. The algorithm was run again for $\mathbf{R}=\rho\mathbf{I}$ with unity weighting using the realistic open loop actuator poles. The results for this case are shown in Table 6. The non-zero value of $\bar{J}$ in Table 6 shows that the desired eigenstructure was not achievable using the algorithm. Assuming that the algorithm was able to find a global minimum for $\bar{J}$, this indicates that the desired eigenstructure in not within the achievable LQR region. However, it may be possible to improve on the closed loop eigenstructure through variations on the weightings assigned to the various eigenstructure elements.

TABLE 6
Results from Robust Eigenstructure Assignment Algorithm for F-4 Example ($R=\rho I$)
Open Loop Actuator Poles of -20 and -10

	Roll Subs.	Dutch Roll	Spiral	Rudder Actuator	Aileron Actuat.
λ_a	-3.996	-.955±j2.088	-.070	-20.158	-10.002
Achievable Eigenvectors					
p_s	.845	-.163±j.035	-.067	.039	.543
r_s	.005	.563∓j.193	.037	-.086	-.035
β	.001	.174±j.193	.002	-.004	-.004
ϕ	-.211	.043±j.058	.997	-.002	-.054
δ_r	.137	.382∓j.382	.003	-.995	.003
δ_a	-.473	.269±j.426	-.007	.027	-.837
$\bar{J}_{vec}$=.848		$\bar{J}_{\lambda}$=.457		$\bar{J}$=1.305	

$$K = \begin{matrix} -.175 & -1.089 & .951 & .022 & .089 & -.072 \\ .539 & .562 & -2.649 & .031 & -.036 & .336 \end{matrix}$$

When the desired eigenstructure lies outside of the LQR achievable region, the designer must make tradeoffs between the desired eigenstructure elements. Because not all of the eigenstructure elements lie within the achievable subspace for this case, varying the weightings to place greater emphasis on any particular elements will produce a tradeoff. Weighting the dutch roll poles more heavily than the other elements will move those poles closer to desired, but will result in other elements moving further from desired. If the pole location is of greater importance than the decoupling of the states in the dutch roll eigenvectors, then those poles should be weighted more heavily than the

other elements. Table 7 presents results from the algorithm with weightings of 10 placed on each dutch roll pole and all other weightings remaining the same as before. The closed loop dutch roll poles moved much closer to desired, but the decoupling of roll and yaw was sacrificed in the dutch roll eigenvectors. Increasing the weighting on the dutch roll poles also moved the other poles a small distance further away from desired. Further variations on the weightings between the poles and eigenvectors resulted in similar tradeoffs between elements.

TABLE 7
Results from Robust Eigenstructure Assignment Algorithm for F-4 Example with Heavier Weighting on Dutch Roll Poles ($R=\rho I$)

	Roll Subs.	Dutch Roll	Spiral	Rudder Actuator	Aileron Actuat.
λ_a	-4.023	-.703±j2.331	-.154	-20.321	-10.053
Achievable Eigenvectors					
p_s	.793	-.664±j.225	-.151	-.053	.519
r_s	.085	.459∓j.163	.032	.087	-.001
β	.021	.117±j.163	-.030	.004	-.001
ϕ	-.197	.167±j.234	.983	.003	-.052
δ_r	.354	.176∓j.291	-.058	.995	.197
δ_a	-.446	.102±j.148	-.071	.017	-.830
$\bar{J}_{vec}=2.124$		$\bar{J}_\lambda=.380^*$		$\bar{J}=2.504$	

$$K = \begin{matrix} -.310 & -1.194 & .947 & .081 & .101 & -.056 \\ .487 & .819 & -1.314 & .099 & -.028 & .297 \end{matrix}$$

* The error for dutch roll poles is multiplied by weights of 10.

Removing the $R=\rho I$ restriction allows the robust eigenstructure assignment algorithm to place the closed loop eigenstructure more closely to the desired structure. Table 8 presents results for the R=[diagonal] case, and Table 9 presents results for the case where R>0. In both cases the eigenstructure weightings used were the same as those used for the $R=\rho I$ case.

TABLE 8
Results from Robust Eigenstructure Assignment Algorithm for F-4 Example with R=[diagonal]

	Roll Subs.	Dutch Roll	Spiral	Rudder Actuator	Aileron Actuat.
λ_a	-3.997	-.811±j2.095	-.054	-20.169	-10.004
Achievable Eigenvectors					
p_s	.851	-.054±j.016	-.054	-.061	-.541
r_s	-.026	.571∓j.209	.037	.088	.030
β	-.006	.173±j.209	.003	.004	.003
ϕ	-.213	.015±j.020	.998	.003	.054
δ_r	.053	.316∓j.361	.004	.993	-.033
δ_a	-.477	.278±j.496	-.004	-.043	.838
J_{vec}=.878		J_λ=.240		J=1.118	
K	-.068	-.877	.751	.022	.078	.005
	.519	.448	-2.78	.024	.033	.330

For the case where R is allowed to be diagonal, Table 8 shows that the algorithm was able to achieve closed loop poles much closer to desired (J_λ=.240) than the $R=\rho I$ case (J_λ=.457) while still yielding nearly the same eigenvector

decoupling. For this case the algorithm returned a R matrix with significantly different diagonal elements (order of magnitude 10). The values returned were

$$\boldsymbol{R} = \begin{bmatrix} 41.588 & 0 \\ 0 & 3.332 \end{bmatrix}$$

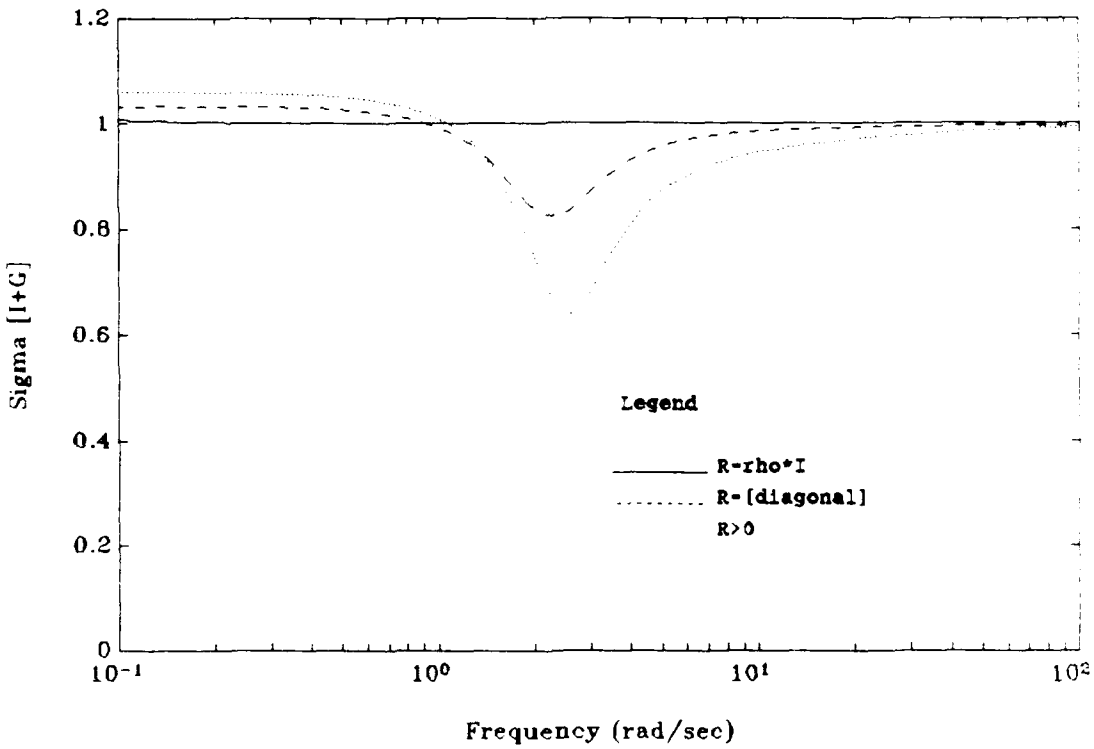

Figure 4 F-4 Example Minimum Singular Values for Variations of R Matrix

As discussed in Chapter II, the large difference in the two diagonal elements of **R** indicates that the perturbations cannot be considered to reside in each individual channel and the independent stability margins of equations (45) and (46) not are guaranteed. Figure 4 shows a plot of the minimum singular value of return difference matrix for the

case where **R**=[diagonal] as well as for **R**=ρ**I** and **R**>0. This figure shows that the diagonal **R** does not possess the independent stability margins guaranteed by **R**=ρ**I**. The stability margins for case **R**=[diagonal] are IGM=[.545,6.024] and IPM=[-49.3,49.3] degrees.

The results for the case where **R**>0 were as expected. As can be seen from Table 9, this case acheived the best overall placement of the closed loop eigenstructure.

TABLE 9
Results from Robust Eigenstructure Assignment Algorithm for F-4 Example with **R**>0

	Roll Subs.	**Dutch Roll**	**Spiral**	**Rudder Actuator**	**Aileron Actuat.**
λ_a	**-3.992**	**-.675±j2.376**	**-.054**	**-20.005**	**-10.003**
Achievable Eigenvectors					
p_s	.837	-.014±j.004	-.054	-.051	-.543
r_s	-.025	.578∓j.207	.037	.088	.034
β	-.007	.139±j.207	.003	.004	.004
ϕ	-.210	.003±j.005	.998	.003	.054
δ_r	.193	.170∓j.457	.004	.995	-.009
δ_a	-.467	.230±j.520	-.004	-.008	.837
$\bar{J}_{vec}$=.878		$\bar{J}_\lambda$=.008		$\bar{J}$=.886	

K

-.265	-.761	1.748	.004	.048	-.143
.544	.445	-3.034	.026	.006	.347

The control weighting matrix returned by the algorithm for this case was

$$\boldsymbol{R} = \begin{bmatrix} 3.838 & 2.303 \\ 2.303 & 2.499 \end{bmatrix}$$

The cost for this improvement was a decrease in the independent stability margins. The plot of Figure 4 shows that this case produced the lowest minimium singular value of the return difference matrix of all of the cases. The stability margins for this case are IGM=[.611,2.745] and IPM=[-37.1,37.1] degrees.

Example 2 - YAH-64 Type Helicopter

Garrard and Liebst [3] used a flight control system similar to the YAH-64 helicopter at hover flight conditions to demonstrate the use of their eigenstructure assignment method. Their method achieved exact pole placement and minimized the quadratic performance index given by

$$J_{es_i} = (v_{ai} - v_{di})^* P_i (v_{ai} - v_{di}) \tag{22}$$

to place the closed loop eigenvectors close to desired. The Garrard and Liebst method does not provide any stability guarantees.

The eight states and four controls for this helicopter model are defined as follows

$$\boldsymbol{x} = \begin{bmatrix} u \text{ - } \textit{forward velocity (ft/sec)} \\ v \text{ - } \textit{lateral velocity (ft/sec)} \\ w \text{ - } \textit{downward velocity (ft/sec)} \\ p \text{ - } \textit{roll rate (rad/sec)} \\ q \text{ - } \textit{pitch rate (rad/sec)} \\ r \text{ - } \textit{yaw rate (rad/sec)} \\ \phi \text{ - } \textit{roll angle (rad)} \\ \theta \text{ - } \textit{pitch angle (rad)} \end{bmatrix}$$

$$\boldsymbol{u} = \begin{bmatrix} u_1 \text{ - } \textit{collective pitch } (^\circ) \\ u_2 \text{ - } \textit{longitudinal cyclic pitch } (^\circ) \\ u_3 \text{ - } \textit{lateral cyclic pitch } (^\circ) \\ u_4 \text{ - } \textit{tail rotor collective pitch } (^\circ) \end{bmatrix}$$

The states and controls are non-dimensionalized by dividing by the maximum expected values at hover conditions. This

results in the following non-dimensional state and control vectors

$$\boldsymbol{x}_{nd}=\boldsymbol{T}_1\boldsymbol{x} \quad , \quad \boldsymbol{u}_{nd}=\boldsymbol{T}_2\boldsymbol{u} \tag{70}$$

where T_1 and T_2 are transformation matrices and are given by

$$\boldsymbol{T}_1 = \begin{bmatrix} .04 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .04 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .04 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.865 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.865 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.865 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2.865 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.865 \end{bmatrix}$$

$$\boldsymbol{T}_2 = \begin{bmatrix} .111 & 0 & 0 & 0 \\ 0 & .067 & 0 & 0 \\ 0 & 0 & .114 & 0 \\ 0 & 0 & 0 & .054 \end{bmatrix}$$

Equations (70) can be substituted into equations (8) and (10) resulting in

$$\dot{\boldsymbol{x}}_{nd}=\boldsymbol{T}_1\boldsymbol{A}\boldsymbol{T}_1^{-1}\boldsymbol{x}_{nd}+\boldsymbol{T}_1\boldsymbol{B}\boldsymbol{T}_2^{-1}\boldsymbol{u}_{nd} \quad , \quad \boldsymbol{u}_{nd}=-\boldsymbol{T}_2\boldsymbol{K}\boldsymbol{T}_1^{-1}\boldsymbol{x}_{nd} \tag{71}$$

The non-dimensional **A**, **B**, and **K** matrices then become

$$\boldsymbol{A}_{nd}=\boldsymbol{T}_1\boldsymbol{A}\boldsymbol{T}_1^{-1} \quad , \quad \boldsymbol{B}_{nd}=\boldsymbol{T}_1\boldsymbol{B}\boldsymbol{T}_2^{-1} \quad , \quad \boldsymbol{K}_{nd}=\boldsymbol{T}_2\boldsymbol{K}\boldsymbol{T}_1^{-1} \tag{72}$$

$$\boldsymbol{A}_{nd}=\begin{bmatrix} -.0286 & -.0637 & .0205 & .0032 & .1113 & -.0036 & 0 \\ .0779 & -.2310 & .0059 & -.1157 & -.0014 & -.0229 & .4468 \\ .0046 & -.0257 & -.2610 & -.0053 & .0314 & .0306 & .0223 \\ .5658 & -3.5812 & .6804 & -2.7000 & -.1340 & -.6620 & 0 \\ .3366 & .8458 & .0143 & -.0092 & -.7500 & .0244 & 0 \\ .2793 & -.3510 & .0573 & -1.0500 & .4130 & -.4000 & 0 \\ 0 & 0 & 0 & 1.000 & -.0051 & .1030 & 0 \\ 0 & 0 & 0 & 0 & .9990 & .0499 & 0 \end{bmatrix}$$

$$B_{nd}=\begin{bmatrix} .1566 & .3456 & -.0399 & -.0007 \\ -.0569 & .0812 & .1718 & .2087 \\ -.15372 & .0345 & -.0087 & .0009 \\ -1.1294 & -2.5785 & 16.2195 & 4.2402 \\ .1857 & -4.3405 & -2.2562 & -.1007 \\ 2.0628 & .4169 & 5.0137 & -2.4116 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Table 10 shows the resulting open loop eigenstructure. Notice that the four poles associated with side and forward velocity are unstable. Also, the yaw and heave modes form a complex conjugate pair which will result in cyclic motion coupled in the yaw and vertical axes. In the open loop eigenvectors there is significant coupling between the states.

TABLE 10
Open Loop Eigenstructure for Helicopter Example

	Roll	Pitch	Side Velocity	Forward Velocity	Yaw/Heave
λ	-3.2610	-.9760	.0820±j.6296	.1100±j.5147	-.2588±j.0428
Open Loop Eigenvectors					
u	-.0024	.3605	-.0462±j.3103	-.0462±j.4723	-.0244∓j.1983
v	-.0784	.0102	-.2171±j.1269	-.0979±j.1297	.0616±j.0923
w	-.0012	.0525	.0040±j.0241	.0082±j.0341	-.0949±j.5468
p	-.8923	.0373	.1334∓j.2352	-.0066∓j.1424	.0850±j.0084
q	.0268	-.6131	-.0234±j.2872	.0666±j.2828	.0488±j.0270
r	-.3407	.3338	.3382±j.4070	.4026±j.2892	-.7740∓j.1489
ϕ	.2844	-.0767	-.2699∓j.3026	-.1982∓j.1093	-.0231±j.0234
θ	-.0030	.6104	.4785±j.0727	.5862∓j.0431	-.0257∓j.0799

<u>Desired Eigenstructure</u> - The desired eigenstructure selected by Garrard and Liebst is shown in Table 11.

Garrard and Liebst selected desired eigenvalues based on helicopter handling qualities requirements put forth by Hoh [18]. The real desired eigenvalues were selected to provide bandwidths of 3.0 rad/sec for roll, 2.9 rad/sec for pitch, 3.0 rad/sec for yaw, and 1.0 rad/sec for vertical velocity (heave).

TABLE 11
Desired Eigenstructure for Helicopter Example

	Roll	Pitch	Side Velocity	Forward Velocity	Heave	Yaw
λ_d	-3.5	-2.9	-.802±j.388	-.801±j.387	-1.0	-3.0
Desired Eigenvectors						
u	0	x	0	1	0	0
v	x	0	1	0	0	0
w	0	0	0	0	1	0
p	.9615	0	x	0	0	0
q	0	.9454	0	x	0	0
r	0	0	0	0	0	1
ϕ	-.2747	0	x	0	0	0
θ	0	-.3260	0	x	0	0

Note: x denotes elements free to float

Selection of the desired eigenvectors was made to provide decoupling of the states in the various mode responses. For the eigenvector associated with roll, the roll rate element was given a magnitude of one. The bank angle was set to the inverse of the roll eigenvalue due to assuming a response of the form

$$\phi = ce^{\lambda t} \quad so \quad \dot{\phi} = p = \lambda ce^{\lambda t} \quad then \quad \phi = \frac{p}{\lambda}$$

The side velocity element for the roll eigenvector is allowed to float because during rolls some sideslip is inevitable and not considered objectionable. The other elements of the roll eigenvector are set to zero to minimize the content of those states in the roll response. The desired pitch mode eigenvector was selected based on the same reasoning used for the roll mode eigenvector. The eigenvectors associated with the two complex conjugate eigenvalue pairs were selected to decouple the lateral and longitudinal states in those modes. The heave and yaw eigenvectors were selected to produce modes with pure heave and pure yaw respectively.

Garrard and Liebst Results - Table 12 shows the final achievable eigenvectors and feedback gain matrix obtained by Garrard and Liebst for full state feedback. Recall that all of the poles were placed exactly with their method. The eigenvectors in Table 11 show excellent decoupling between the states. The independent stability margins resulting from this design are IGM=(.599,3.03) and IPM=(-39.1,39.1) degrees.

The results that follow show that the desired eigenstructure does not lie within the LQR region obtainable with the robust eigenstructure assignment algorithm. The objective of using the robust eigenstructure assignment

TABLE 12
Garrard and Liebst Helicopter Example Results for Full State Feedback

	Roll	Pitch	Side Velocity	Forward Velocity	Heave	Yaw
Achievable Eigenvectors						
u	.0024	-.1381	.0118∓j.0303	.4143±j0	-.1003	-.0112
v	.0890	-.0235	.4737±j0	.0093±j.0332	-.0181	-.0366
w	0	-.0293	-.0376±j.0193	-.0024±j0	.9939	-.0028
p	.9577	.0041	.3720±j.4473	.0408±j.0377	.0089	-.0065
q	.0001	.9357	.0084±j.0581	-.4350∓j.4154	-.0290	-.0073
r	-.0002	-.0036	-.0028±j.0056	-.0192∓j.0085	-.0014	.9986
ϕ	-.2736	.0003	-.5943∓j.2705	-.0606∓j.0194	-.0089	-.0321
θ	0	-.3223	-.0369∓j.0549	.6440±j.2075	.0290	-.0142

$$\mathbf{K}_{nd} \quad \begin{matrix} -.1503 & -.0403 & -.5006 & -.0047 & .0356 & -.0225 & -.0747 & .1554 \\ .9957 & -.2973 & .0060 & -.0096 & -.7554 & -.0873 & -.1458 & -1.2572 \\ .1794 & -.0131 & .0957 & .0138 & -.0278 & .1674 & .2489 & -.1189 \\ .3170 & .0018 & -.2408 & .4557 & -.3460 & -.7669 & .4082 & -.3841 \end{matrix}$$

Note: All values in this table are nondimensionalized

algorithm with this example is then to obtain an eigenstructure close to desired but with improved stability margins over the Garrard and Liebst achievable closed loop eigenstructure.

<u>Robust Eigenstructure Assignment Algorithm Results</u> - To determine if the desired poles were achievable, this case was first run with zero weighting on the eigenvectors ($\mathbf{Fv}$=[0]). The algorithm was able to place all of the poles very near the desired values ($\bar{J}$=.0002). Next, all of the desired eigenvector elements with assigned values were given unity weighting as well to determine if the complete desired eigenstructure was within the achievable region. Table 12 presents results for this case with $\mathbf{R}>0$. A fully populated R matrix was selected to allow maximum flexibility in

placing the eigenstructure. The results in Table 13 show that the complete eigenstructure is not achievable using the robust eigenstructure assignment algorithm. The closed loop poles in Table 13 are relatively far from desired considering they are known to be nearly achievable. The roll, pitch and yaw poles are nearly identical for this case. The achievable eigenvectors also show much more coupling between the states than desired. Variations on the weightings used for the algorithm can be used to get a more acceptable closed loop eigenstructure.

TABLE 13
Robust Eigenstructure Assignment Algorithm Results for Helicopter Example with Unity Weighting and R>0

	Roll	Pitch	Side Velocity	Forward Velocity	Heave	Yaw
λ_a	-3.3430	-3.3137	-.8957±j.6347	-.7272±j.1326	-1.249	-3.3142
Achievable Eigenvectors						
u	.0181	-.1128	-.0008±j.0023	.4955∓j0	-.1429	.0782
v	.0755	-.0011	.3545∓j0	-.0622∓j.1200	-.0302	.0554
w	.0119	-.0354	.0142∓j.0004	-.1806±j.0734	.9770	.0283
p	.8698	.1053	.2546∓j.6122	-.0304∓j.0727	-.0071	.5633
q	-.1583	.8920	.0297∓j.0436	-.4722±j.1338	.0569	-.6276
r	.3697	-.3303	.0628∓j.1918	.0012±j.0038	.1351	.4597
ϕ	-.2718	-.0201	-.5267±j.3321	.0194±j.1039	-.0052	-.1852
θ	.0418	-.2640	-.0524±j.0022	.6602∓j.0637	-.0509	.1822
$\bar{J}_\lambda$=.636		$\bar{J}_{vec}$=3.011			$\bar{J}$=3.647	

K_{nd}

.2349	.0379	-.7122	-.0958	.1354	.2151	-.0366	.1903
.5684	-.4127	.0122	-.0243	-.8210	-.1037	-.2399	-1.0973
.6354	.1957	.1808	.1223	-.1646	.0106	.4657	-.5214
.0286	-.3630	-.3567	.2135	-.2264	-.6013	-.1248	-.1785

Note: $\bar{J}_\lambda$ represents the eigenvalue contribution to performance index $\bar{J}$ and $\bar{J}_{vec}$ represents the eigenvector contribution to $\bar{J}$.

One point of interest is worth discussion prior to proceeding with variations on the weightings. The R matrix returned by the algorithm for this example is relatively close to diagonal and the diagonal elements are of the same relative magnitude. The R matrix returned is

$$R=\begin{bmatrix} 1.2636 & .1027 & .1968 & -.0817 \\ .1027 & 1.5340 & -.2174 & -.1977 \\ .1968 & -.2174 & 2.1366 & .1330 \\ -.0817 & -.1977 & .1330 & 1.1704 \end{bmatrix}$$

The significance of this R matrix is that because it comes close to approximating a diagonal matrix, the minimum singular value of the return difference matrix is nearly one. Figure 5 shows a plot of the minimum singular values of the return difference matrix for this case. The minimum singular value of .992 for this system yields good independent stability margins of IGM=(.502,125.0) and IPM=(-59.5,59.5). The use of R>0 allows the maximum flexibility in placing the closed loop eigenstructure within the LQR limitations. It turns out that R being near diagonal is a general result for all of the cases run with this example for R>0. Therefore, the flexibility of R>0 can be utilized and the resulting closed loop systems will still possess good independent stability margins.

The results in Table 13 provide insight as to what variations on the weightings should be attempted. The location of the closed loop poles will determine the speed and type of the vehicle's response and have a significant

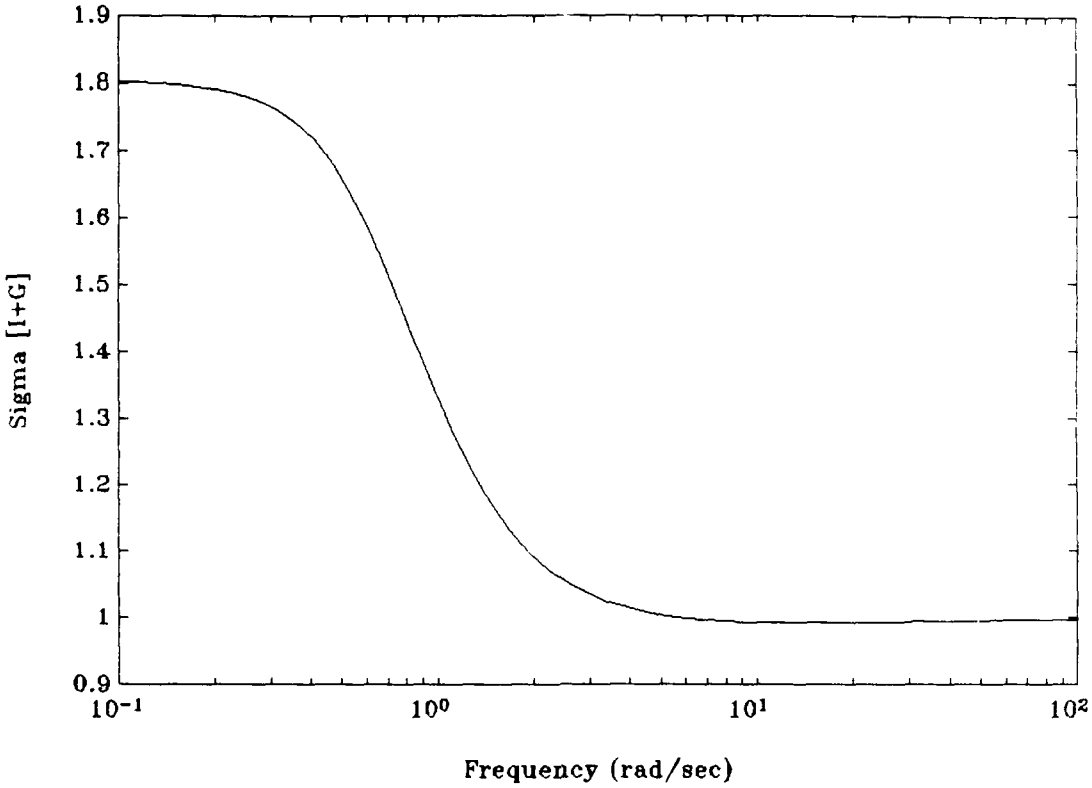

Figure 5 Helicopter Example Minimum Singular Values for Unity Weighting Case with R>0

impact on handling qualities, making it important to move them closer to desired. Increasing the weighting for the desired poles will move the achievable poles closer to desired. It appears that the desired amount of decoupling may be possible for the heave, side velocity, and forward velocity eigenvectors. The roll, pitch, and yaw eigenvectors show significant coupling in all three axes which is not surprising since the three poles are almost identical. Recall that because the complete eigenstructure is not achievable, a tradeoff between the various elements is probably required. Weighting the roll, pitch, and yaw

eigenvectors more heavily may result in better decoupling in those vectors, but it also may move the poles further from desired or cause increased coupling in the other eigenvectors. On the other hand, decreasing the weighting on the roll, pitch and yaw eigenvectors will likely allow the other elements of the eigenstructure to move closer to desired while possibly still providing some decoupling in these eigenvectors. These considerations lead to using the following weightings for the next case:

$$\boldsymbol{Fe}=[4\ 4\ 4\ 4\ 4\ 4\ 4\ 4] \qquad \boldsymbol{Fv}=\begin{bmatrix} .5 & 0 & 1 & 1 & 1 & 1 & 1 & .5 \\ 0 & .5 & 1 & 1 & 1 & 1 & 1 & .5 \\ .5 & .5 & 1 & 1 & 1 & 1 & 1 & .5 \\ .5 & .5 & 0 & 0 & 1 & 1 & 1 & .5 \\ .5 & .5 & 1 & 1 & 0 & 0 & 1 & .5 \\ .5 & .5 & 1 & 1 & 1 & 1 & 1 & .5 \\ .5 & .5 & 0 & 0 & 1 & 1 & 1 & .5 \\ .5 & .5 & 1 & 1 & 0 & 0 & 1 & .5 \end{bmatrix}$$

Table 14 presents the results for this first variation on the weightings. As expected, the poles are closer to desired, yielding a much smaller contribution to $\bar{J}$ [$\bar{J}_\lambda$=4(.056)]. The achievable heave, side velocity, and forward velocity eigenvectors are also slightly improved over the unity weighting case (Table 13). The lower weighting on the roll, pitch, and yaw eigenvectors resulted in values further from desired than the unity weighting case. The eigenvector associated with the roll mode shows a large roll yaw rate component. The pitch mode eigenvector

has large components associated with roll rate and yaw rate and also has some undesired roll angle content. The yaw mode eigenvector has a large pitch rate component and a moderate roll rate component.

TABLE 14
Helicopter Example Results with R>0
First Weighting Variation
Weights:
Poles - 4, Eigenvectors - .5 (roll, pitch, yaw), 1 others

	Roll	Pitch	Side Velocity	Forward Velocity	Heave	Yaw
λ_a	-3.5254	-2.9439	-.7804±j.4746	-.7284±j.3339	-1.0085	-2.8550
Achievable Eigenvectors						
u	-.0077	-.0945	.0092±j.0157	.4552∓j0	-.0990	.0515
v	.0600	.0517	.4505∓j0	-.0348∓j.0454	-.0257	.0002
w	-.0271	.0127	.0366±j.0085	-.0009±j.0187	.9927	-.0695
p	.8489	.4558	.3020∓j.5100	.0017∓j.0070	-.0002	.1659
q	.0568	.6796	.0113∓j.0396	-.4111±j.3742	-.0304	-.3503
r	.4546	-.5006	-.1504∓j.0451	.0238∓j.0594	-.0465	.9069
ϕ	-.2540	-.1361	-.5606±j.3182	-.0149±j.0137	.0048	-.0915
θ	-.0225	-.2221	-.0274±j.0369	.6575∓j.2078	.0324	.1067
$\bar{J}_\lambda$=4(.056)=.223		$\bar{J}_{vec}$=1.314+.5(.922)=1.775			$\bar{J}$=1.998	

K_{nd}

-.1002	.1096	-.4898	-.0251	.0881	.1128	-.0024	.1568
.6988	-.3288	.0293	-.0454	-.6998	-.0659	-.1979	-1.0365
.4479	.1377	.1349	.0943	-.1489	.1166	.3970	-.3809
-.5061	-.4662	-.3692	.1929	-.0500	-.6003	-.0776	.3729

Note: $\bar{J}_\lambda$ represents the eigenvalue contribution to performance index $\bar{J}$ and $\bar{J}_{vec}$ represents the eigenvector contribution to $\bar{J}$.

The achievable poles in Table 14 show a minor problem encountered in using the algorithm. Recall that the desired poles for the pitch and yaw modes were -2.9 and -3.0 respectively. The desired pitch mode pole has the smaller magnitude of the two, but the results in Table 14 show that the algorithm matched the lower magnitude achievable pole

(-2.855) with the yaw mode. The reason for this discrepancy is that the algorithm matches desired and achievable poles by starting with the lowest magnitude desired pole and finding the minimum error between that desired pole and all of the achievable poles. The matched poles are eliminated from consideration, and the algorithm then matches the desired pole with the next larger magnitude by comparing the error between it and all of the remaining achievable poles. In this case, the two achievable poles for pitch and yaw lie very close together. The algorithm matched the desired pitch mode pole first because it has the smaller magnitude and the achievable pole closest to it was the one with the larger magnitude. Thus, the desired pitch mode pole was matched with the larger value, leaving the smaller magnitude achievable pole to be matched with the yaw mode desired pole.

The results for one final weighting variation with this example are shown in Table 15 to demonstrate the necessity of trading off between the requirements for the various elements of the desired eigenstructure. For this case the eigenvalues were again assigned weights of four while the roll mode eigenvector elements were given weights of 0.5. Elements of all other eigenvectors were assigned weightings of unity.

Comparison of the Table 15 results to the results for the first weighting variation in Table 14 reveals that the

TABLE 15
Helicopter Example Results with R>0
Second Weighting Variation
Weights - Poles: 4, Eigenvectors: .5 (roll), 1 others

	Roll	Pitch	Side Velocity	Forward Velocity	Heave	Yaw
λ_a	-3.5638	-2.8680	-.7197±j.3490	-.8593±j.3178	-.9867	-2.9933
Achievable Eigenvectors						
u	-.0104	-.1276	-.0029±j.0258	.4480±j0	-.0938	.0051
v	.0516	-.0081	.4624±j0	-.0434∓j.0418	-.0127	-.0308
w	-.0437	-.1109	.0535±j.0077	-.0097±j.0227	.9937	-.1565
p	.8032	.1094	.4532∓j.3726	-.0111±j.0059	.0167	-.0109
q	.0844	.9240	.0221∓j.0564	-.3944±j.3862	-.0382	.0084
r	.5331	-.0222	.0408∓j.0794	.1163∓j.1253	-.0045	.9864
ϕ	-.2407	-.0373	-.6123±j.2163	-.0082±j.0085	-.0167	-.0303
θ	-.0311	-.3222	-.0475±j.0526	.6438∓j.2153	.0389	-.0192
$\bar{J}_\lambda$=4(.0387)=.151		$\bar{J}_{vec}$=1.0307+.5(.9828)=1.522			$\bar{J}$=1.673	

$$K_{nd} = \begin{matrix} -.1522 & .1385 & -.4831 & -.0557 & .1592 & .1906 & .0350 & .2004 \\ .7082 & -.3502 & .0307 & -.0575 & -.7115 & -.0775 & -.2454 & -1.0428 \\ .3232 & .0986 & .1158 & .0836 & -.0688 & .1567 & .4161 & -.2677 \\ -.3853 & -.3815 & -.3324 & .1769 & -.1402 & -.6619 & -.1723 & .3178 \end{matrix}$$

Note: $\bar{J}_\lambda$ represents the eigenvalue contribution to performance index $\bar{J}$ and $\bar{J}_{vec}$ represents the eigenvector contribution to $\bar{J}$.

second variation improved the overall eigenstructure assignment. The achievable poles in Table 14 are slightly closer to desired than those resulting from the first variation. The eigenvectors associated with heave and side velocity were also slightly closer to desired for the second variation. The penalty paid for improving the majority of the eigenstructure was a degradation in the roll mode forward velocity mode eigenvectors. The complex conjugate forward velocity eigenvectors moved slightly further away from desired than in case of the first weighting variation. The degradation in the roll mode eigenvector was expected, because it was weighted more lightly than the other

eigenvectors. The magnitude of the yaw rate component of this eigenvector increased over the case of the first weighting variation.

Figure 6 shows a plot of the minimum singular values for the unity weighting case (Table 13) as well as the two weighting variation cases (Tables 14 and 15). The singular values in Figure 4 indicate that the closed loop system of the unity weighting case possesses the best stability robustness. However, minimum singular values for the two other cases are also close to one so these closed loop systems also possess good independent stability margins. Table 16 shows the independent stability margins for each of the three cases.

TABLE 16
Independent Stability Margins for Three Weighting Cases
Helicopter Example

	Unity Weighting	First Weighting Variation	Second Weighting Variation
$\alpha = \min \underline{\sigma}[I+G]$	.992	.971	.982
IGM	(.502,125.0)	(.507,34.8)	(.505,54.1)
IPM (°)	(-59.5,59.5)	(-58.1,58.1)	(-58.8,58.8)

<u>Comparison to Garrard and Liebst Results</u> - All three of the design cases presented using the robust eigenstructure assignment algorithm yield independent stability margins superior to those of the Garrard and Liebst results in Table 12. However, the algorithm was not able to yield the

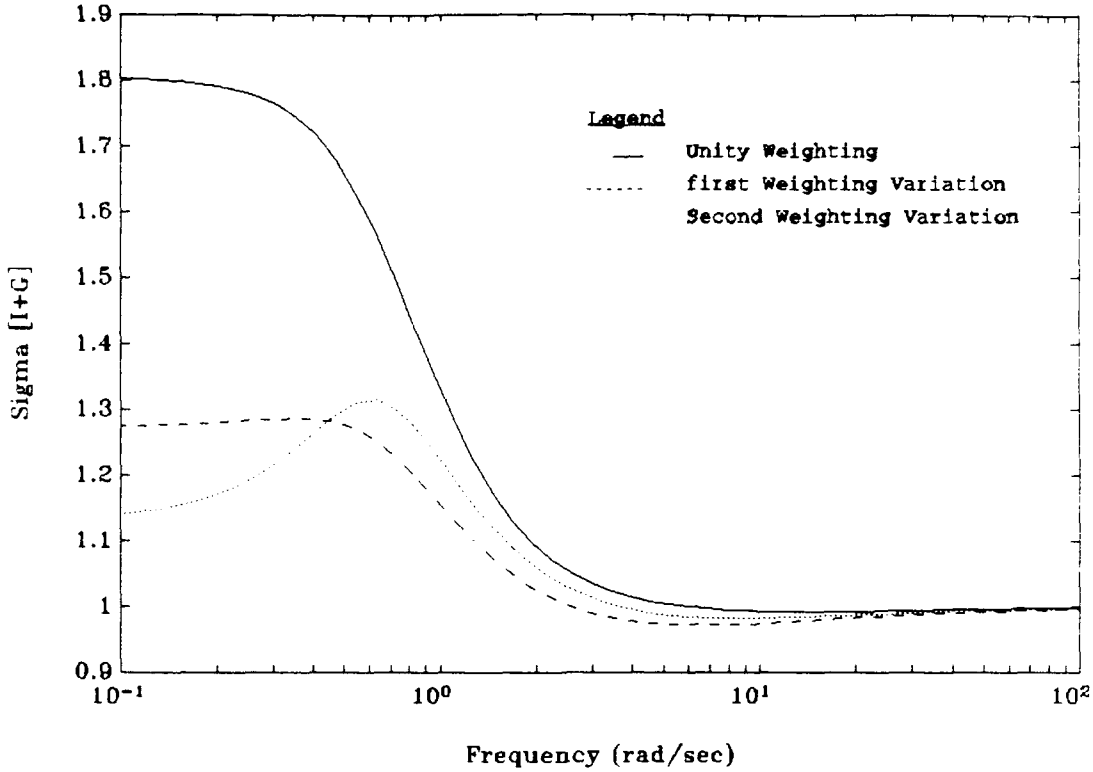

Figure 6 Helicopter Example Minimum Singular Values for Three Weighting Cases

same eigenvector decoupling achieved by Garrard and Liebst. Figure 7 shows a minimum singular value plot for the return difference matrices of the Garrard and Liebst full state design and the design resulting from the second weighting variation (Table 15). The curves in Figure 6 show that the robust eigenstructure assignment algorithm design possesses much better independent stability margins than the Garrard and Liebst design. Recall that for the Garrard and Liebst design IGM=(.599,3.03) and IPM=(-39.1,39.1) while the margins for the algorithm design (from Table 16) are IGM=(.505,54.1) and IPM=(-58.8,58.8).

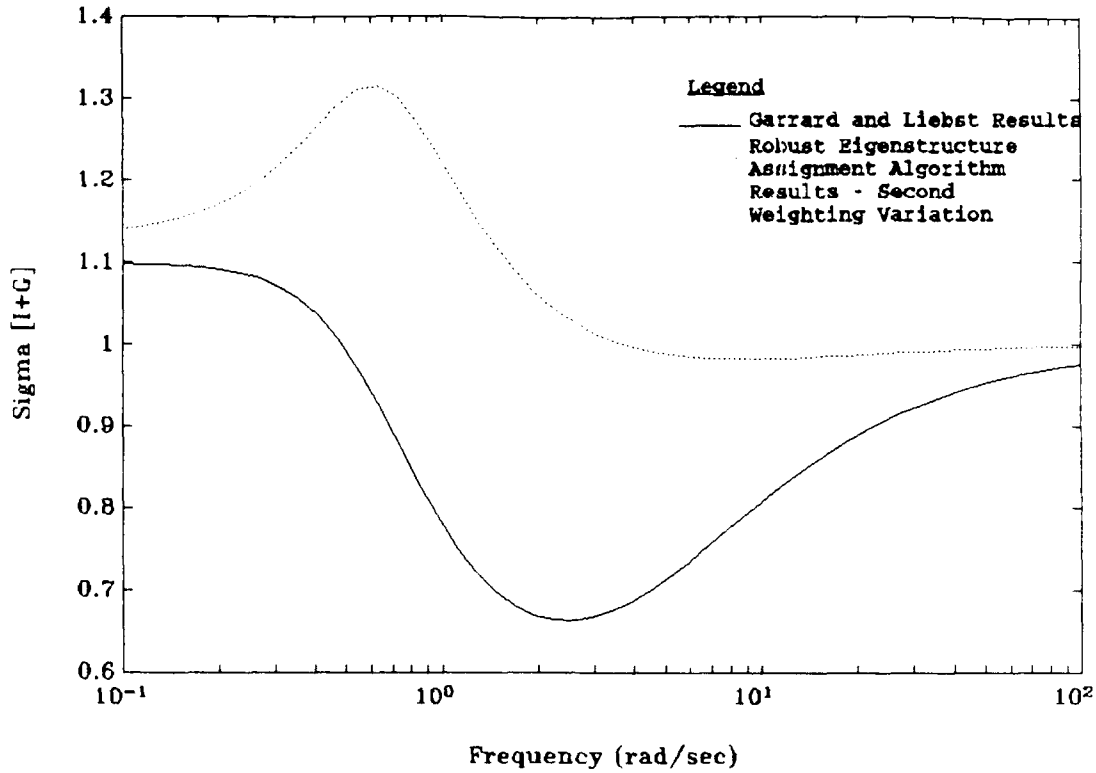

Figure 7 Comparison of Minimum Singular Values of Return Difference Matrix for Garrard and Liebst Design and for Robust Eigenstructure Assignment Algorithm Design.

The algorithm was not able to reproduce the eigenvector decoupling achieved by Garrard and Liebst due to the restrictions of the LQR Because the desired eigenstructure was not entirely with e achievable region, the designs resulting from the use of the algorithm required tradeoffs between the various elements of the eigenstructure. The algorithm was able to nearly achieve the desired closed loop poles and much of the desired decoupling in the eigenvectors. However, the eigenvector associated with the roll mode showed a significant yaw rate component that was

not present in the Garrard and Liebst results. The use of further variations in the weights could result in a roll mode eigenvector closer to desired, but this would be accomplished at the expense of other elements of the eigenstructure. The poles and/or the other eigenvectors would be pulled further from desired.

VI. Conclusions and Recommendations

The algorithm developed in this thesis provides a useful tool to robustly assign the closed loop eigenstructure for full state feedback systems within the constraints of the LQR. The algorithm achieves this eigenstructure assignment by minimizing a quadratic cost function involving the difference between the desired and achievable eigenstructures. One feature of the algorithm is the ability to assign relative weightings to each element of the desired eigenstructure to specify the importance of achieving particular elements. Another feature of the algorithm is that the LQR control weighting matrix, **R**, can be selected as ρI, diagonal, or symmetric positive definite. Specifying $\mathbf{R}=\rho I$ will guarantee good independent gain and phase margins, while specifying $\mathbf{R}>0$ provides the maximum flexibility in achieving the desired eigenstructure. These features provide the user with a great deal of flexibility in achieving a closed loop eigenstructure close to desired. Two control system examples were used to demonstrate the use of the algorithm.

A six state, 2 input F-4 inner loop example served to verify the pole placement portion of the algorithm through comparison to results obtained by Robinson. The FORTRAN code developed for this thesis reduced the required computational time over the MATLAB routine developed by

Robinson by a factor of approximately ten. This example also demonstrated the complete eigenstructure assignment capability of the algorithm to be superior to a method presented by Harvey and Stein. The algorithm developed in this thesis placed the closed loop eigenstructure much closer to desired than the Harvey and Stein method and resulted in significantly better independent stability margins.

An eight state, four input helicopter control system example was also used to demonstrate the algorithm. Garrard and Liebst used this example to demonstrate their eigenstructure assignment technique. The Garrard and Liebst technique, which provides no stability robustness guarantees, placed the poles at the exact desired locations, and then assigned the eigenvectors. The desired eigenstructure for this example was not achievable within the LQR restrictions, thus producing the requirement to trade off between the various desired eigenstructure elements. The algorithm was used to design a system that provided an eigenstructure close to desired but with improved stability robustness over the Garrard and Liebst design. The resulting system did have better stability robustness and placed the majority of the eigenstructure close to desired. However, the required tradeoff resulted in an undesired and significant yaw rate component in the eigenvector associated with the roll mode.

The development of this algorithm provides several opportunities for further research. While the FORTRAN code developed for this thesis made significant speed improvements over the MATLAB based routine, further improvements are possible. The source code in Appendix B can be made more efficient to provide speedier operation, or an even faster minimization routine can be used with the algorithm. A second area to be investigated is the inclusion of the control/state cross coupling matrix in the LQR performance index. The LQR performance index can be rewritten as follows

$$J=\int_0^\infty \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} Q & N \\ N^T & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} dt$$

where **N** is a weighting matrix defining the cross coupling between the states and controls. This matrix can be included in the algorithm, thus providing even greater flexibility in achieving closed loop eigenstructures. One final area for further research is the addition of a state estimator. Realistically, full state feedback is seldom (if ever) available in actual systems, so a state estimator could be implemented with the loop transfer recovery technique to regain the LQR stability margins. The validity of this method for aircraft control system design should then eventually be evaluated through flight test.

Appendix A
Subroutine Descriptions

This appendix provides detailed information on each FORTRAN subroutine written by the author. Included for each routine are a short description, required inputs, available outputs, and calls made to other routines.

SUBROUTINE SORT

This subroutine sorts eigenvalues in ascending order then puts columns of the modal matrix in the same order. The eigenvalue weighting vector and the columns of the eigenvector weighting matrix are also reordered. The calling statement is

```
call SORT(N,edg,ed,maged,Fes,Fe,WDESS,WDES,Fvs,Fv)
```

Inputs

N - dimension of system **A** matrix
edg - N dimensional vector of unsorted desired eigenvalues
Fes - N dimensional vector of unsorted eigenvalue weightings
WDESS - N x N dimensional matrix of unsorted eigenvectors
Fvs - N x N dimensional matrix of unsorted eigenvector weightings

Outputs

ed - N dimensional sorted eigenvalue vector
maged - N dimensional eigenvalue magnitude vector
Fe - N dimensional sorted eigenvalue weighting vector
WDES - N x N dimensional sorted modal matrix
Fv - N x N dimensional sorted eigenvector weighting matrix

Calls

DSVRGP (IMSL sorting routine [19])

SUBROUTINE EAFUNC

This subroutine calculates the value of the performance index $\bar{J}$ and the closed loop eigenstructure for given **Q** and R matrices. Form of call statement is

call EAFUNC(nx,X,RJ)

Inputs

nx = size of X vector

X = vector of the upper triangular portions of **H** and **M**

Outputs

RJ = value of performance index $\bar{J}$

Inputs Passed to EAFUNC Through Common Block

Open loop **A** and **B** matrices
n = dimension of **A**
m = dimension of **B**
ed = nx1 desired eigenvalue vector (complex)
Fe = nx1 weighting vector for eigenvalues
WDES = nxn modal matrix
Fv = nxn matrix whose columns weight corresponding columns of WDES
iwrite = code automatically set in main program designating when to write to output file
nrcode = code specifying what type of **R** matrix to use
1 - **R**=θI
2 - **R**=[diagonal]
other - **R**>0

Outputs Passed to Main Program Through Common Block

ea = nx1 vector of achievable eigenvalues in ascending order
G = mxn LQR optimal gain matrix
ACL = nxn closed loop **A** matrix
WACH = nxn achievable modal matrix
alpha = nx1 vector of eigenvector difference minimization parameter

Calls

MAKEQR, REG, EIGVV, WNORM, SORT, IMINS

SUBROUTINE FMINS

This subroutine finds the **H** and **M** matrices that minimize the performance index calculated in subroutine EAFUNC. The Nelder-Mead simplex algorithm is used to perform the minimization. The calling statement is

```
call FMINS(NX,XGUESS,X,tol,NXP1,v,Fvec,vs,vss,ievalmax)
```

Inputs

NX - dimension of the X vector
XGUESS - NX dimensional initial guess vector containing the upper triangular portions of the **H** and **M** matrices
tol - user defined convergence tolerance (default=.001)
NXP1 - NX + 1
ievalmax - maximum number of simplex iterations

Outputs

X - NX dimensional vector with upper triangular portions of **H** and **M** yielding the lowest values of $\bar{J}$
v - NX x NXP1 matrix whose columns contain a simplex of guess vectors
Fvec - NXP1 dimensional vector of $\bar{J}$ values for each column of simplex v
vs - NX dimensional scratch vector
vss - NX x NXP1 dimensional scratch matrix

Calls

EAFUNC, DSVRGP (IMSL sorting routine [19]), FMINSTEP

SUBROUTINE FMINSTEP

This subroutine calculate each subsequent simplex iteration in the minimization effort. It is part of the Nelder-Mead simplex algorithm. The calling statement is

```
call FMINSTEP(v,NX,NXP1,Fvec,vr,vk,ve,vt,vs,vss,vc,vbar)
```

Inputs

v - NX x NXP1 simplex whose columns are guess vectors
NX - dimension of X vector
NXP1 - NX + 1
Fvec - NXP1 dimensional vector of $\bar{J}$ values for each column of simplex v

Outputs

v - NX x NXP1 simplex whose columns are guess vectors
Fvec - NXP1 dimensional vector of $\bar{J}$ values for each column of simplex v
vr - NX dimensional vector containing reflected point
vk - NX dimensional scratch vector
ve - NX dimensional vector containing expanded point
vt - NX dimensional scratch vector
vs - NX dimensional scratch vector
vss - NX x NXP1 dimensional scratch matrix
vc - NX dimensional vector containing contracted point
vbar - NX dimensional vector containing average vector

Calls

EAFUNC, DSVRGP (IMSL sorting routine [19])

SUBROUTINE MAKEQR

This subroutine builds the **Q** and **R** matrices from the upper triangular portions of the symmetric matrices **H** and **M**. The calling statement is

```
call MAKEQR(N,M,NX,X,Q,R,RM,QH,nrcode)
```

Inputs

N - dimension of **Q** and **H** matrices
M - dimension of **R** and **M** matrices
NX - dimension of X vector
X - NX dimensional vector of the upper triangular portions of matrices **H** and **M**
nrcode - interger designating what type of **R** matrix to use
  1 - $\mathbf{R}=\rho\mathbf{I}$
  2 - **R**=[diagonal]
  3 - **R**>0

Outputs

Q - N x N positive semi-definite LQR state weighting matrix
R - M x M positive definite LQR control weighting matrix
RM - M x M symmetric **M** matrix where $\mathbf{R}=\mathbf{M}^T\mathbf{M}$
QH - N x N symmetric **H** matrix where $\mathbf{Q}=\mathbf{H}^T\mathbf{H}$

Calls

MMUL (LQGLIB routine)

SUBROUTINE WNORM

This subroutine normalizes the eigenvectors to one. The calling statement is

```
call WNORM(WVEC,N)
```

Inputs

WVEC - N dimensional eigenvector
N - dimension of eigenvector

Output

WVEC - N dimensional normalized eigenvector

Calls

none

Appendix B

Subroutine Source Codes

This appendix contains the source code listings for all of the FORTRAN subroutines written by the author. Also included is a listing of the m-file that provides their interface between MATLAB and the FORTRAN routines.

```
      PROGRAM EIGSPACE
      IMPLICIT COMPLEX*16 C
      IMPLICIT REAL*8 (A-B,D-H,O-Z)
      COMMON /INOU/KIN,KOUT
      COMMON A,B,ed,ea,G,NR,NA,ND,M,N,NN,ACL,Fv,Fe,
     + WDES,WACH,calpha,iwrite,nrcode
      DIMENSION X(65),A(10,10),B(10,10),R(10,10),Q(10,10),
     1 RK(10,10),G(10,10),ACL(10,10),Fv(10,10),Fe(10)
      DIMENSION XGUESS(65),XS(65),GRAD(65),
     1 X2(65)
      DIMENSION v(65,66),Fvec(66),vs(65),vss(65,66),
     + Fvec1(66)
      DIMENSION FeS(10),FvS(10,10)
      DIMENSION edr(10),edi(10),wdessr(10,10),wdessi(10,10)
      REAL*8 maged(10)
      COMPLEX*16 ea(10),edg(10),WDES(10,10),WACH(10,10)
      COMPLEX*16 ed(10),WDESS(10,10),calpha(10)
      INTEGER IPERMM(66),IPERMD(10)
      EXTERNAL PPFUNC,DMACH,DUMCGF
      open(UNIT=10,FILE='brad.dat',STATUS='old')
      open(UNIT=9,FILE='PPOUT.DAT',STATUS='old')
      rewind 10
      rewind 9
      KIN=5
      KOUT=6
C    -------------------------------------------------------
C       - set integers specifying array sizes
c         and assign values to scaling vector for X
c    -------------------------------------------------------
      iwrite=0
      read(10,*) N
      NA=N
      NN=2*N
      NA2=N*N
      ND=NN*(4*N+3)
c-------------------------------------------------------------
c        - read values for A matrix - file should have a
c          list of values starting with the dimension of
c          the A matrix, followed by the values of A listed
c          by column (i.e. column 1 followed by column 2 ...)
c          -- next is the column dimension of the B matrix
c             followed by the values for B listed by column
c-------------------------------------------------------------
      jj=1
      icount=0
      do 10 i=1,NA2
        icount=icount+1
        if(icount.eq.11) then
          jj=jj+1
          icount=1
        endif
      read(10,*) A(icount,jj)
```

```
   10 continue
c    ------------------------------------------------------------
c       - read the values for the B matrix
c    ------------------------------------------------------------
      read(10,*) M
      NR=M
      NB=N*M
      jj=1
      icount=0
      do 20 i=1,NB
        icount=icount+1
        if(icount.eq.11) then
          icount=1
          jj=jj+1
        endif
        read(10,*) B(icount,jj)
   20 continue
c    ------------------------------------------------------------
c        - read the desired eigen values
c    ------------------------------------------------------------
      do 30 i=1,N
        read(10,*) FeS(i),edr(i),edi(i)
        edg(i)=DCMPLX(edr(i),edi(i))
   30 continue
c    ------------------------------------------------------------
c        - read the desired eigenvectors and the
c          associated weighting
c    ------------------------------------------------------------
      jj=1
      icount=0
      do 35 i=1,NA2
        icount=icount+1
        if(icount.eq.11) then
          icount=1
          jj=jj+1
        endif
        read (10,*) FvS(icount,jj),wdessr(icount,jj),
     +              wdessi(icount,jj)
        WDESS(icount,jj)=DCMPLX(wdessr(icount,jj),
     +    wdessi(icount,jj))
   35 continue
c    -------------------------------------------------------------
c       - read convergence tolerence, maximum number of
c         evaluations and code specifying type of R
c    -------------------------------------------------------------
      read(10,*) tol
      read(10,*) ievalmax
      read(10,*) nrcode
      if(tol.lt.0.0d0) tol=.001d0
c    -------------------------------------------------------------
c       - sort desired eigenvalues and eigenvectors in order
c         of ascending eigenvalue modulas and normalize
```

```
c         eigenvectors to one
c     ------------------------------------------------------------
      call SORT(N,edg,ed,maged,IPERMD,FeS,Fe,WDESS,
     1 WDES,FvS,Fv)
      call WNORM(WDES,N)
c     ------------------------------------------------------------
c       - set initial guess for R and Q.  The values in
c         XGUESS are the upper triangular portions of matrices
c         M and H.  Q and R are calculated in subroutne
c         MAKEQR.
c     ------------------------------------------------------------
      ix=0
      if(nrcode.eq.1) then
        XGUESS(1)=1.0d0
        ix=1
        goto 51
      endif
      if(nrcode.eq.2) then
        do 41 i=1,M
          ix=ix+1
          XGUESS(ix)=1.0d0
   41   continue
      else
        icount=0
        do 50 i=1,M
          icount=icount+1
          do 40 jj=icount,M
            ix=ix+1
            XS(ix)=1.0d0
            X(ix)=0.0d0
            if(icount.eq.jj)then
              XGUESS(ix)=1.0d0
            else
              XGUESS(ix)=0.0d0
            endif
c            write (9,*) XGUESS(ix)
   40     continue
   50   continue
      endif
   51 continue
c
      icount=0
      do 70 i=1,N
        icount=icount+1
        do 60 jj=icount,N
          ix=ix+1
          XS(ix)=1.0d0
          X(ix)=0.0d0
          if(icount.eq.jj)then
            XGUESS(ix)=1.0d0
          else
            XGUESS(ix)=0.0d0
```

```
          endif
   60   continue
   70 continue
      do 80 i=1,ix
        XS(ix)=1.0d0
   80 continue
      ixp1=ix+1
c     ------------------------------------------------------------
c       - initialize X,Fvec,Fvec1,vs and vss
c     ------------------------------------------------------------
      do 91 i=1,ix
        X(i)=0.0d0
        vs(i)=0.0d0
   91 continue
      do 93 i=1,ixp1
        Fvec1(i)=0.0d0
        do 92 jj=1,ix
          vss(jj,i)=0.0d0
   92   continue
   93 continue
c     ------------------------------------------------------------
c       - Call FMINS for first time
c     ------------------------------------------------------------
      CALL FMINS(ix,XGUESS,X,tol,ixp1,v,Fvec1,vs,vss,
     + IPERMM,ievalmax)
c     ------------------------------------------------------------
c       - Reset XGUESS to X returned from FMINS
c     ------------------------------------------------------------
      kcount=0
      do 100 i=1,ix
        XGUESS(i)=X(i)
  100 continue
c
  110 continue
      kcount=kcount+1
      do 120 i=1,ix
        X2(i)=0.0d0
  120 continue
      do 121 i=1,ix
        X2(i)=0.0d0
        vs(i)=0.0d0
  121 continue
      do 123 i=1,ixp1
        Fvec(i)=0.0d0
        do 122 jj=1,ix
          vss(jj,i)=0.0d0
  122   continue
  123 continue
c     ------------------------------------------------------------
c       - Make next call to FMINS
c     ------------------------------------------------------------
      call FMINS(ix,XGUESS,X2,tol,ixp1,v,Fvec,vs,
```

```
     + vss,IPERMM,ievalmax)
c    ------------------------------------------------------------
c       - Reset XGUESS to X2 returned from FMINS
c    ------------------------------------------------------------
      do 130 i=1,ix
        XGUESS(i)=X2(i)
  130 continue
c    ------------------------------------------------------------
c       - Determine if another itterations is required
c    ------------------------------------------------------------
      delJ=dabs(Fvec1(1)-Fvec(1))
      if (Fvec1(1).lt.Fvec(1)) goto 141
      if (Fvec1(1).gt.Fvec(1)) then
        Fvec1(1)=Fvec(1)
        do 140 i=1,ix
          X(i)=X2(i)
  140   continue
      endif
      if (delJ.gt.tol.and.kcount.lt.50) goto 110
  141 continue
c    ------------------------------------------------------------
c       - Make final call to EAFUNC to write output file
c    ------------------------------------------------------------
      iwrite=1
      CALL EAFUNC(ix,X,RJ)
      end
```

```
      SUBROUTINE SORT(N,edg,ed,maged,IPERMD,FeS,Fe,WDESS,
     1 WDES,FvS,Fv)
      IMPLICIT COMPLEX*16 C
      IMPLICIT REAL*8 (A-B,D-H,O-Z)
      DIMENSION FeS(N),Fe(N),FvS(N,N),Fv(N,N)
      REAL*8 maged(N)
      INTEGER IPERMD(N)
      COMPLEX*16 edg(N),ed(N),WDESS(N,N),WDES(N,N)
c     ------------------------------------------------------------
c       - sort the desired eigenvalues in ascending order then
c         put the weighting matrices Fe and Fv and eigenvector
c         matrix WDES in the same order
c     ------------------------------------------------------------
      do 36 i=1,N
       maged(i)=dsqrt((dreal(edg(i)))**2+(dimag(edg(i)))**2)
       IPERMD(i)=i
   36 continue
      call DSVRGP(N,maged,maged,IPERMD)
      do 37 i=1,N
       ed(i)=edg(IPERMD(i))
       Fe(i)=FeS(IPERMD(i))
       do 38 jj=1,N
        WDES(jj,i)=WDESS(jj,IPERMD(i))
        Fv(jj,i)=FvS(jj,IPERMD(i))
   38  continue
   37 continue
      return
      end
```

```
      SUBROUTINE FMINS(NX,XGUESS,X,TOL,NXP1,v,Fvec,vs,
     + vss,IPERMM,ievalmax)
      IMPLICIT COMPLEX*16 C
      IMPLICIT REAL*8 (A-B,D-H,O-Z)
      DIMENSION XGUESS(NX),X(NX),v(NX,NXP1),Fvec(NXP1),
     + vs(NX),vss(NX,NXP1),vr(65),vk(65),ve(65),
     + vt(65),vc(65),vbar(65)
      INTEGER IPERMM(NXP1)
c
      icallf=0
      icount=0
c     ------------------------------------------------------------
c       - Build initial simplex near XGUESS
c          v(i,j)=simplex matrix
c          vs(i)=scratch vector
c          Fvec(i)=function values corresponding to v(i,j)
c                  columns
c     ------------------------------------------------------------
      xnx=dflotj(NX)
      aa=0.5d0
      p=aa*(dsqrt(xnx+1.0d0)+xnx-1.0d0)/(xnx*dsqrt(2.0d0))
      q=aa*(dsqrt(xnx+1.0d0)-1.0d0)/(xnx*dsqrt(2.0d0))
      do 1010 i=1,NX
        v(i,1)=XGUESS(i)
        vs(i)=v(i,1)
        X(i)=XGUESS(i)
 1010 continue
      icallf=icallf+1
      call eafunc(NX,vs,Fv)
      Fvec(1)=Fv
      i=1
      do 1040 jj=1,NX
        do 1020 kk=1,NX
          vs(kk)=X(kk)
 1020   continue
        i=jj+1
        do 1030 kk=1,NX
          if(jj.eq.kk) then
            v(kk,i)=vs(kk)+p
          else
            v(kk,i)=vs(kk)+q
          endif
          vs(kk)=v(kk,i)
 1030   continue
        icallf=icallf+1
        call EAFUNC(NX,vs,Fv)
        Fvec(i)=Fv
 1040 continue
c     ------------------------------------------------------------
c       - sort the simplex in order of increasing Fvec(i)
c          IPERMM(i)=vector of index of sorted simplex
c                   sort is in ascending order
```

```
c          vsum(i)=summation of abs(v(:,i)
c     ------------------------------------------------------------
      do 1050 i=1,NXP1
        IPERMM(i)=i
 1050 continue
c
      call DSVRGP(NXP1,Fvec,Fvec,IPERMM)
      do 1070 i=1,NXP1
        do 1060 jj=1,NX
          vss(jj,i)=v(jj,i)
 1060   continue
 1070 continue
      do 1090 i=1,NXP1
        do 1080 jj=1,NX
          v(jj,i)=vss(jj,ipermm(i))
 1080   continue
 1090 continue
c     ------------------------------------------------------------
c          iterate until the specified tolerance, tol, is
c          met
c     ------------------------------------------------------------
 1100 continue
      if(icount.gt.ievalmax) goto 1130
      test=0.0d0
      vsum=0.0d0
      do 1120 i=2,NXP1
        do 1110 jj=1,NX
          vsum=dabs(v(jj,i)-v(jj,1))+vsum
 1110   continue
        test=dmax1(test,vsum)
 1120 continue
      if(test.le.tol) go to 1130
c     ------------------------------------------------------------
c        initialize vr,vk,ve,vt,vs,vss,vc,vbar
c     ------------------------------------------------------------
      do 1121 i=1,NX
        vr(i)=0.0d0
        vk(i)=0.0d0
        ve(i)=0.0d0
        vt(i)=0.0d0
        vs(i)=0.0d0
        vc(i)=0.0d0
        vbar(i)=0.0d0
        do 1122 jj=1,NXP1
          vss(i,jj)=0.0d0
 1122   continue
 1121 continue
c     ------------------------------------------------------------
c        - Calculate the next step in the simplex
c     ------------------------------------------------------------
      call FMINSTEP(v,NX,NXP1,Fvec,vr,vk,ve,vt,vs,vss,
     + vc,vbar,IPERMM)
```

```
      icount=icount+1
      goto 1100
1130  continue
      do 1140 i=1,NX
        X(i)=v(i,1)
1140  continue
      return
      end
```

```
      SUBROUTINE FMINSTEP(v,NX,NXP1,Fvec,vr,vk,ve,
     + vt,vs,vss,vc,vbar,IPERMM)
      IMPLICIT COMPLEX*16 C
      IMPLICIT REAL*8 (A-B,D-H,O-Z)
      DIMENSION v(NX,NXP1),Fvec(NXP1),vr(NX),vk(NX),
     + ve(NX),vt(NX),vs(NX),vss(NX,NXP1),vc(NX),
     + vbar(NX)
      INTEGER IPERMM(NXP1)
      icall=0
      alpha=1.0d0
      beta=0.5d0
      gamma=2.0d0
      xnx=dflotj(NX)
c     ------------------------------------------------------------
c       - Calculate average vector
c     ------------------------------------------------------------
      do 2020 i=1,NX
        vb=0.0d0
        do 2010 jj=1,NX
          vb=vb+v(i,jj)
 2010   continue
        vbar(i)=vb/xnx
 2020 continue
c     ------------------------------------------------------------
c       - Calculate reflected point
c     ------------------------------------------------------------
      do 2030 i=1,NX
        vr(i)=vbar(i)+alpha*(vbar(i)-v(i,NXP1))
 2030 continue
      icall=icall+1
      call EAFUNC(NX,vr,fr)
      do 2040 i=1,NX
        vk(i)=vr(i)
 2040 continue
      fk=fr
      if(fr.lt.Fvec(1)) then
c     ------------------------------------------------------------
c       - Calculate expanded point
c     ------------------------------------------------------------
          do 2050 i=1,NX
            ve(i)=vbar(i)+gamma*(vr(i)-vbar(i))
 2050     continue
          icall=icall + 1
          call EAFUNC(NX,ve,fe)
          if(fe.lt.Fvec(1)) then
            do 2060 i=1,NX
              vk(i)=ve(i)
 2060       continue
            fk=fe
          endif
      else
c     ------------------------------------------------------------
```

```
c       - Calculate contracted point
c     ------------------------------------------------------------
        if(fr.ge.Fvec(NXP1)) then
          do 2070 i=1,NX
            vt(i)=v(i,NXP1)
 2070     continue
          ft=Fvec(NXP1)
        else
          do 2080 i=1,NX
            vt(i)=vr(i)
 2080     continue
          ft=fr
        endif
        do 2090 i=1,NX
          vc(i)=vbar(i)-beta*(vbar(i)-vt(i))
 2090   continue
        icall=icall+1
        call EAFUNC(NX,vc,fc)
        if(fc.lt.Fvec(NX)) then
          if(fc.ge.fr) goto 2135
          do 2100 i=1,NX
            vk(i)=vc(i)
 2100     continue
          fk=fc
        else
c     ------------------------------------------------------------
c       - Move point half the distance to the best point
c     ------------------------------------------------------------
          do 2120 i=2,NX
            do 2110 jj=1,NX
              v(jj,i)=(v(jj,1)+v(jj,i))/2.0d0
              vs(jj)=v(jj,i)
 2110       continue
            icall=icall+1
            call EAFUNC(NX,vs,Fv)
            Fvec(i)=Fv
 2120     continue
          do 2130 i=1,NX
            vk(i)=(v(i,1)+v(i,NXP1))/2.0d0
 2130     continue
          icall=icall+1
          call EAFUNC(NX,vk,fk)
 2135   endif
      endif
      do 2140 i=1,NX
        v(i,NXP1)=vk(i)
 2140 continue
      Fvec(NXP1)=fk
      do 2150 iii=1,NXP1
        IPERMM(iii)=iii
 2150 continue
c     ------------------------------------------------------------
```

```
c       - Resort the simplex
c     ------------------------------------------------------------
      call DSVRGP(NXP1,Fvec,Fvec,IPERMM)
      do 2170 i=1,NXP1
        do 2160 jj=1,NX
          vss(jj,i)=v(jj,i)
 2160   continue
 2170 continue
      do 2190 i=1,NXP1
        do 2180 jj=1,NX
          v(jj,i)=vss(jj,IPERMM(i))
 2180   continue
 2190 continue
      return
      end
```

```
      SUBROUTINE EAFUNC(NX,X,RJ)
      IMPLICIT COMPLEX*16 C
      IMPLICIT REAL*8 (A-B,D-H,O-Z)
      COMMON /INOU/KIN,KOUT
      COMMON A,B,ed,ea,G,NR,NA,ND,M,N,NN,ACL,Fv,Fe,
     + WDES,WACH,calpha,iwrite,nrcode
      DIMENSION X(65),A(10,10),B(10,10),R(10,10),Q(10,10),
     1 RK(10,10),G(10,10),ACL(10,10),Fv(10,10),Fe(10),Fes(10)
      DIMENSION DUM(860,1),IDUM(20),WR(10),WI(10),Z(10,10),
     1 IV1(10),FV1(10),ACLS(10,10)
      COMPLEX*16 ea(10),WDES(10,10),WACH(10,10),
     1 WDESS(10,10),WACHS(10,10),calpha(10)
      COMPLEX*16 ed(10),cedif(10),edtmp(10),eatmp(10)
      REAL*8 magea(10)
c     ------------------------------------------------------------
c        -this subroutine calls the cost function subroutine,
c         allowing variable arrays to be set
c     ------------------------------------------------------------
      RJ=0.0d0
      do 176 i=1,10
        ea(i)=DCMPLX(0.0d0,0.0d0)
        do 177 jj=1,10
          ACL(i,jj)=0.0d0
          WACH(i,jj)=dcmplx(0.0d0,0.0d0)
          G(i,jj)=0.0d0
  177   continue
  176 continue
c
      call EA(NX,X,RJ,NR,NA,ND,N,M,NN,A,B,R,Q,RK,G,ACL,Fv,
     1 Fe,Fes,DUM,IDUM,WR,WI,Z,IV1,FV1,ea,ed,WDES,WACH,cedif,
     2 edtmp,eatmp,magea,ACLS,WDESS,WACHS,calpha,iwrite,
     3 nrcode)
c
      RETURN
      END
C-----------------------------------------------------------------
C-----------------------------------------------------------------
C-----------------------------------------------------------------
      SUBROUTINE EA(NX,X,RJ,NR,NA,ND,N,M,NN,A,B,R,Q,RK,G,ACL,Fv,
     1 Fe,Fes,DUM,IDUM,WR,WI,Z,IV1,FV1,ea,ed,WDES,WACH,cedif,
     2 edtmp,eatmp,magea,ACLS,WDESS,WACHS,calpha,iwrite,nrcode)
      IMPLICIT COMPLEX*16 C
      IMPLICIT REAL*8 (A-B,D-H,O-Z)
      COMMON /INOU/KIN,KOUT
      DIMENSION X(NX),A(N,N),B(N,M),R(M,M),Q(N,N),
     1 RK(N,N),G(M,N),ACL(N,N),Fv(N,N),Fe(N),WNORMA(10),
     2 FvS(10,10),FeS(N)
      DIMENSION DUM(ND,1),IDUM(NN),WR(N),WI(N),Z(N,N),IV1(N),
     1 FV1(N)
      DIMENSION RM(10,10),QH(10,10)
      DIMENSION edifmag(10),ACLS(N,N)
      DIMENSION FvDUM(10,10),FeDUM(10,10)
```

```
      COMPLEX*16 ea(N),ed(N),WDES(N,N),WACH(N,N),WDESS(N,N),
     1 WACHS(N,N),calpha(N)
      COMPLEX*16 cedif(N),edtmp(N),eatmp(N)
      INTEGER IPERMA(10),imin(10)
      REAL*8 magea(10)
      LOGICAL ELIM
c     -------------------------------------
c       set required constants
c     -------------------------------------
      IPRT=0
c
      if(iwrite.ne.0) then
        write(9,*) N
        write(9,*) M
      endif
c     ---------------------------------------------------------
c         - calculate the Q and R matrices
c     ---------------------------------------------------------
      CALL MAKEQR(N,M,NX,X,Q,R,RM,QH,nrcode)
      if(iwrite.ne.0) then
        do 557 i=1,M
          do 556 jj=1,M
            write (9,*) R(jj,i)
  556       continue
  557     continue
        do 555 i=1,N
          do 554 jj=1,N
            write(9,*) Q(jj,i)
  554       continue
  555     continue
      endif
c     ---------------------------------------------------------
c          calculate the lqr gain matrix, G
c     ---------------------------------------------------------
      CALL REG(NA,NR,N,M,NN,A,B,Q,R,RK,G,ACL,DUM,IDUM,IPRT)
      do 528 i=1,N
        do 529 jj=1,N
          ACLS(i,jj)=ACL(i,jj)
  529     continue
  528 continue
c     ---------------------------------------------------------
c       - calculate the new closed loop eigenvalues
c     ---------------------------------------------------------
      ipc=1
      CALL EIGVV(NA,N,ACLS,WR,WI,Z,IV1,FV1,IPC,IERR)
c     ---------------------------------------------------------
c       -  put the achievable eigenvectors into matrix WACHS
c     ---------------------------------------------------------
      if(iwrite.ne.0) then
        do 530 i=1,N
          do 5555 jj=1,N
            write (9,*) RK(jj,i)
```

```
 5555     continue
  530   continue
        do 531 i=1,N
          do 532 jj=1,M
            write(9,*) G(jj,i)
  532     continue
  531   continue
      endif
c
      icomplex=0
      do 539 i=1,N
        if(WI(i).ne.0.0d0) icomplex=icomplex+1
  539 continue
      NCMP=N-icomplex/2
      ii=0
      do 540 i=1,NCMP
        ii=ii+1
        if(dabs(WI(ii)).gt.0.0) then
          do 535 jj=1,N
            WACHS(jj,ii)=DCMPLX(Z(jj,ii),Z(jj,ii+1))
            WACHS(jj,ii+1)=DCONJG(WACHS(jj,ii))
  535     continue
          ii=ii+1
        else
          do 536 jj=1,N
            WACHS(jj,ii)=DCMPLX(Z(jj,ii),0.0d0)
  536     continue
        endif
  540 continue
c     ------------------------------------------------------------
c       - normalize the eigenvectors
c     ------------------------------------------------------------
      call WNORM(WACHS,N)
c     ------------------------------------------------------------
c       - find the poles that are closest to each other and
c         take their difference
c         -- calculate the magnitude of the poles and sort in
c            ascending order
c     ------------------------------------------------------------
      do 30 i=1,N
       eatmp(i)=dcmplx(WR(i),WI(i))
   30 continue
c     ------------------------------------------------------------
c        -- sort achievable eigenvalues in ascending order
c           and put eigenvectors in same order
c     ------------------------------------------------------------
      call SORT(N,eatmp,ea,magea,IPERMA,FeDUM,FeDUM,
     +          WACHS,WACH,FvDUM,FvDUM)
c     ------------------------------------------------------------
c         -- put eigenvalues, eigenvectors, and weightings
c            into scratch arrays for calculations
c     ------------------------------------------------------------
```

```
   40 continue
      if(iwrite.ne.0) then
        do 43 i=1,N
          eareal=dreal(ea(i))
          eaimag=dimag(ea(i))
          write(9,*) eareal
          write(9,*) eaimag
   43   continue
      endif
      do 41 i=1,N
        eatmp(i)=ea(i)
        edtmp(i)=ed(i)
        FeS(i)=Fe(i)
        do 42 jj=1,N
          WACHS(jj,i)=WACH(jj,i)
          WDESS(jj,i)=WDES(jj,i)
          FvS(jj,i)=Fv(jj,i)
          if(iwrite.ne.0) then
            wreal=dreal(wach(jj,i))
            wimag=dimag(wach(jj,i))
            write(9,*) wreal
            write(9,*) wimag
          endif
   42 continue
   41 continue
c     ------------------------------------------------------------
c          -- find the acheivable poles closest to desired
c             and calculate the difference
c     ------------------------------------------------------------
      jjj=0
      do 501 jj=1,N
        do 502 kk=1,N
          if(Fv(jj,kk).ne.0.0d0) jjj=jjj+1
  502   continue
  501 continue
      NL=N
      RJ=0.0d0
      do 50 i=1,N
c     ------------------------------------------------------------
c          -- calculate difference between desired poles
c             and each remaining acheivable pole
c     ------------------------------------------------------------
        do 51 jj=1,NL
          cedif(jj)=edtmp(1)-eatmp(jj)
   51   continue
c     ------------------------------------------------------------
c          -- find the minimum difference for the current
c             desired pole
c     ------------------------------------------------------------
        do 56 jj=1,NL
          edifmag(jj)=dsqrt(dreal(cedif(jj))**2+
     +                  dimag(cedif(jj))**2)
```

```
   56   continue
        call imins(NL,edifmag,imin(i))
        iii=IMIN(i)
c
        RJVEC=0.0d0
        if(jjj.eq.0) then
          if(iwrite.ne.0) then
            write(9,*) jjj
            write(9,*) jjj
            goto 503
          endif
        endif
c     ----------------------------------------------------
c        - calculate the calpha for the eigenvectors
c     ----------------------------------------------------
        sum1=0.0d0
        sum2=0.0d0
        if(dimag(eatmp(iii)).ne.0.0d0) then
          do 584 jj=1,N
            sum1=sum1+dimag(WDESS(jj,1))*dreal(WACHS(jj,iii))
     +          -dreal(WDESS(jj,1))*dimag(WACHS(jj,iii))
            sum2=sum2+dreal(WDESS(jj,1))*dreal(WACHS(jj,iii))
     +          +dimag(WDESS(jj,1))*dimag(WACHS(jj,iii))
  584     continue
          ph1=datan2(sum1,sum2)
          calpha1=dcmplx(dcos(ph1),dsin(ph1))
          calpha2=-1*calpha1
        else
          calpha1=(1.0d0,0.0d0)
          calpha2=-1*calpha1
        endif
c     ------------------------------------------------------------
c        determine which calpha produces minimum cost
c     ------------------------------------------------------------
        DELWI1=0.0d0
        DELWI2=0.0d0
        do 585 jj=1,N
          DELWI1=FvS(jj,1)*((DREAL(WDESS(jj,1)-
     +           calpha1*WACHS(jj,iii)))**2
     +          +(DIMAG(WDESS(jj,1)-calpha1*WACHS(jj,iii)))**2)
     +          +DELWI1
          DELWI2=FvS(jj,1)*((DREAL(WDESS(jj,1)-
     +           calpha2*WACHS(jj,iii)))**2
     +          +(DIMAG(WDESS(jj,1)-calpha2*WACHS(jj,iii)))**2)
     +          +DELWI2
  585   continue
        if(DELWI1.lt.DELWI2) then
          DELWI=DELWI1
          calphai=calpha1
        else
          DELWI=DELWI2
          calphai=calpha2
```

```
        endif
        RJVEC=DELWI
        if (iwrite.ne.0) then
          do 587 jj=1,N
            if(eatmp(iii).eq.ea(jj)) calpha(jj)=calphai
  587     continue
          if(i.eq.N)then
            do 586 jj=1,N
              alreal=dreal(calpha(jj))
              alimag=dimag(calpha(jj))
              write(9,*) alreal
              write(9,*) alimag
  586       continue
          endif
        endif
  503   continue
c     ------------------------------------------------------------
c         -- add this pole's contribution to J
c     ------------------------------------------------------------
       RJ=RJ+FeS(1)*edifmag(iii)**2+RJVEC
c     ------------------------------------------------------------
c         -- reset the eigenvalue/vector arrays to eliminate
c            those poles already matched up.  Program will
c            reset the achievable values by
c            skipping the set that had the minimum eigenvalue
c            difference modulas.
c            The desired values are reset by
c            eliminating the first set (this is the eigenvalue
c            that was used to calculate cedif(i).
c     ------------------------------------------------------------
        k=0
        NL=NL-1
        do 52 jj=1,NL
          k=k+1
          ELIM=jj.eq.iii
          IF(ELIM) K=k+1
          eatmp(jj)=eatmp(k)
          edtmp(jj)=edtmp(jj+1)
          FeS(jj)=FeS(jj+1)
          if(jjj.eq.0) goto 591
          do 590 kk=1,N
            WACHS(kk,jj)=WACHS(kk,k)
            WDESS(kk,jj)=WDESS(kk,jj+1)
            FvS(kk,jj)=FvS(kk,jj+1)
  590     continue
  591     continue
   52   continue
   50 continue
c
      if (iwrite.ne.0) write(9,*) RJ
c
      RETURN
```

END

```
      SUBROUTINE IMINS(NL,EDIFMAG,I)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION EDIFMAG(NL)
      i=1
      do 5000 jj=1,nl
        if(dabs(edifmag(jj)).lt.dabs(edifmag(i))) i=jj
 5000 continue
      return
      end
```

```
      SUBROUTINE WNORM(WVEC,N)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION WNORMV(10)
      COMPLEX*16 WVEC(N,N)
      do 5 i=1,N
       WNORMV(i)=0.0d0
    5 continue
      do 10 i=1,N
       do 20 jj=1,N
        WNORMV(i)=WNORMV(i)+(dreal(WVEC(jj,i)))**2+
     +            (dimag(WVEC(jj,i)))**2
   20  continue
   10 continue
      do 30 i=1,N
       do 40 jj=1,N
        WVEC(jj,i)=WVEC(jj,i)/dsqrt(wnormv(i))
   40  continue
   30 continue
      return
      end
```

```
      SUBROUTINE MAKEQR(N,M,NX,X,Q,R,RM,QH,nrcode)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(NX),Q(N,N),R(M,M)
      DIMENSION RM(M,M),QH(N,N)
C    ------------------------------------------------------------
C      - this subroutine takes the upper triangular portion
c        of matrices H and M (input through vector X)
c        and returns the symmetric positive definite
c        Q and R matrices
c          -nrcode=1 then R=rho*I, nrcode=2 then R=diagonal
c          -any other nrcode yields R free to be full
c    ------------------------------------------------------------
      ix=0
      if(nrcode.eq.1.or.nrcode.eq.2) then
        ix=1
        do 116 i=1,M
          do 117 jj=1,M
            if(i.eq.jj) then
              RM(i,jj)=X(ix)
            else
              RM(i,jj)=0.0d0
            endif
  117     continue
          if(nrcode.eq.2) ix=ix+1
  116   continue
      else
        icount=0
        do 101 i=1,M
          icount=icount+1
          do 102 jj=icount,M
            ix=ix+1
            RM(i,jj)=X(ix)
            RM(jj,i)=X(ix)
  102     continue
  101   continue
      endif
      icount=0
      do 111 i=1,N
        icount=icount+1
        do 112 jj=icount,N
        ix=ix+1
        QH(i,jj)=X(ix)
        QH(jj,i)=X(ix)
  112 continue
  111 continue
C    ------------------------------------------------------------
c      - calculate the matrix products QHt*QH and RMt*RM
c        which is the same as HQ*QH and RM*RM because both
c        are symmetric
c    ------------------------------------------------------------
      call MMUL(M,M,M,M,M,M,RM,RM,R)
      call MMUL(N,N,N,N,N,N,QH,QH,Q)
```

```
return
end
```

```
function [Q,R,P,ea,va,θ,Jbar]=LQREA(a,b,ed,Fe,vd,Fv,tol,rcode)
%LQREA    Eigenstructure assignment using the Linear Quadratic
%         Regulator.
%    Form is
%    [Q,R,P,ea,va,theta,Jbar]=LQREA(a,b,ed,Fe,vd,Fv,tol,rcode)
%    Input parameters,
%         a=nxn A matrix,          b=nxm B matrix
%         ed=nxn diagonal matrix of desired eigenvalues
%         Fe=nx1 matrix weighting each eigenvalue
%         vd=nxn matrix whose columns are the desire eigenvectors
%                -must be in same order as associated eigenvalues
%         Fv=nxn matrix whose elements weight corresponding
%                elements of vd
%         tol=convergence tolerance for performance index
%         rcode=determines type of R
%              (1) R=ro*I
%              (2) R=[diag]
%              (other) R=positive definite
%    Output parameters,
%         ea=nxn diagonal matrix of acheivable eigenvalues
%         va=nxn matrix of acheivable eigenvectors
%         θ=nx1 vector of value that minimizes the difference
%                   between desired and acheivable eigenvectors
%         Jbar=final value of performance index
%         P=unique positive definite solution to Riccati equation
%         Q,R=final state and control weighting matrices
[nr,nc]=size(a);
[mr,mc]=size(b);
n=nr;
m=mc;
at=a(:);
bt=b(:);
for i=1:n,
  edd(i,1)=Fe(i);
  edd(i,2)=real(ed(i,i));
  edd(i,3)=imag(ed(i,i));
end
vdd(:,1)=Fv(:);
vdd(:,2)=real(vd(:));
vdd(:,3)=imag(vd(:));
ievalmax=1000;
save brad.dat n at m bt edd vdd tol ievalmax rcode /ascii
!run eigspace
load ppout.dat
count=1;
n=ppout(count);
count=count+1;
m=ppout(count);
count=count+1;
n2=n*n;
m2=m*m;
nm=n*m;
```

```
R=zeros(m,m);
R(:)=ppout(count:count+m2-1);
count=count+m2;
Q=zeros(n,n);
Q(:)=ppout(count:count+n2-1);
count=count+n2;
P=zeros(n,n);
P(:)=ppout(count:count+n2-1);
count=count+n2;
K=zeros(m,n);
K(:)=ppout(count:count+nm-1);
count=count+nm;
ea=zeros(n,n);
for i=1:n
 ea(i,i)=ppout(count)+j*ppout(count+1);
 count=count+2;
end
va=zeros(n,n);
for jj=1:n
 for i=1:n
  va(i,jj)=ppout(count)+j*ppout(count+1);
  count=count+2;
 end
end
theta=zeros(n,n);
for i=1:n
 theta(i,i)=ppout(count)+j*ppout(count+1);
 count=count+2;
end
Jbar=ppout(count);
```

Bibliography

1. Moore, B. C. "On the Flexibility Offered by State Feedback in Multivariable Systems Beyond Closed Loop Eigenvalue Assignment," IEEE Transactions on Automatic Control, Volume AC-21: 689-692 (October 1976).

2. Sobel, Kenneth M., Wangling Yu, and Frederick J. Lallman. "Eigenstructure Assignment with Gain Suppression Using Eigenvalue and Eigenvector Derivatives," Journal of Guidance, Volume 13, No. 6: 1008-1013 (November-December 1990).

3. Garrard, William L. and Bradley S. Liebst. "Design of a Multivariable Helicopter Flight Control System for Handling Qualities Enhancement," Journal of the American Helicopter Society, Volume 35: 23-30 (October 1990).

4. Garrard, William L. and Bradley S. Liebst. "Active Flutter Suppression Using Eigenspace and Linear Quadratic Design Techniques," Journal of Guidance, Volume 8, No. 3: 304-311 (May-June 1985).

5. Costigan, M. and R. Calico. "An Analysis of Lateral-Directional Handling Qualities and Eigenstructure of High Performance Aircraft," AIAA paper 89-0017, 27th Aerospace Sciences Meeting, January 9-12, 1989, Reno Nevada.

6. Anderson, Brian D. O. and John B. Moore. Optimal Control, Linear Quadratic Methods. New Jersey: Prentice Hall Inc., 1990.

7. Innocenti, M. and C. Stanziola. "Performance-Robustness Trade Off of Eigenstructure Assignment Applied to Rotorcraft," Aeronautical Journal: 124-131 (April 1990).

8. Wilson, Robert F. and James R. Cloutier. "Optimal Eigenstructure Achievement with Robustness Guarantees," American Control Conference Proceedings: 952-957 (1989).

9. Broussard, John R. "A Quadratic Weight Selection Algorithm," IEEE Transactions on Automatic Control, Volume AC-27: 945-947 (August 1982).

10. Harvey, Charles A. and Gunter Stein. "Quadratic Weights for Asymptotic Regulator Properties," IEEE Transactions on Automatic Control, Volume AC-23: 378-387 (June 1978).

11. Robinson, Lieutenant Jeffrey D. A Linear Quadratic Regulator Weight Selection Algorithm for Robust Pole Assignment. MS thesis, AFIT/GAE/ENG/90D-23. School Of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990 (AD-A230598).

12. Liebst, Bradley S., William L. Garrard, and William M. Adams. "Design of an Active Flutter Suppression System," Journal of Guidance, Volume 9, No. 1: 64-71 (January-February 1886).

13. Ridgely, Captain D. Brett and Siva s. Banda. Introduction to Robust Multivariable Control. AFWAL-TR-85-3102, Control Dynamics Branch, Flight Dynamics Laboratory, Wright-Patterson AFB OH, February 1986 (AD-A165891).

14. Kwakernaak, Huibert and Raphael Sivan. Linear Optimal Control Systems. New York: John Wiley & Sons, Inc., 1972.

15. Safonov, Michael G. and Michael Athans. "Gain and Phase Margin for Multiloop LQG Regulators," IEEE Transactions on Automatic Control, Volume AC-22: 173-179 (April 1977).

16. Floyd, Captain R. M. LQGLIB User's Manual - A Description of Computer Routines for Use in Linear Systems Studies.

17. Kuester, James L. and Joe H. Mize. Optimization Techniques With Fortran. New York: McGraw-Hill Book Company, 1973.

18. Hoh, Rodger, et al. "Proposed Airworthiness Design Standard: Handling Qualities Requirements for Military Rotocraft," Tech. Report No. 1194-2, System Technology, Inc., Hawthorne, California (December 1985).

19. IMSL User's Manual; Math/Library. IMSL, Inc., Houston, Texas, Volume 1, Chapters 1-2, Version 1.0 (April 1987).

20. Franklin, Gene F. and J. David Powell. Feedback Control of Dynamic Systems, Massachusetts: Addison-Wesley Publishing Company, Inc., 1986.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Master's Thesis

4. TITLE AND SUBTITLE
AN ALGORITHM FOR ROBUST EIGENSTRUCTURE ASSIGNMENT USING THE LINEAR QUADRATIC REGULATOR

5. FUNDING NUMBERS

6. AUTHOR(S)
Thomas C. Huckabone, Captain, USAF

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/91D-7

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

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Linear Quadratic Regulator (LQR) can guarantee a robust closed loop eigenstructure for full state feedback. The algorithm developed here takes advantage of the stability guarantees of LQR to achieve an eigenstructure close to desired but within the allowable region of LQR. The algorithm selects the LQR weighting matrices, **Q** and **R**, that minimize the distance between the elements of the desired and LQR achievable eigenstructures. The minimization is accomplished by using a simplex based optimization routine. Specific weightings placed on the elements of the desired eigenstructure define the relative importance of each element. The algorithm is programed in FORTRAN and is designed to be run from the software package MATLAB. Two examples are examined to illustrate the use of the program, including a helicopter flight control system. The results show that this algorithm is a valid technique for achieving robust eigenstructure assignment with full state feedback.

14. SUBJECT TERMS
Eigenstructure Assignment; Linear Quadratic Regulator; Eigenvalues; Eigenvectors; Control Theory; Linear Systems; Optimization

15. NUMBER OF PAGES
118

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