l^1-Optimal Control: Solution Software and Design Examples

by

David M. Richards

B.S. Chemical Engineering
B.S. Electrical Engineering
Massachusetts Institute of Technology
(1986)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

Master of Science
in
Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology

July 1989

Signature of Author ______________________________
Department of Electrical Engineering and Computer Science
July 21, 1989

Certified by ______________________________
Professor Munther A. Dahleh
Assistant Professor, E.E.C.S. Thesis Supervisor

Accepted by ______________________________
Professor Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

l^1-Optimal Control:

Solution Software and Design Examples

by

David M. Richards

Software is developed to solve the l^1 optimization problem in the context of l^1-optimal control theory. The software is tested, and the space station attitude control/momentum management problem and the X29 forward swept wing aircraft pitch control problem are presented as design examples to evaluate the performance of the software on real problems as well as to determine what parts of the l^1 theory need to be further researched to allow more efficient computation of solutions. Through these examples it is shown that l^1-optimal theory does not always lend itself to simple and straightforward computation of results. Particular problem areas are identified, and reccommendations are made for future research in the calculation of l^1-optimal controllers.

Acknowledgements

I would first like to thank Professor Munther Dahleh for providing me with this opportunity and for the constant encouragement when things were looking rather bleak. I would also like thank the Building 39 crew for their support and career guidance, and the Cambridge gang for the temporary accomodations during those final frantic days. Finally, there is, of course, my family. They were always there to listen to my problems and to share in the joys of success (as well as an occasional meal or two!).

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background and Outline

A recently proposed design methodology for multivariable feedback control systems is l^1-optimal theory [1]. This methodology arises from the problem of designing a stabilizing controller to minimize system output magnitude for arbitrary bounded persistent disturbances. The realization of this objective is the minimization of the l^1 norm of the closed loop system impulse response.

Because of the newness of this methodology, l^1-optimal design has not been applied in real situations, and no software package is available to set up and solve the l^1 optimization. These two things, a convenient means of calculation and concrete design examples, are necessary for a theory to become accepted in practice, and therein lies the motivation for this thesis.

The first objective is to develop software to generate and solve the l^1 optimization problem from the designer's system description. The algorithm and details of implementation are presented in Chapter 2.

The second objective is to use the software to carry out actual l^1 designs. While some simpler examples are used to demonstrate that the code works as intended, two main examples are used. These are the pitch control of a forward swept wing aircraft and the attitude control and momentum management of a space station. The space station example is presented in Chapter 3 and the aircraft example in Chapter 4.

Finally, Chapter 5 presents conclusions and comments. The purpose of this research is not so much to develop the best possible control systems for the example problems, but rather to take a look at the l^1 problem from a practical viewpoint. Specific concerns are the simplicity (or difficulty) of calculation in real problems, the elements of a control problem which are important for l^1 design, and the areas which need further attention.

1.2 The Example Problems

The first example treated in this thesis is the attitude control and momentum management of a space station. This problem arises from using momentum exchange devices such as control moment gyros (CMG's) to perform attitude maneuvers such as counteracting disturbance torques [2,3,4,5]. Reaction torques are produced on the station by altering the momentum of the CMG's. Uncontrolled, the CMG's will eventually saturate (reach some upper limit of accumulated momentum) and desaturation maneuvers are necessary. Desaturation is accomplished by applying external torques to reduce the stored momentum.

A desirable method of controlling the accumulated momentum is to use gravity-gradient torques which result from unequal gravitational forces across the body of the station. In this scheme, a controller will try to drive the station to a torque equilibrium attitude (TEA) in which the disturbance torque is balanced by the gravity-gradient torque.

The major disturbance torque is the aerodynamic torque from atmospheric drag. This is a constant offset with sinusoidal components from density deviations and articulating components of the station such as solar panels or antennas. The constant offset will dictate a constant TEA, but the cyclic components of the disturbance will cause a cyclic attitude profile.

Under these conditions, the job of the controller is to seek the constant TEA while minimizing the cyclic excursions around this constant. Another desirable quality of the controller is robustness since other disturbances may be encountered or the properties of the station may change as people or equipment come and go. Finally, the controller will be designed using a linearized version of a non-linear model. Since the linearization is only valid for small deviations away from the linearization point, it is desired to minimize excursions away from this point. For the space station case, variable values are nominally zero, so the objective is to keep all values as small as possible at all times.

For these reasons, l^1optimal control seems like a good choice for the space station. l^1 controllers are designed to minimize the maximum magnitude of system response to arbitrary l^∞ signals and include some guarantee of robustness. These properties are just what is called for in the space station problem.

The second example is the forward swept wing aircraft example as presented by Quinn [11]. This problem concerns the pitch control of the X29 aircraft using the canard and flaperon control surfaces. The interest in this problem stems from the fact that the X29 will not fly without sophisticated controls as it is unstable. The pitch problem is further complicated by the presence of a non-minimum phase zero which limits the possible performance of any control system.

In the context of this thesis, the X29 problem is used as an example of a more difficult problem (notably the non-minimum phase zero). Also, the X29 has more sharply defined performance specifications than does the space station, so the controller design has more specific goals. This will test the ability to include the performance specs in the controller design.

Chapter 2

Software Development

2.1 Software Objectives

The goal of the software development was to produce a software package which would accept a state space description of a system and return a state space description of the l^1-optimal controller. For compactness of computation (and to take advantage of matrix manipulation routines generally available), it was desired to keep all calculations in state space representation whenever possible. Since the linear program, which is the heart of the l^1 optimization, involves the actual time sequences of the l^1-optimal impulse response matrix, state space representation must be abandoned at certain points.

As implemented for this project, the code is written partly in MATRIXx[1] and partly in FORTRAN as subroutines which are linked into MATRIXx through the USER function. This was done mainly for convenience as MATRIXx was

[1]MATRIXx ©1988, Integrated Systems, Inc., is a commercial software package for linear system analysis and control design

being used throughout the project. With a good library of matrix functions, the code may be easily translated wholly into FORTRAN, C, or another language of choice. Other than matrix inversion, the only complex functions used are generalized eigenvalue/vector calculation and minimal system realization.

2.2 The Algorithm

Following standard notation for controller design, a two input two output system description is taken as the input. A block diagram of such a system is shown in Fig. 2.1. **G** represents a 2×2 block matrix with the four blocks being the transfer functions (possibly matrices) from **w** and **u** to **z** and **y**. The inputs **u** are the control inputs and **y** are the measured outputs. The outputs **z** are the regulated outputs, and **w** are the exogenous inputs. For the l^1- optimal problem, the objective is to find the stabilizing controller **K** which minimizes the l^1 norm of the closed loop transfer functions from **w** to **z**. The input to the algorithm is the state space system matrix **G** shown in Eqn. 2.1.

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \mathbf{z} \\ \mathbf{y} \end{bmatrix} = \overbrace{\begin{bmatrix} \mathbf{A} & \mathbf{B_1} & \mathbf{B_2} \\ \mathbf{C_1} & \mathbf{D_{11}} & \mathbf{D_{12}} \\ \mathbf{C_2} & \mathbf{D_{21}} & \mathbf{D_{22}} \end{bmatrix}}^{\mathbf{G}} \begin{bmatrix} \mathbf{x} \\ \mathbf{w} \\ \mathbf{u} \end{bmatrix} \tag{2.1}$$

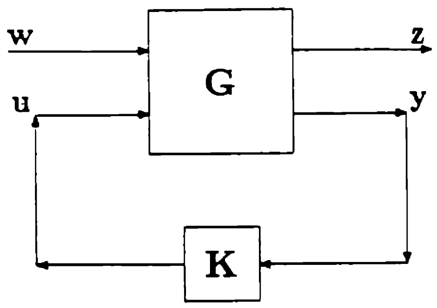


Figure 2.1: Two Input Two Output Block Diagram

The following is a basic outline of the procedure for calculating an l^1-optimal controller for the system described by Fig. 2.1. For a full treatment of l^1 theory, see [1].

1. Using a parametrization of the controller $\mathbf{K}$ in terms of known system parameters and an arbitrary stable factor $\mathbf{Q}$, the closed loop transfer function $\mathbf{H}_{zw}$ from $\mathbf{w}$ to $\mathbf{z}$ is written as:

$$\mathbf{H}_{zw} = \mathbf{T}_1 - \mathbf{T}_2\mathbf{Q}\mathbf{T}_3 \tag{2.2}$$

where the $\mathbf{T}_i$'s are known and $\mathbf{Q}$ is from the parametrization of $\mathbf{K}$ [6,9].

2. The interpolation points are calculated. Since $\mathbf{Q}$ is stable, it cannot cancel the left unstable zeros of $\mathbf{T}_2$ or the right unstable zeros of $\mathbf{T}_3$. These are multivariable zeros and as such, each has an associated vector. The zeros of $\mathbf{T}_2$ will be denoted a_i with associated vectors $\vec{\beta}_i$ for $i = 1, \ldots, N$ and

the zeros of $\mathbf{T}_3$ by b_i with associated vectors $\vec{\varsigma_i}$ for $i = 1, \ldots, M$. These are found by solving the generalized eigenvalue/vector problem for $\mathbf{T}_2^T$ (remember the left zeros are needed) and for $\mathbf{T}_3$.

Since the problem is in discrete time, the unstable zeros are those which are outside the unit disk, assuming a z-transform defined such that z^{-1} is the unit delay.

3. At this point calculations cannot be done in state space. $\mathbf{H}_{zw}$, an $m \times n$ matrix, is expanded into its time sequence representation $\mathbf{h}_{zw}$. Denote $\mathbf{H}_{zw}$ as $\mathbf{\Phi}$ and $\mathbf{h}_{zw}$ as ϕ. The elements of ϕ will be sequences of length l. In some cases the optimal solutions are of infinite length and truncating will achieve a suboptimal solution. Larger l will achieve a solution closer to the optimum, so a compromise between solution length and solution optimality must be made.

   The interpolation conditions shown in Eqns. 2.3 and 2.4 are set up as constraints of a linear program which solves for the elements of ϕ. Since standard linear programming requires positivity of variables, each element of ϕ is represented as $\phi_{ij}(k) = x_{ij}(k) - y_{ij}(k)$ with all $x, y \geq 0$. The linear program is shown in Fig. 2.2.

$$\vec{\beta_i}\mathbf{\Phi}(a_i) = \vec{\beta_i}\mathbf{T}_1(a_i) \quad \forall i = 1, \ldots, N \tag{2.3}$$

$$\mathbf{\Phi}(b_i)\vec{\varsigma_i} = \mathbf{T}_1(b_i)\vec{\varsigma_i} \quad \forall i = 1, \ldots, M \tag{2.4}$$

$$\boxed{\begin{array}{ll}
\max \mu \quad \text{s.t.} & \\
\displaystyle\sum_{k=0}^{l}\sum_{j=1}^{n} x_{ij}(k) + y_{ij}(k) \le \mu & \forall i = 1,\ldots,m \\
\displaystyle\sum_{i=1}^{m}\sum_{k=0}^{l} \beta_{pi} x_{ij}(k) a_p^{-k} - \beta_{pi} y_{ij}(k) a_p^{-k} = \sum_{i=1}^{m} \beta_{pi} \Phi_{ij}(a_p) & \left\{\begin{array}{l} \forall p = 1,\ldots,N \\ \forall j = 1,\ldots,n \end{array}\right. \\
\displaystyle\sum_{j=1}^{m}\sum_{k=0}^{l} \varsigma_{rj} x_{ij}(k) b_r^{-k} - \varsigma_{rj} y_{ij}(k) b_r^{-k} = \sum_{j=1}^{m} \Phi_{ij}(b_r) \varsigma_{rj} & \left\{\begin{array}{l} \forall r = 1,\ldots,M \\ \forall i = 1,\ldots,m \end{array}\right.
\end{array}}$$

Figure 2.2: Linear Program for l^1 Solution

4. After solving the linear program, the elements of ϕ are known. At this stage the solution can be transformed back to state space. If only $\mathbf{H}_{zw}$ is desired, the procedure is complete. Generally the optimal controller is of interest, however, and an additional step is necessary.

5. The optimal solution $\mathbf{\Phi} = \mathbf{T}_1 - \mathbf{T}_2\mathbf{Q}\mathbf{T}_3$ can be manipulated to solve for $\mathbf{Q}$ as shown in Eqn. 2.5.

$$\mathbf{Q} = \mathbf{T}_2^{-1}(\mathbf{T}_1 - \mathbf{\Phi})\mathbf{T}_3^{-1} \tag{2.5}$$

   $\mathbf{Q}$ can then be inserted into the factorization of $\mathbf{K}$ to yield the final result.

The procedure just outlined is complete as long as $\mathbf{T}_2$ is not "tall" (more rows than columns) and $\mathbf{T}_3$ is not "wide" (more columns than rows). In either of these latter cases, there will be infinitely many interpolation points and the algorithm must be altered slightly.

The first modification is to Step 2. Rather than calculating the interpolation points for the entire $\mathbf{T}_2$ and $\mathbf{T}_3$, each of these is partitioned into square blocks (and possibly a non-square remaining block). The interpolation points are calculated for each of these blocks and the zero vectors are augmented with leading and/or trailing zeros to multiply $\mathbf{T}_1$.

The second modification is also to Step 2 to reflect the fact that there are actually infinitely many zeros of $\mathbf{T}_2$ and/or $\mathbf{T}_3$. A set of relations are defined which multiply $\mathbf{T}_2$ or $\mathbf{T}_3$ to yield zero. For example if $\mathbf{T}_2 = \begin{bmatrix} U_1 & U_2 \end{bmatrix}^T$ where U_1 and U_2 are polynomial ratios, then the relation is described by Eqn. 2.6.

$$\begin{bmatrix} -U_2 & U_1 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = 0 \tag{2.6}$$

If $\mathbf{T}_2$ had more than one column the relations would become more complex but will still be a vector of polynomial ratios. The number of relations for each matrix is equal to the dimension of the null space of that matrix. If all the elements are left in full form (no pole-zero cancellation), then they all have the same denominator, and the relations may be described using only the numerator polynomials.

2.3 Implementation

A flowsheet and code listing for the l^1 software implemented in this thesis is contained in Appendix A. Excep the portion of the code which sets up the LP, MATRIXX user functions are defined to solve the problem. With a good library of matrix functions and some knowledge of control theory, however, these may be easily translated entirely into FORTRAN (used here for the LP portion of the code) or any convenient language.

This algorithm is intended for discrete time systems, and the z-transform is defined such that z^{-1} is the unit delay. With this definition, stable systems are those that have all poles p_i inside the unit disk ($\|p_i\| < 1$). For systems in state space representation, the equivalent is that all eigenvalues of the **A** matrix are inside the unit disk.

The two keys to this software are a robust set of matrix functions and a robust LP solver. The necessary matrix functions include eigenvalues, generalized eigenvalues, inversion, and multiplication. The LP solver is perhaps the more difficult part as the LP's are sensitive to such things as tolerances, initial guesses, variable limits, and the conditioning of the problem itself. It is often necessary to change some of the problem parameters one or more times in order to obtain convergence to a solution. As a result, this usually turns out to be an iterative step.

The first step in the program is the calculation of $\mathbf{T}_1$, $\mathbf{T}_2$, and $\mathbf{T}_3$. These quantities are determined using the plant ($\mathbf{G}_{22}$) and any pair of associated regulator and estimator gains.

A first attempt was to select the gains such that the resulting poles were all at the origin. This results in polynomial matrices which are easy to deal with in subsequent steps. It may happen, though, that one or more of the states is uncontrollable or unobservable, thereby making it impossible to perform such placement. This is true, for instance, if the $\mathbf{w}$ inputs are weighted. Also, extremely large gains may be required which may introduce numerical difficulties at later stages. Finally, due to roundoff errors, the eigenvalues never ended up exactly at the origin. In fact, the higher the order of the system, the more severe these problems become. For example, suppose the coefficient of the z^0 term of the characteristic polynomial is 10^{-16} instead of 0. The z^n coefficient is always 1. For $n = 2$ the poles are at $\pm 10^{-8}\mathrm{i}$. For $n = 8$ they are at $\sqrt[8]{10^{-16}}$ which have magnitude 10^{-2} compared to 10^{-8} in the $n = 2$ case.

A more practical system is to input a set of state and input/output weights and to solve the corresponding regulator and estimator problems. This gets away from the simplicity of having polynomial matrices, but offers some freedom in the choice of gains which may allow some fine tuning to eliminate the problems described above.

Given a parametrization, the next step is to determnine the interpolation points. This is done by solving the generalized eigenvalue problem for $\mathbf{T}_2$ and $\mathbf{T}_3$. If $\mathbf{T}_2$ is tall or $\mathbf{T}_3$ is wide, then the interpolations are found for each $n \times n$ block of $\mathbf{T}_2$ and each $p \times p$ block of $\mathbf{T}_3$ where $\mathbf{T}_2$ is $m \times n$ and $\mathbf{T}_3$ is $p \times q$. The interpolations of any leftover blocks are also found. The associated vectors are found by evaluating the singular value decomposition (SVD) of the system at each interpolation point. The null vectors are given by the columns of $\mathbf{V}$ ($\mathbf{G}_{SVD} = \mathbf{UQV}^*$) associated with the zero singular values. This seems to be a more robust calculation than finding the eigenvectors associated with the zero eigenvalues.

Multiple zeros as interpolations impose derivative conditions. Each zero after the first implies a derivative with respect to z^{-1}. For lack of a state space method of calculating system matrix derivatives, each component is transformed into a numerator and denominator polynomial whose derivatives can easily be calculated.

Another method of finding the derivatives is to calculate the derivatives of the system pulse response. This is equivalent to carrying out the divisions of the polynomial ratios and is simpler because the rule for derivatives of ratios does not need to be used. The drawback, though, is that the pulse response is not necessarily finite. An approximation may be made by truncating the pulse response, but if the pulse response decays very slowly, a prohibitively large number of elements may be necessary for an adequate approximation.

A special case of multiple zeros is when there are common zeros between $\mathbf{T}_2$ and $\mathbf{T}_3$. These impose derivative conditions also as can be best seen in the simple SISO case. In a SISO system, the factors $\mathbf{T}_2$, $\mathbf{Q}$, and $\mathbf{T}_3$ commute. Thus, $\mathbf{T}_2\mathbf{Q}\mathbf{T}_3 = \mathbf{T}'\mathbf{Q}$ where $\mathbf{T}' = \mathbf{T}_2\mathbf{T}_3$. If $\mathbf{T}_2$ and $\mathbf{T}_3$ had a common zero, $\mathbf{T}'$ will have that zero with a multiplicity of two which gives a derivative condition. The MIMO case is not so simple because one of the zeros is from the left and the other is from the right. In this case both the right and left vectors must multiply the derivative system to produce zero.

For systems with common multiple zeros between $\mathbf{T}_2$ and $\mathbf{T}_3$, the search strategy is somewhat complicated. If there are p repetitions of a zero z_i in $\mathbf{T}_2$ and q repetitions in $\mathbf{T}_3$, the correct approach seems to be to start with $\max(p, q)$. If $p > q$, then the left vector is the one associated with the p^{th} occurence of the zero in $\mathbf{T}_2$. This should be used with all q of the right vectors associated with that zero in $\mathbf{T}_3$. For $p < q$, the roles of $\mathbf{T}_2$ and $\mathbf{T}_3$ should be reversed. It is not entirely clear that this is really the appropriate strategy, so this is an aspect of the theory which should be explored.

The next step is the calculation of relations for non-square systems. Although the relations could in fact be calculated in state space, they are currently calculated using polynomial transfer functions and pulse responses. Since pole-zero cancellations are not carried out on $\mathbf{T}_2$ and $\mathbf{T}_3$, the elements of these matrices have identical denominator polynomials. Thus, only the numerators are used for the relations.

To find the relations, a matrix $\mathbf{M}_p$ of the numerator polynomials is formed, and Eqn. 2.7, where $\vec{r}$ is the desired result, is solved using Cramer's Rule.

$$\mathbf{M}_p\vec{r} = \vec{0} \tag{2.7}$$

For "tall" $\mathbf{T}_2$ and "wide" $\mathbf{T}_3$, $\mathbf{M}_p$ will be an $m \times n$ block matrix with $m < n$. Each of the $m \times n$ blocks is the 1xNS numerator polynomial coefficient vector of the corresponding transfer function of $\mathbf{T}_2$ or $\mathbf{T}_3$. NS is the number of states used to represent $\mathbf{T}_2$ and $\mathbf{T}_3$ in state space. It is assumed that this matrix has rank m which means that there will be $(n-m)$ linearly independent solutions to Eqn. 2.7. Since multiplying polynomials amounts to convolving the coefficient vectors, a special routine was written to do the determinant calculations for Cramer's Rule.

A final detail in setting up the problem is to get zeros at infinity since the generalized eigenvalue problem does not return these. To do this, the minimum relative degree (# of poles - # of finite zeros) of the elements of each row of $\mathbf{T}_2$ and each column of $\mathbf{T}_3$ are calculated. The minimum possible relative degree of each element (i,j) of $\mathbf{T}_2\mathbf{Q}\mathbf{T}_3$ is then the sum of the corresponding minimum values from row i of $\mathbf{T}_2$ and column j of $\mathbf{T}_3$. This is a result of the fact that through Q, each entry of row i in $\mathbf{T}_2$ multiplies each column of $\mathbf{T}_3$ in the ij^{th} entry of $\mathbf{T}_2\mathbf{Q}\mathbf{T}_3$.

The relative degree of a ratio of polynomials is exactly the number of zeros at infinity — the desired quantity. If a function has r zeros at infinity, then the first r elements are zeros. Translated into LP constraints, these mean that the first r coefficients of an entry in $\mathbf{\Phi}$ are equal to the first r coefficients of the corresponding entry in $\mathbf{T}_1$.

The LP itself is the heart of the l^1 optimization. Typical LP solvers require tolerances to determine whether constraints are considered to be satisfied, limits on variable values, and initial guesses for variable values. All of these have an impact on convergence to a solution and may need to be adjusted one or more times to achieve a satisfactory solution. The NAG FORTRAN library routine E04NAF is used in this thesis.

The solution to the LP is the closed loop pulse response of the transfer function from the exogenous inputs to the regulated outputs. In general, this is not the only item of interest. For example, the optimization may have been done on the sensitivity, but it is desired to know the transfer functions from plant inputs to plant outputs. In such a case additional calculations are necessary to obtain the desired results.

One item of interest is usually the optimal controller. An optimal objective function is not useful in practice unless one knows the controller to realize it. The controller is calculated using the $\mathbf{Q}$ parameter and the factors $\mathbf{M},\mathbf{N},\mathbf{X}$, and $\mathbf{Y}$. These factors are part of a doubly coprime factorization of the plant as shown in Eqs. 2.8-2.10.

$$G_{22} = NM^{-1} = \tilde{M}^{-1}\tilde{N} \tag{2.8}$$

$$\begin{bmatrix} \tilde{X} & -\tilde{Y} \\ -\tilde{N} & \tilde{M} \end{bmatrix} \begin{bmatrix} M & Y \\ N & X \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \tag{2.9}$$

$$K = (Y - MQ)(X - NQ)^{-1} = (\tilde{X} - Q\tilde{N})^{-1}(\tilde{Y} - Q\tilde{M}) \tag{2.10}$$

Although the calculation seems rather straightforward, there are several things which need to be watched. First, $\mathbf{T}_2$ and/or $\mathbf{T}_3$ may be strictly proper when solving for Q, in which case the inverses will not be representable in state space form. Since these are discrete time systems, the sequences can be advanced (multiplied by z) until the system is proper and the inverse can be calculated. Since each multiplication by z becomes a time delay (z^{-1}) when the system is inverted, the end result (after multiplication by the modified inverse) must be advanced appropriately. That this can be done without creating a noncausal system is ensured since the zeros at infinity were included as interpolations, creating an appropriate number of leading zeros in the expression for $\mathbf{T}_1 - \mathbf{\Phi}$.

Secondly, if either $\mathbf{T}_2$ or $\mathbf{T}_3$ had no interpolations (generally because they were either wide or tall, respectively), it is necessary to be sure to use a stable inverse when calculating Q. If there were interpolations, then any inverse can be used because any unstable poles will be cancelled by zeros created by the interpolations. As implemented, the software does not check to be sure the inverse is a stable one. It is necessary for the user to check this. A method of ensuring that a stable inverse is found should be devised for this step.

Another factor to consider is that the $(\mathbf{X}-\mathbf{N}\mathbf{Q})$ or $(\tilde{\mathbf{X}}-\mathbf{Q}\tilde{\mathbf{N}})$ term is inverted when calculating the controller. For some systems, numerical difficulties will be encountered when trying to invert matrices. Thus, the choice of parametrizations should be influenced by ease of calculation in later stages. This will become apparent in the space station example in Chapter 3.

Also, it may occur in simpler systems that the system becomes degenerate (reduces to a constant and thus requires no states for a state space representation) at some point in the calculations. This condition will cause MATRIXx to signal an error and stop execution and so must be watched for in order that appropriate action can be taken before the calculations fail.

Special considerations and prnblems encountered when implementing the l^1 algorithm were presented in this section. For the actual code, see Appendix A.

2.4 Software Performance

As implemented, the code is able to return an l^1- optimal controller for a system given the two-input two-output state space description as input. Examples 4.2, 5.5, and 6.6 from Dahleh [1] were used for initial debugging, and all produce the expected results. These problems are rather simple and do not correspond to any particular real control problems.

On more complex examples, however, some problems did manifest themselves. There were two basic types of problems — the example-specific and the software or algorithm inherent problems. The inherent problems are discussed here and the example specific problems are discussed in the following chapters.

The first problems encountered were with the MATRIXx software. One difficulty was with the calculation of the system zeros via the 'ZEROS' command. In some cases not all zeros were found, and in some cases different zeros were found for single-input single-output (SISO) systems when the system was transposed. Since SISO systems are identical when transposed, the zeros should be identical.

Another problem with MATRIXx is the limited ability to program. Particularly notable are the limited conditional statement which does not have logical operators ('AND', 'OR', etc.) and the lack of conditional looping. Another drawback is the limited ability to check for or recover from errors which may be correctible. If a MATRIXx error is encountered, the procedure must be restarted from scratch after having made the necessary adjustments. For these reasons, it appears that translation wholly into FORTRAN or C would be a useful step.

Another difficulty is that with the current algorithm the l^1 problem uses a large amount of memory for even a fairly small system. This was in fact a limiting factor in several of the examples which were attempted. In other cases the example would not encounter memory problems but would overwhelm the capabilities of the LP solver that was used.

As an example of typical problem dimensions, consider the problem of a modified F18 aircraft [10] which was a candidate problem for l^1 application. This system had three control outputs and six regulated variables — a total of eighteen transfer functions. Now assume that an impulse response of ten elements was desired. This yields a total of $18 \times 10 = 180$ $\mathbf{\Phi}$ elements. Each element becomes two variables in the LP using the convention $\phi_{ij}(k) = x_{ij}(k) - y_{ij}(k)$ where the x's and y's are positive variables. Add in the optimization variable μ and the total becomes 361 variables.

Now the constraints must be considered. First there are six constraints stating that the l^1 norms of the rows are less than or equal to the optimal value μ. For each interpolation there are either three or six individual constraints as each null vector multiplies either the three columns (left zeros) or the six rows (right zeros). Assume a total of nine interpolation constraints for simplicity.

Next are the relations which form the bulk of the constraints. There will be three relations since there are three more rows than columns. The F18 system has four states which leads to relation elements of length 10 (resulting from the use of Cramer's Rule). Each of these relations is multiplied with each column of $\mathbf{T}_1$ for a total of $3 \times 3 = 9$ relations. Since the system is in polynomial form the multiplications become convolutions of coefficient vectors. Thus, each relation comprises $10 + 10 - 1 = 19$ separate constraints in the LP for a total of $9 \times 19 = 171$.

Finally there are about 30 constraints for zeros at infinity, giving a total of approximately 216 $(= 6 + 9 + 171 + 30)$. The total problem is thus about 80,000 elements and is sparse, a problem beyond the capabilities of typical LP solvers. Also consider that a roughly equivalent space is needed for work space by the LP solver. Each of these elements is represented as a FORTRAN double precision real number which occupies 8 bytes of memory. The problem data alone is seen to occupy over $1,000,000$ bytes meaning that a reasonably powerful computer is needed.

Chapter 3

Space Station Example

3.1 Space Station Model

For examining the problem of space station attitude control and momentum management, a dynamic model of the station's orbital movement was necessary. It was desired to use as simple a model as possible while retaining the essential features of the system. These main features are the rigid body dynamics of the station, the CMG dynamics, and the gravity gradient effects.

For space station analysis, three coordinate frames are utilized for convenience of reference. First is the body frame, denoted by x_B, y_B and z_B. This coordinate frame is aligned with the principal axes of the station and is convenient for looking at quantities associated with the station itself such as the CMG torques.

Secondly, there is the local vertical local horizontal (LVLH) frame, denoted by x_L, y_L and z_L. This frame is referenced to the earth and the station orbit.

The line from the center of the earth through the space station defines z_L. The line perpendicular to z_L in the direction of the orbit defines x_L, and y_L completes a right-handed coordinate system.

The final frame is the inertial frame, denoted by x_I, y_I, and z_I, which is fixed in space. Notice that both the body and LVLH frames are functions of orbital position. The inertial frame is necessary as a fixed reference for the moving frames. The inertial frame is usually fixed as being coincidental with the LVLH frame at some initial time T_0. The LVLH frame then rotates at orbital frequency with respect to the inertial frame (after one orbit they are again coincident).

It will be desired that the space station orbit in LVLH attitude. That is, the body coordinates should be aligned with the LVLH coordinates at all times. This means that, in the inertial frame, the space station must be rotating with orbital frequency ω_0 as it orbits.

The basic equations for the space station rigid body dynamics are Eqns. 3.1 and 3.2 where **H** is the momentum of the station, **I** is the inertia tensor, and $\vec{\omega}$ is the station rotation in the body frame.

$$\mathbf{H} = \mathbf{I}\vec{\omega} \tag{3.1}$$

$$\dot{\mathbf{H}} = \sum \tau_{ex} = \mathbf{I}\dot{\vec{\omega}} + \vec{\omega} \times \mathbf{I}\vec{\omega} \tag{3.2}$$

Eqn. 3.2 relates the change in momentum of the station to the external torques (τ_{ex}) acting on the system. τ_{ex} can be expanded as shown in Eqn. 3.3, where

τ_{CMG} is the reaction torque of the CMG's on the station, τ_{gg} is the gravity gradient torque, and τ_d is a disturbance torque (which may itself have several contributing terms).

$$\sum \tau_{ex} = \tau_{CMG} + \tau_{gg} + \tau_d \tag{3.3}$$

The final set of simplified dynamic equations is reached after substituting an expression for the gravity gradient torque, referencing all quantities to the appropriate frames, and assuming small cross products of inertia and small angular deviations. The results are shown in Eqs. 3.4-3.6.

$$I_1\ddot{\theta}_1 + 4\omega_0^2(I_2 - I_3)\theta_1 - \omega_0(I_1 - I_2 + I_3)\dot{\theta}_3 = -u_1 + w_1 \tag{3.4}$$

$$I_2\ddot{\theta}_2 + 3\omega_0^2(I_1 - I_3)\theta_2 = -u_2 + w_1 \tag{3.5}$$

$$I_3\ddot{\theta}_3 + \omega_0^2(I_2 - I_1)\theta_3 + \omega_0(I_1 - I_2 + I_3)\dot{\theta}_1 = -u_3 + w_3 \tag{3.6}$$

The u_i's represent the body axis components of the control torques, the w_i's represent the disturbance torques, and the θ_i's are the roll, pitch, and yaw (1-2-3) Euler angles of the body axes referenced to the LVLH frame. Similarly, the simplified CMG dynamics are given by Eqs. 3.7-3.9.

$$u_1 = \dot{h}_1 - \omega_0 h_3 \tag{3.7}$$

$$u_2 = \dot{h}_2 \tag{3.8}$$

$$u_3 = \dot{h}_3 + \omega_0 h_1 \tag{3.9}$$

These sets of equations are standard for the problem and a detailed derivation can be found in Bishop [4] or Wie [5].

Other than the assumptions of small angles and small products of inertia made to simplify the equations, the other main assumption in the model is that flexible body dynamics can be ignored. This is not really a major approximation for attitude control since the flexible body dynamics are generally not in the same frequency range as orbital dynamics. Also, this example is not intended to contain such great detail.

The actual values for the I_i and w_i were taken from Wie [5] and are typical numbers for most proposed space station models. They are essentially similar to the numbers used in Bishop [4]. Looking at Eqn. 3.5 it is seen that the pitch dynamics are decoupled from the roll and yaw dynamics. This allows control design to be done on two smaller systems rather than on one large system.

3.2 Pitch Axis Controller Design

The matrices **A**, **B**, and **C** for the state space description of the pitch dynamics described by Eqn. 3.5 and Eqn. 3.8 are listed in Appendix B. Both the continuous time and discretized versions are given. The four states in the pitch axis dynamics are the pitch rate, the pitch, the CMG momentum, and the integral of the CMG momentum. The integral of the momentum is included to ensure that the momentum settles to zero for a step disturbance.

The first attempt at an l^1 controller design was for the pitch axis dynamics

Block	*Zeros*
pitch	1, 1, -1
h	1, 1.3732
$\int h$	1.3732, -1
control	1, 1, 1.3732

Table 3.1: Interpolations on $\mathbf{T}_2$

since this was the simplest system. Initially, the regulated variables were the pitch angle, the CMG momentum, the integral of the momentum, and the control input. Scalar weights were used to scale the variables to approximately unity.

For this particular configuration, both $\mathbf{T}_2$ and $\mathbf{T}_3$ are 4×1 matrices. $\mathbf{T}_3$ has no interpolations, and for $\mathbf{T}_2$ interpolations are found for the four SISO blocks since it is a "tall" matrix. The interpolations for each block are listed in Table 3.1. The interpolations are listed to point out one of the MATRIXx problems that was found. For the $\int h$ block, the 'ZEROS' function of MATRIXx does not find the zero at $z = -1$. This turned out not to be a problem since this is apparently a redundant constraint and was satisfied despite the fact that it was not explicitly included in the LP formulation.

The regulated variables, the assigned weights, and the resulting l^1 norms for several solutions are given in Table 3.2. The first noticeable feature of these

	Regulated Variables			
	pitch	CMG momentum (h)	$\int h$	control
weight	1.0	10^{-4}	10^{-6}	10^{-2}
optimal norms				
$l = 9$	0.1164	1.2577	1.2653	1.2653
$l = 16$	0.0975	0.9332	0.9482	0.9482
$l = 20$	0.0973	0.9305	0.9454	0.9454

Table 3.2: Optimization Results With Scalar Weights

results is that as expected, a larger length of solution (l) yields a lower norm. This will be the case whenever there are interpolations on the unit circle or relations. Note also that the norms in Table 3.2 are the scaled norms. To get the actual values, these numbers should be divided by the scale factors. The units for the unscaled norms are radians, ft-lbs, ft-lb-sec, and (ft-lb)/sec, respectively.

The l^1 norms of the responses represent the largest possible ∞-norm of the responses to any l^∞ signal of unit magnitude. For the $l = 20$ case, then, the largest possible magnitudes of the regulated variables would be 5.6 degrees, 9305 ft-lbs, 945400 ft-lb-sec, and 95 (ft-lb)/sec. Of more direct interest, however, is the response to the particular disturbance used in this model:

$$w(t) = 4 + 2\sin\omega_0 t + 0.5\sin 2\omega_0 t. \tag{3.10}$$

Figures 3.1 and 3.2 show the pitch angle responses for the $l = 9$ and $l = 20$ cases. Figures 3.3 and 3.4 show the CMG momentum responses, and Figures 3.5 and 3.6 show the control input.

An interesting characteristic of these solutions is that while the peak magnitudes of the responses decrease with increasing l, the steady-state peak values actually increase. Presumably this trend would continue until the two quantities were equal. These responses are nearly identical to the LQR synthesized results of Wie [5].

A feature of the l^1 optimization revealed by this initial design is that some of the LP constraints are redundant. The optimization was done neglecting the derivative conditions (multiple zeros were ignored) and neglecting the $\int h$ interpolation at $z = -1$ (MATRIXx problem), and the solution was identical to the case with these interpolations included. It is unknown whether any of the relation constraints were redundant, but this is a possibility. A means of eliminating the redundant constraints before starting the optimization would be very helpful, particularly in larger problems, since it would reduce the size of the LP.

Calculating the optimal controllers turned out to be very difficult. For the

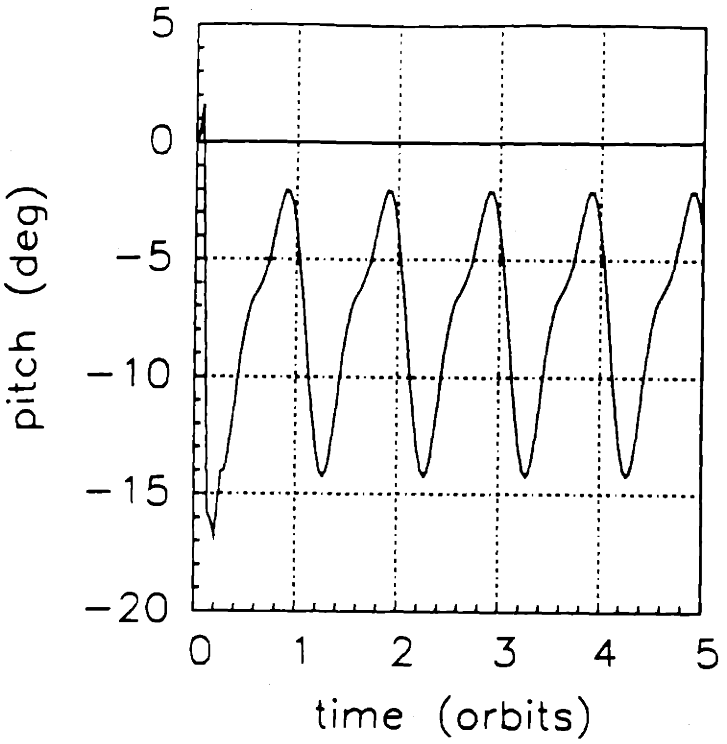


Figure 3.1: Pitch Disturbance Response ($l = 9$)

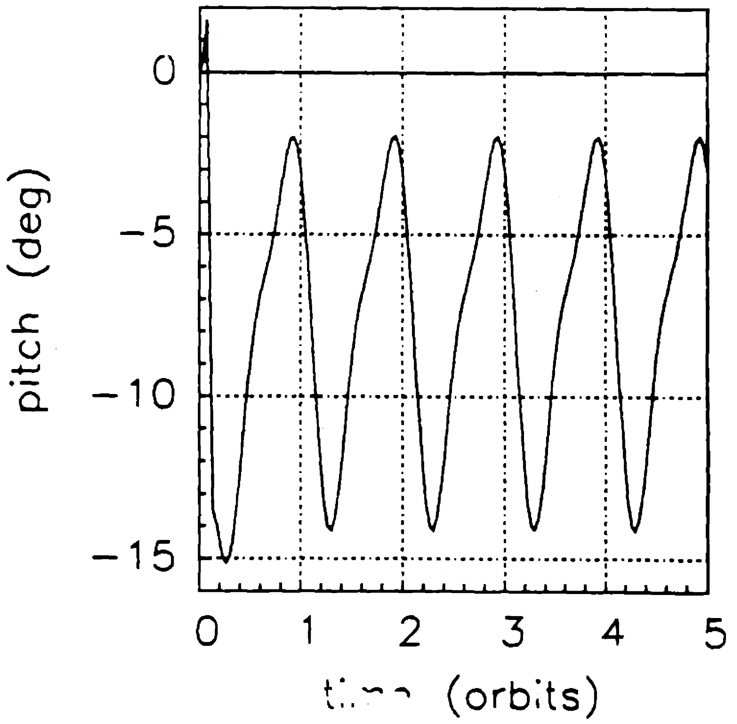


Figure 3.2: Pitch Disturbance Response ($l = 20$)

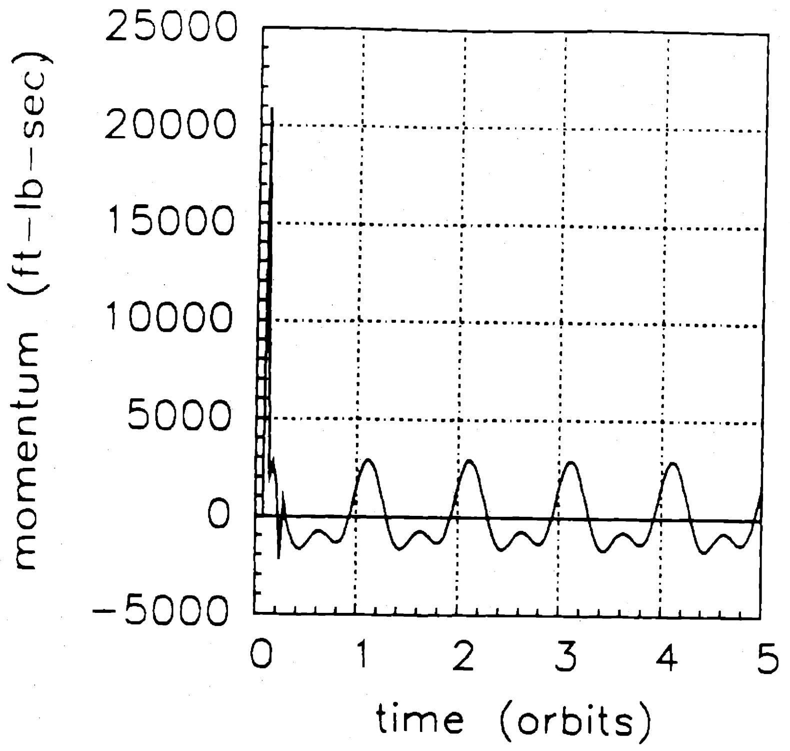


Figure 3.3: CMG Momentum Disturbance Response ($l = 9$)

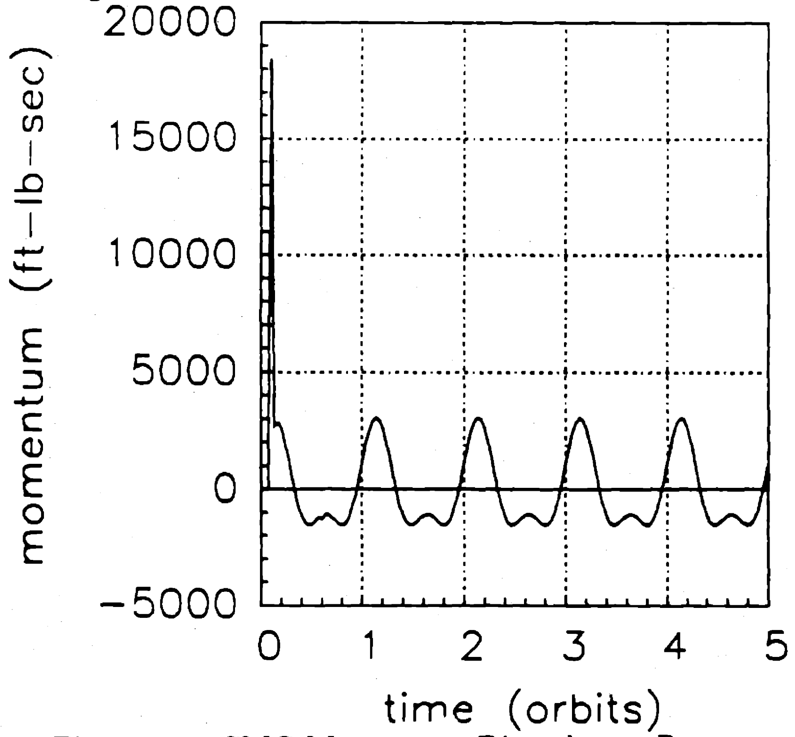


Figure 3.4: CMG Momentum Disturbance Response ($l = 20$)

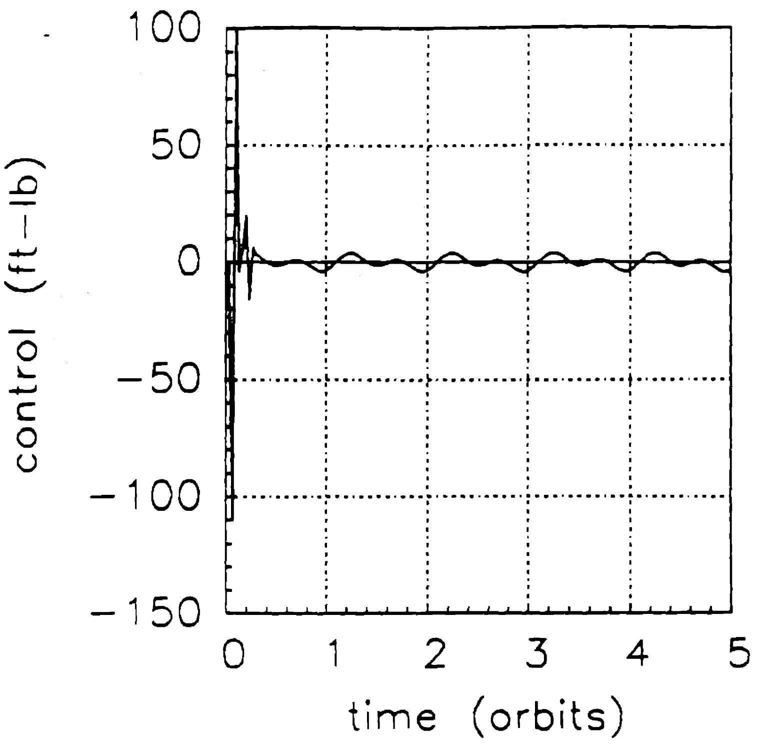


Figure 3.5: Control Input ($l = 9$)

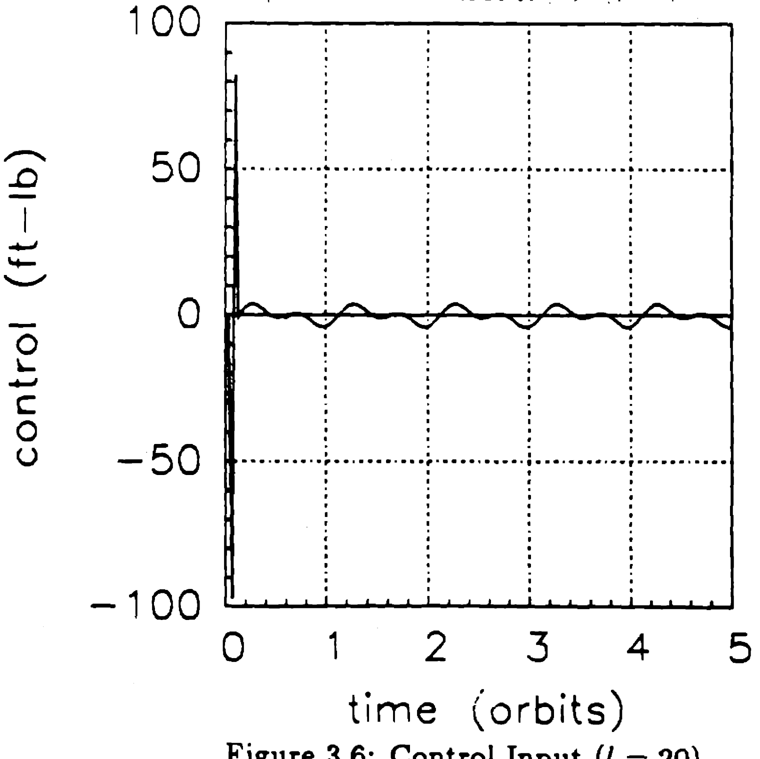


Figure 3.6: Control Input ($l = 20$)

$l = 9$ and $l = 16$ cases, controllers were calculated, but did not simplify as they should have. In the $l = 20$ case, the controller could not be calculated at all. The problem is with determining the minimal realization for a system. Even in the cases for which results were obtained, the controllers had many poles and zeros which should have been canceled but were not. With increasing l this problem became worse until the controller was no longer stabilizing.

For calculating the controller the code implemented here was nearly useless. The process had to be carried out one step at a time with careful intervention by the user at each step to manually perform the appropriate cancellations and simplifications. This is a laborious task and is all but impossible beyond SISO systems.

To explore the effect of changing the problem, the scale factors were varied and different combinations of the regulated variables were used. This particular problem was relatively insensitive to such changes. The solutions were essentially identical in all cases.

The next design approach was to specify rejection of the sinusoidal components of the disturbance in the pitch output. To do this, a sinusoidal weight was introduced on the pitch output of the station as shown in Fig. 3.7. After optimization, this weight was included as part of the controller rather than as part of the plant. The effect of this is to leave the transfer functions of the unweighted variables unchanged while including the denominator of the weighting

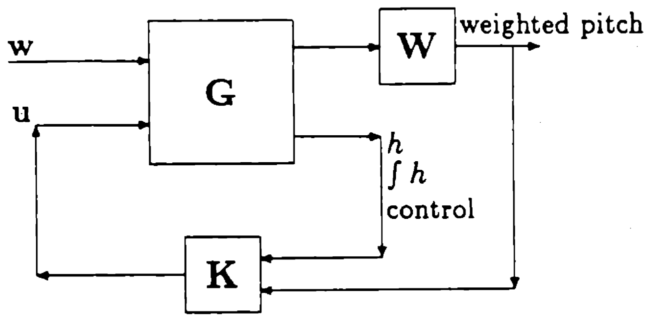


Figure 3.7: Weighted System

function in the numerator of the weighted transfer function, giving the desired rejection of the sinusoid.

The actual weighting function was Eqn. 3.10 which is the disturbance model. The effect of this weighting can be seen in the frequency response of the system. The scalar weighted system is shown in Fig. 3.8 and the sinusoidally weighted system in Fig. 3.9. The w-transform is used in Figs. 3.8 and 3.9 to scale the frequency axis to approximate the continuous time frequency. The relation is described by Eqn. 3.11.

$$z = \frac{1 + w\Delta t/2}{1 - w\Delta t/2} \tag{3.11}$$

Fig. 3.9 exhibits notches at orbital frequency (0.0011 rad/sec) and twice orbital frequency. This is the result of the weighting to reject the sinusoids at these frequencies. The result in the time domain is a non-cyclic response to inputs at these frequencies.

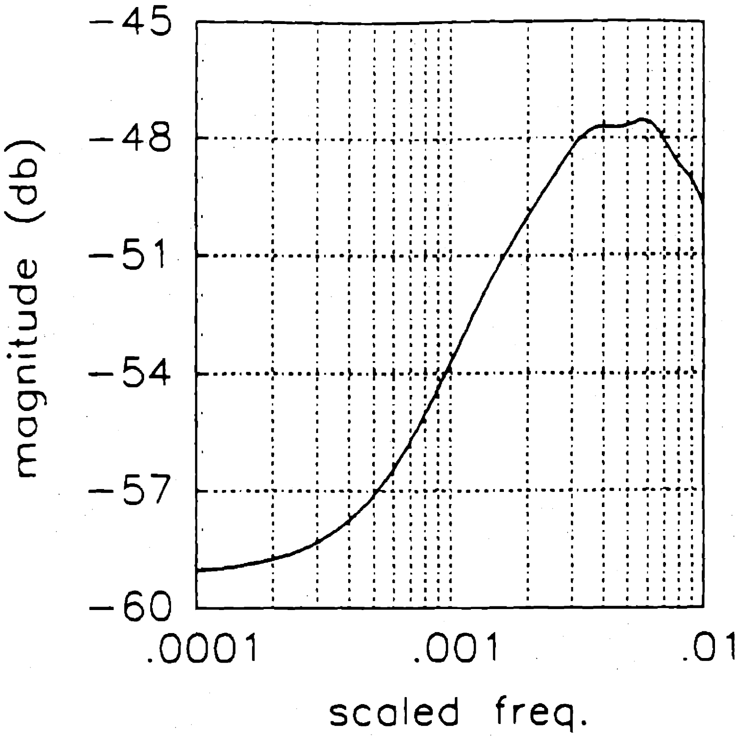


Figure 3.8: Frequency Response with Scalar Weight

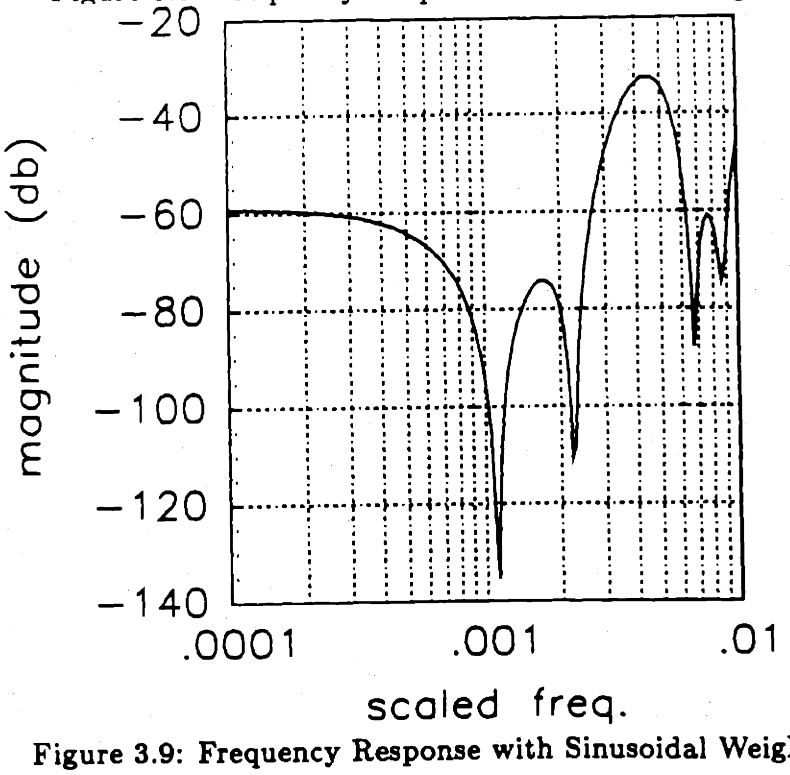


Figure 3.9: Frequency Response with Sinusoidal Weight

The first iteration of this design used only the weighted pitch as a regulated variable. The resulting system possessed the desired property of rejecting the sinusoidal disturbance, but the transients at startup from zero initial conditions were excessively large as were the required maneuvering rates. As a result, other combinations of variables were explored in an attempt to achieve better values.

Since the system regulating the weighted pitch was designed to reject the cyclic portion of the disturbance, the large magnitudes in the disturbance response must have been a result of the step component. To cure this, the regulated quantity was changed to the sum of the weighted and unweighted pitch. This had a profound effect on the disturbance response as can be seen in Fig. 3.10. After rearranging the system to include the weighting function in the the controller ($\mathbf{K} \rightarrow \mathbf{K}'$), the l^1 norm of the pitch pulse response is 12 degrees. This is the maximum pitch deviation for any arbitrary l^∞ signal of unit magnitude.

The fast oscillations during the startup transient indicate that large control magnitudes are necessary at these points and saturation limits may be exceeded. This situation could be rectified by including the control as a regulated variable to smooth out the peaks somewhat. This, of course, would have the effect of raising the magnitude of the pitch response slightly.

An interesting issue arose from the weighted pitch designs. One system was scaled to radians and one was scaled to degrees. It was expected that

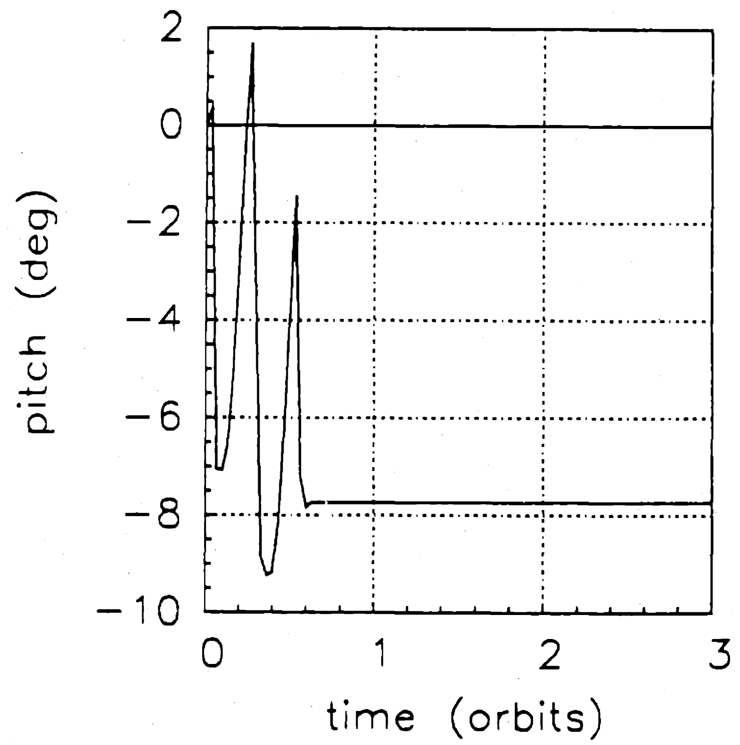


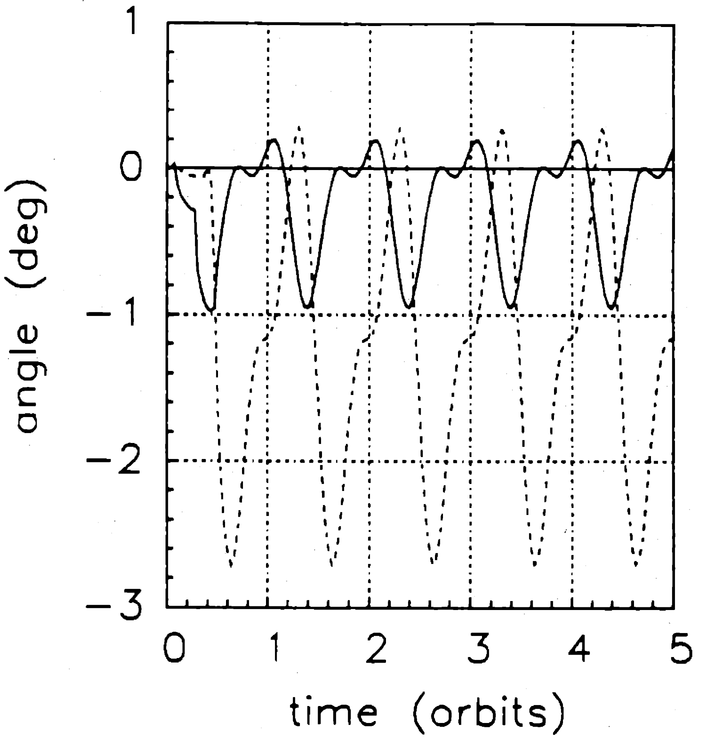
Figure 3.10: Response Regulating Weighted and Unweighted Pitch

these would yield identical results (scaled by $180/\pi$, of course), and did in the majority of cases. In several cases MATRIXX would calculate different zeros for the two systems yielding different optimizations. This of course was an error, but in the radian system, the solution had a curious property. The solutions for $l = 27, l = 28, l = 29$, and $l = 30$ were precisely identical and the solutions for $l = 45$ and $l = 60$ decreased as expected. The issue is whether the l^1 norm strictly decreasing with increasing l. If it is, then a plateau as observed in the above problem cannot be a valid solution.

3.3 Roll-Yaw Axis Controller Design

The roll/yaw control design for the space station was carried out for only the unweighted case because of difficulties obtaining the parametrization in the very first step. The discretized system is such that it is extremely difficult to obtain a stabilizing set of regulator gains. Only after many trial-and-error iterations was a set finally obtained. The discretized system and the regulator gains are shown in Appendix C.

Initially, the design was done regulating only the roll and yaw angles. The roll and yaw axis disturbances were both taken to be $1 + \sin(\omega_0 t) + 0.5\sin(2\omega_0)$ as in [5], and the resulting responses are shown in Fig. 3.11. In this case $l = 15$ and the optimal norm is 1.49 degrees. It was not reasonably possible, however,



roll —   yaw - - -

Figure 3.11: Roll and Yaw Responses

to derive the controller from the optimal solution.

Different values of l were used and the same trend was observed here as was seen in the pitch control design. Larger l yielded smaller norms (for example, $l = 20$ gives a norm of 1.25 degrees compared to 1.49 degrees at $l = 15$) but slightly larger peak values for the oscillations in steady state. It should also be noted that although the control and momentum variables were not regulated, their values fell well within the saturation limits.

Although the momentum and control variables fell within limits, an attempt was made to regulate the momentum in addition to the angular deviations. This was done to test the operation of the algorithm on a more complex system. The problem appeared to be formulated correctly, but the LP solver could not converge to a solution. More powerful LP software would probably overcome this difficulty.

The next step would be to specify rejection of the sinusoidal disturbances. This was not accomplished because of the difficulty in finding a stabilizing set of gains for the parametrization. Rejection cannot be done on the roll and yaw simultaneously, however, as shown in [5]. It is also shown that a cyclic disturbance at orbital frequency cannot be rejected for roll attitude. Such a disturbance can be rejected for the yaw attitude and roll momentum, however.

Chapter 4

X29 Example

4.1 X29 Model

The model for the X29 was taken directly out of [11]. A diagram of a forward swept wing aircraft is shown in Fig. 4.1. The model describes the use of the canard and flaperon to control the pitch angle and angle of attack. The pitch angle θ is the angle of the nose of the aircraft with respect to horizontal, and the angle of attack α is the angle of the nose with respect to the direction of the aircraft's velocity. The flight path angle γ is $\theta - \alpha$ and is the angle of the velocity with respect to horizontal. These quantities are all shown in Fig. 4.2.

Three different systems are presented in [11]. The first has the canard deflection as input and the angle of attack as output. The second has pitch attitude as output, and the third system uses both the canard and the flaperon to control the pitch attitude and the angle of attack. In this work, only the first case is treated.

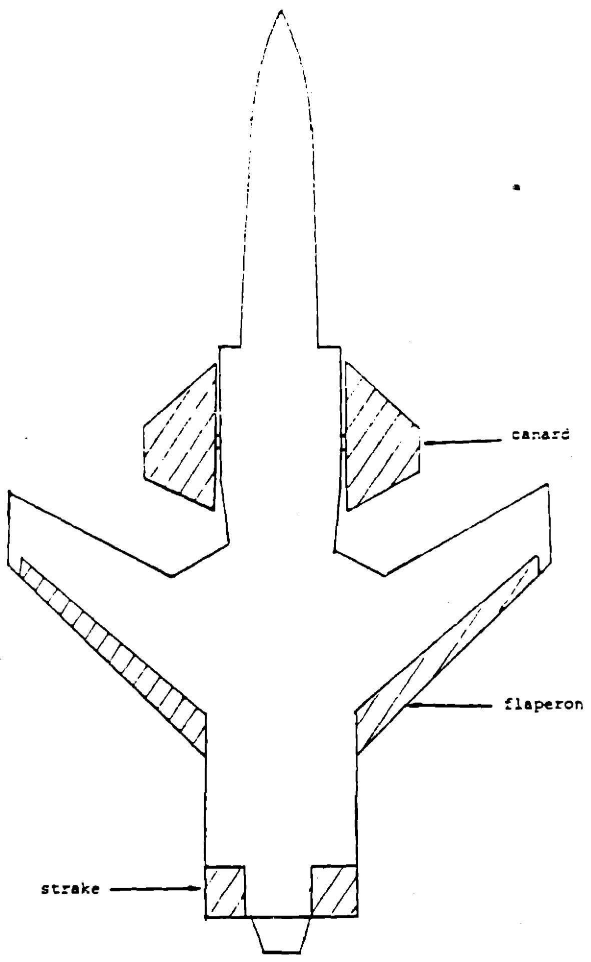


Figure 4.1: Diagram of the X29 Control Surfaces

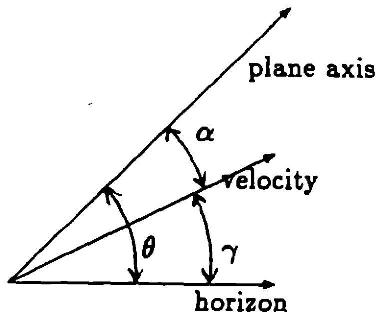


Figure 4.2: Pitch, Flight Path, Angle of Attack

4.2 Control Design

The challenge of the control design in the case of the X29 is to meet the various design specifications. For the first design case with the canard controlling the angle of attack, the specifications are that 10% error or less is required in command following for $0.01 < \omega < 1$ rad/sec, disturbances must be attenuated by at least a factor of 1.5 for $\omega < 1$ rad/sec, and the system must be robust in the face of multiplicative errors. Multiplicative errors are introduced from ignoring the high frequency wing torsion mode and from the scaling.

A final objective is to avoid exciting the wing bending mode at about 60 rad/sec. To accomplish this, it is desired to have the system bandwidth less than 60 rad/sec. This will cause attenuation of any excitation of the bending mode. In the design, this spec is approximated by requiring the crossover frequency of **GK** to be below 60 rad/sec.

All together, these specs require that the sensitivity be small and that the closed loop transfer function rolls off sufficiently at high frequency to guarantee robust performance. To meet these requirements, a good strategy is to minimize the sensitivity function, particularly in the range $\omega < 1$.

Minimizing the sensitivity is also a good objective in that the constraints of the optimization problem can easily be checked. The **Q** parametrization of the sensitivity has the property that it is equal to unity at the interpolations for

$\mathbf{T}_2$ and zero at the interpolations for $\mathbf{T}_3$. Thus, a quick and easy check was available while the code was executing.

With no weighting of the sensitivity, the design specifications were not quite met. The results are shown in Figs. 4.3-4.4. The sensitivity was sufficiently small in the appropriate range at low frequencies, but the bandwidth of the system was too high and did not satisfy the robustness requirements.

In fact the requirements were not met even with several weights which were used, an example of which is depicted in Fig. 4.5. The weighting was the discretized equivalent of $\frac{1}{s+1}$. Compared to Fig. 4.3, Fig. 4.5 demonstrates the effect of the higher weighting at low frequency. The low frequency sensitivity has been decreased at the expense of the high frequency sensitivity as expected.

After using several weighting functions, it was determined that the requirements could not be met (or would be difficult to meet) using only the sensitivity function. The next step was to use the complementary sensitivity $\mathbf{GK}(\mathbf{I}-\mathbf{GK})^{-1}$ in the objective function. The sum of the sensitivity and the complementary sensitivity was used as the objective.

The results of the new optimization are shown in Figs. 4.6-4.8. The low frequency sensitivity has increased slightly and $\mathbf{GK}$ crosses the 0 db point at about 40 rad/sec. This new system meets the design specs. Various weights were used to try to tune the system better, but none were found which produced

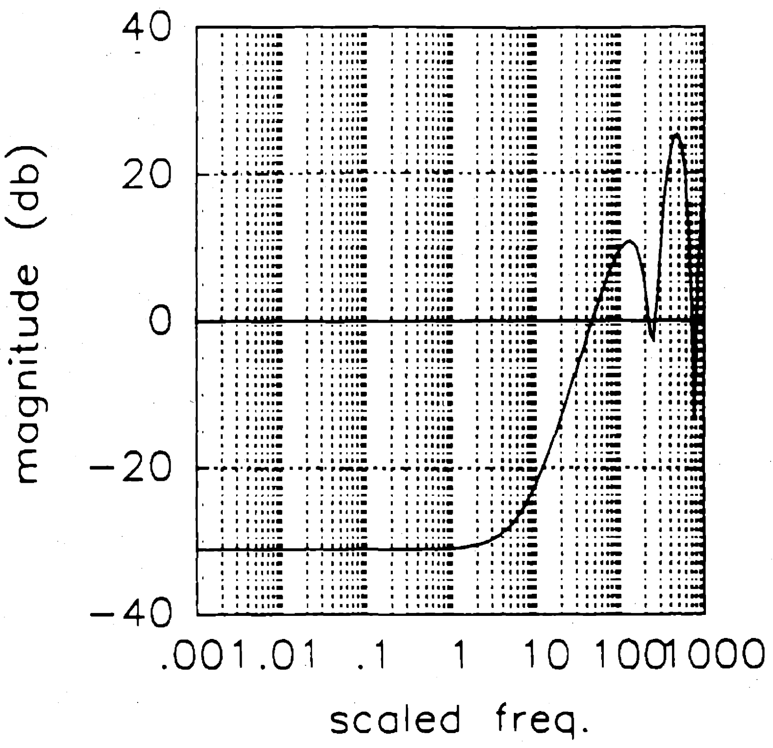


Figure 4.3: Sensitivity for Unweighted Optimization

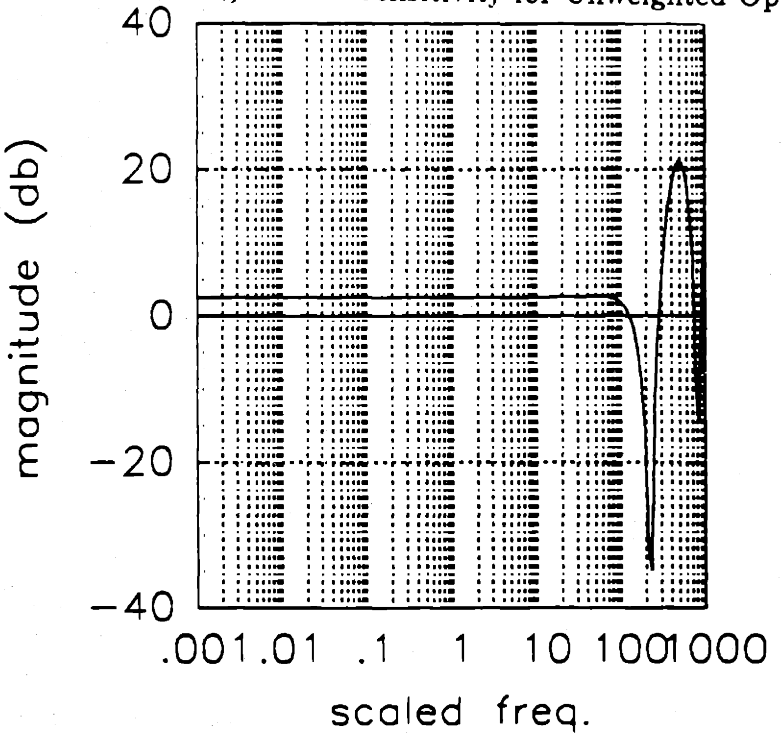


Figure 4.4: $GK(I - GK)^{-1}$ for Unweighted Optimization

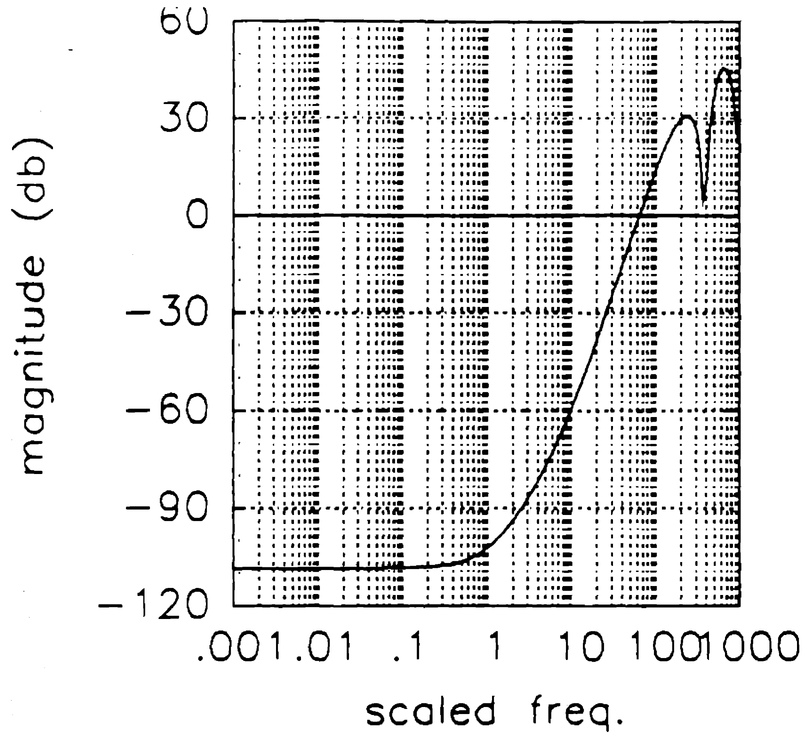


Figure 4.5: Optimal Sensitivity With Weight $\frac{1}{s+1}$

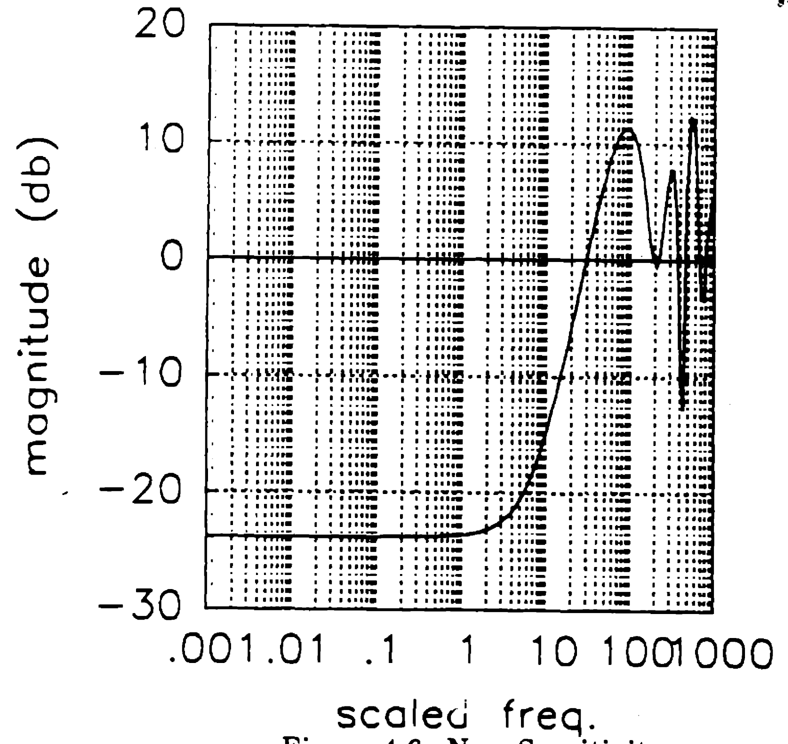


Figure 4.6: New Sensitivity

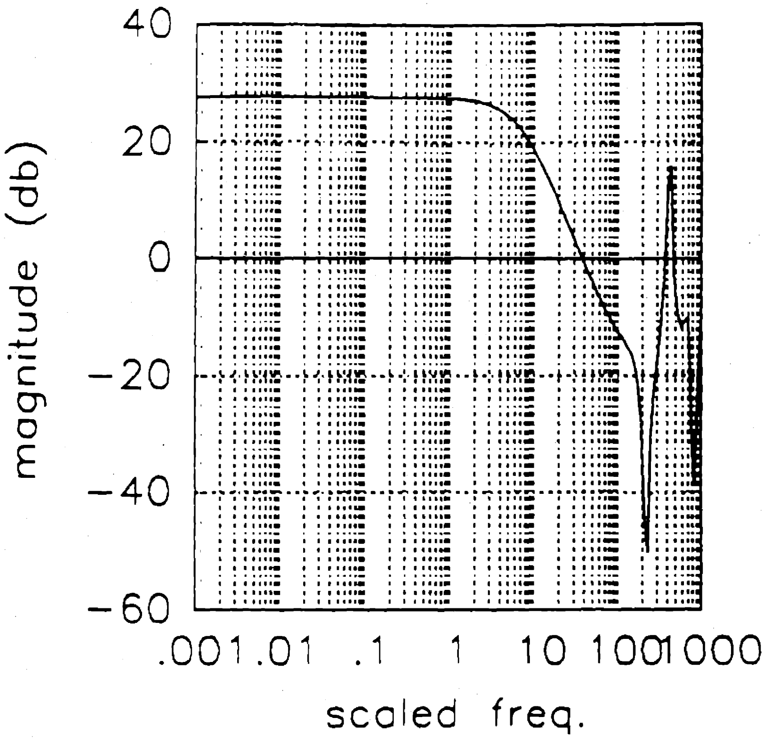


Figure 4.7: GK

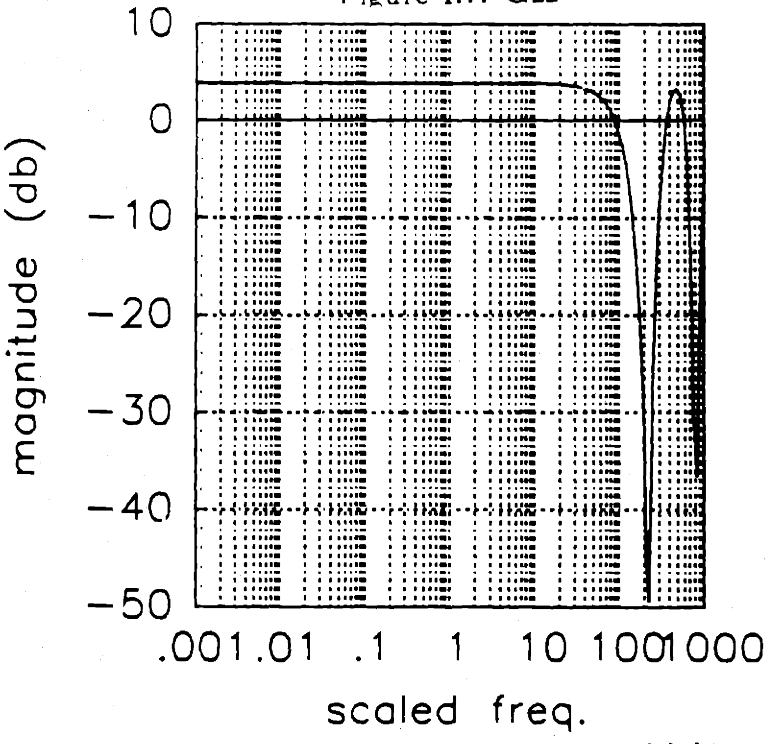


Figure 4.8: Complelentary Sensitivity

much improved results.

For a step command, the system here has about 6% error, which is slightly better than the LQG/LTR design of 4. For other signals, however, it is not clear what the outcome will be since the two designs use very different methods of arriving at a controller.

Chapter 5

Summary and Conclusions

5.1 Conclusions

Software has been implemented to solve the l^1-optimization in the context of the l^1-optimal control problem. This software has been applied to two real control problems — the attitude control and momentum management of a space station and the pitch and angle of attack control of the X29. Although the design examples did not work out as well as was hoped, several interesting and important observations can be made.

First of all, l^1-optimal solutions do not in general lend themselves well to calculation using high level software like MATRIXx. There are many special cases which need to be checked for and this is not convenient in MATRIXx.

The first area of difficulty is the very first step of determining the closed loop parametrization. For fully controllable and observable systems, it is possible to place all poles at the origin, which is preferred because it is easier to check such a system for correctness. In some situation it is necessary, or simply desirable, to solve the regulator and estimator problems to obtain the parametrization. Currently, the user must choose at run-time.

A more serious problem is the transformation of the problem back and forth from state space and time domain. Calculating in state space is for the most part more robust than using the transformed polynomials or time sequences. This is a factor in the calculation of system derivatives and relations. The derivatives are the main problem as the largest errors seem to occur there. A consistent method of calculating the derivatives would be a significant addition to the reliability of the algorithm.

Another significant improvement would be a means of eliminating the redundant constraints of the LP before attempting a solution. This would first of all decrease the size of the problem, and in addition it would help the reliability of the LP. In many of the large examples attempted, the constraint matrix suffered from serious ill-conditioning when certain constraints were added during calculation.

A final area where improvement can be made is in the calculation of the optimal controller from the optimal closed loop solution. Much of the problem here is once again the transformation from time domain to state space and back. Errors incurred in these transformations move around the poles and zeros sufficiently that cancellations are not correctly carried out at later steps. This leaves many near cancellations in the final system. The system ends up not only of higher order than it should have, but in many cases the calculated controller is not even stabilizing as a result of the uncancelled poles and zeros.

It should be noted that a very good LP solver is required to obtain any results from the l^1 optimization in many cases. The NAG subroutine E04NAF was used for this study and was limited to systems of less than about 150 variables and 100 constraints.

As far as actual execution, the code works very well for square systems and for systems which transformed easily from state space to polynomial representations. Systems which did not transform easily led to bad relations and infeasible LP's. Also, systems for which the poles could be placed at the origin with reasonable ("reasonable" being somewhat ill-defined) also worked very well with the software.

For the space station, pitch axis control design was successfully carried out. This design encorporated rejection of sinusoids at orbital frequency and twice orbital frequency. Within this constraint, the optimization was done on the combined weighted and unweighted impulse responses of the pitch. Including the unweighted impulse response reduced the peak magnitude of the disturbance response by an order of magnitude compared to the case where it was not included.

Roll and yaw axis control design was done without the cyclic disturbance rejection. This design was kept simple due to limitations of the current software for solving the problem and the inherent numerical difficulty of dealing with the space station system.

It is expected that this design would possess properties superior to a simple LQR design with a cyclic disturbance rejection filter as in [5]. The controller, designed for a linearized model of a non-linear system, would actually operate in the non-linear environment. The linear model is only valid in a small region around the point of linearization. The l^1controller is designed to minimize arbitrary bounded disturbances as well as providing cyclic disturbance rejection. This should give it superior performance in the non-linear environment.

This assumption could not be tested, however, because the controllers could not be extracted from the problem solution in the roll/yaw case. If the controllers were calculated, a simulation of the full non-linear model could be carried out. This would make an interesting continuation of the present work.

The final design example was for the angle of attack control for the X29 forward swept wing aircraft. This design was required to meet certain performance specifications. Using the l^1 optimal design, the specifications were met with relatively little design effort. Although the design specs were met with the illustrated design, it may be possible to further tune the system with more complex weights and objective functions.

5.2 Summary and Future Direction

The overall project of software development and meaningful control design was in the end a little overambitious. By the end of the project, the software was more under control in that the problem areas had been identified and could be worked around somewhat. The original simple translation of the theory into computer code did not work very well and underwent much revision to get even to the point that it is at now.

More effort needs to go into the software in order to deal with problems of practical dimension and complexity, particularly in the areas noted previously. Currently the success or failure is very problem dependent, though small changes to the right place in the code can sometimes allow a particular problem to be dealt with.

Finally, based on the results presented here, further investigation into the l^1-optimal control of the space station appears worthwhile. Particularly of interest would be to derive the optimal controllers and to carry out simulation of the full non-linear model with these linear system controllers. It is felt that the l^1-optimal controllers would possess good robust performance properties.

Appendix A
Computer Code

This appendix contains the details of the software written for this thesis. The first section is a block diagram showing the functional blocks of the program and a table of the routines associated with each block. The second section contains actual listings of the routines.

A.1 Flow Diagram

Fig. A.1 shows the basic outline of the l^1 algorithm. Notice that there is no logic in this sequence. All the logic is contained in the appropriate subsections of the code.

Table A.1 lists the names of the routines used in the program. The controlling routine, LONE.UDF, is not listed. The function of LONE.UDF is simply to call the other routines in the appropriate sequence.

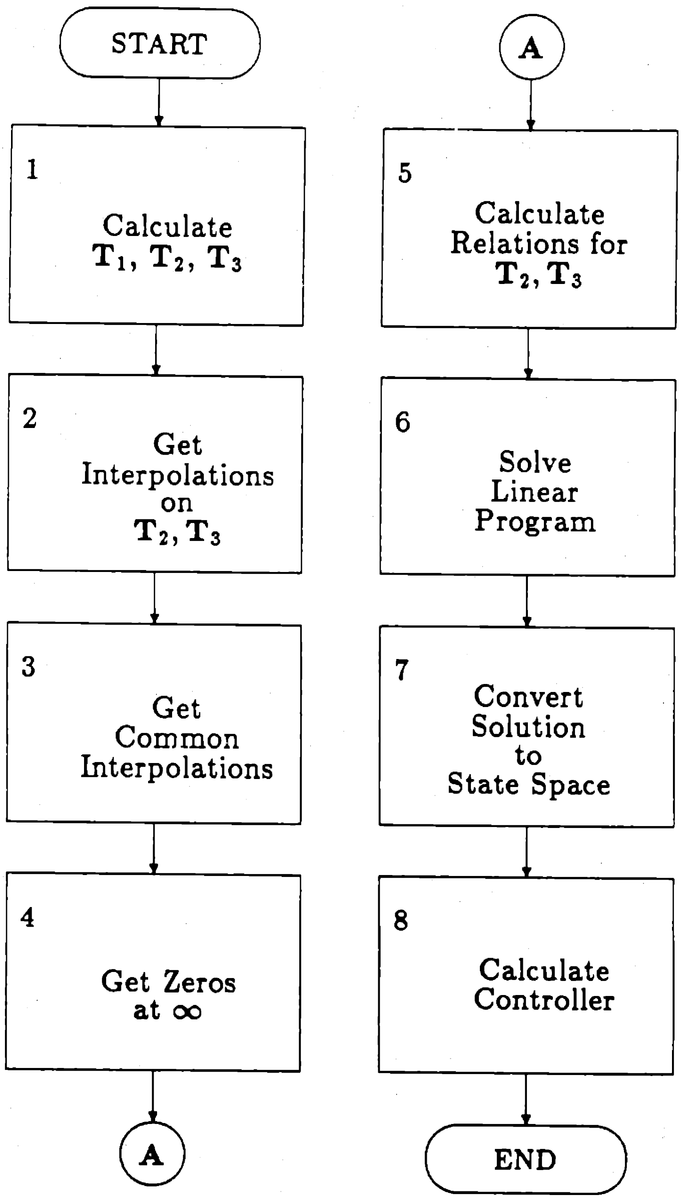


Figure A.1: Block Flow Diagram

Section	*Main Routine*	*Auxiliary Routines*
1	HUV.UDF	
2	ZERCON.UDF	GETSVD.UDF
		GETRHS.UDF
3	GETCMN.UDF	
4	ZINF.UDF	RELDEG.UDF
5	RELAT.UDF	SEQDET.UDF
6	USER.FOR	SIMPLX.FOR
7	CLSS.UDF	BLDK.UDF
8	KCALC.UDF	COPRIME.UDF
		KSS.UDF
		PTOSS.UDF
Miscellaneous Routines		
	CHECK.UDF	FACT.UDF
	SSADD.UDF	SSMUL.UDF
	SSMUL.UDF	

Table A.1: List of Subroutines

A.2 Code Listings

Following are listings of the complete set of routines used in the l^1 algorithm. Each contains a brief description of its function and comments in the code itself which have been added for readability. Comments marked by %'s are not part of the code itself.

LONE.UDF

Routine LONE.UDF is the main routine which calls the functional routines in the correct sequence.

```
//[SCL,NSCL,SKOPT,NSK]=LONE(S,NS,IW,IZ)
//
// CALCULATE THE L1 OPTIMAL CONTROLLER
//
//  begin code here ...
//    GET H,U,V for G = H - U*Q*V
//
//  Choose pole placement or
//    regulator/estimator
//
display(' Enter [1] for pole placement')
inquire i 'or other to enter gains.'
if i=1, [t1,t2,t3]=huv(s,ns,iw,iz); ...
else    [t1,t2,t3]=huvtmp(s,ns,iw,iz); end
//
//     HOW LONG IS PHI?
//
tot=0;l1=0;l2=0;
x=size(t2);
diff=x(1)-x(2);
```

```
if diff>0, l1=diff*ns+1; end
x=size(t3);
diff=x(2)-x(1);
if diff>0, l2=diff*ns+1; end
//
//   ASK FOR LENPHI
//
display('default lenphi is:')
lenphi=max([l1+ns-1 l2+ns-1 2*ns+1])
inquire x 'enter lenphi (0 for default):'
if x<>0, lenphi=x; end
clear x diff
//
//    SET UP LP, SOLVE LP, AND GET Q,K
//
[row,col]=size(t1);
row=row-2*ns;
col=col-2*ns;
nphi=lenphi*row*col;
//
//  1) sum|(xij(k)+yij(k)| over j,k <= OPTNORM for all i
//
tot=user([row col 2:nphi],1,0)
//
//  2) constraints of form uzvec(a,i)*H(a)=uzvec(a,i)*PHI(a)
//        for each zero 'a' and associated vectors 1 to i.
//        H(b)*vzvec(b,i)=PHI(a)*vzvec(a,i).
//
// ***  get left zeros of U  ***
//
display('*** INTERPOLATIONS ON T2 ***')
[nuz,uz,uzvec,tot]=zercon(t1,t2,ns,1,lenphi,tot);
tot
//
// ***  get right zeros of V  ***
//
display('*** INTERPOLATIONS ON T3 ***')
[nvz,vz,vzvec,tot]=zercon(t1,t3,ns,2,lenphi,tot);
tot
//
```

```
//  Take care of common zeros of T2,T3
//
if nuz>0, if nvz>0, ...
 tot = ...
  getcmn(t1,ns,[nuz nvz],[uz;vz],[lenphi tot],uzvec,vzvec), ...
end, end
clear nuz uz uzvec nvz vz vzvec
//
//  3) constraints of form lefvec*H = lefvec*PHI.
//        H*rigvec = PHI*rigvec.
//
display('*** RELATIONS ON T2 ***')
tot=relat(t1,t2,ns,lenphi,1,tot)
//
display('*** RELATIONS ON T3 ***')
tot=relat(t1,t3,ns,lenphi,2,tot)
//
//  4) Take care of zeros at infinity
//
tot=zinf(t1,t2,t3,ns,lenphi,tot)
//
//  *** GET LINEAR PROGRAM OPTIMAL SOLUTION ***
//
display('*** SOLVING LINEAR PROGRAM ***')
phi=user(0*ones(1,nphi+1),2,tot)
//
//  *** GET STATE SPACE CLOSED LOOP XFER FUNCTION ***
//
[SCL,NSCL]=clss(phi,lenphi,row,col);
clear phi
retf
//
  %%  Code after this point does is not useful %%
  %%    for any meaningful problems            %%
//
//  *** GET STATE SPACE (T1-PHI) ***
//
display(' -- (T1-PHI) --')
//[SKOPT,NSK]=kss(t1,ns,phi,lenphi,nphi);
[SKOPT,NSK]=ssadd(t1,2*ns,scl,nscl,-1);
```

```
[phi,np]=minimal(skopt,nsk);
if np<nsk, SKOPT=phi; nsk=np; end
clear phi np
NSK
//
//   *** FINISH CALCULATING CONTROLLER **
//
display(' -- inv(T2)(T1-PHI) --')
[t2inv,iadv]=ssinv(t2,ns);
[SKOPT,NSK]=ssmul(t2inv,ns,SKOPT,NSK);
[row,col]=size(SKOPT);
row=row-NSK; col=col-NSK;
if iadv>0, ...
  [num,den]=tform(SKOPT,NSK); ...
  [i,inum]=size(num); ...
  inum=inum/col; ...
  if iadv>(NSK+1-inum), ...
    for i=(nsk+2-inum):iadv, ...
      if max(abs(num(:,[1:col])))>1e-15, ...
         display('-- BIG TROUBLE --'), end, ...
      [i1,i2]=size(num); ...
      num=num(:,[col+1:i2]); ...
  end, end, ...
  for i=1:iadv, ...
    num=[num 0*ones(row,col)]; end, ...
  [SKOPT,NSK]=sform(num,den,col); ...
  NSK; ...
end
[SKOPT,NSK]=minimal(SKOPT,NSK);
//
if NSK>0, ...
  display(' -- Q --'), ...
  [t3inv,iadv]=ssinv(t3,ns); ...
  [SKOPT,NSK]=ssmul(SKOPT,NSK,t3inv,ns); ...
  [row,col]=size(SKOPT); ...
  row=row-NSK; col=col-NSK; ...
  if iadv>0, ...
    [num,den]=tform(SKOPT,NSK); ...
    [i,inum]=size(num); ...
    inum=inum/col; ...
    if iadv>(NSK+1-inum), ...
      for i=(nsk+2-inum):iadv, ...
        if max(abs(num(:,[1:col])))>1e-15, ...
           display('-- BIG TROUBLE --'), end, ...
        [i1,i2]=size(num); ...
        num=num(:,[col+1:i2]); ...
    end, end, ...
    for i=1:iadv, ...
      num=[num 0*ones(row,col)]; end, ...
    [SKOPT,NSK]=ptoss(num,den,col); ...
  end, ...
  [SKOPT,NSK,t]=minimal(SKOPT,NSK); ...
end
if NSK>0, ...
  ei=eig(skopt([1:nsk],[1:nsk])); ...
  ei=diag(ei); ...
  for i=1:nsk, if norm(ei(i))>=1, ...
    display('  Warning: Q not stable.'), ...
  end, end, ...
end
//
[m,n]=size(s);
if NSK=0, SKOPT=0; end
[SKOPT,NSK] = ...
  kcalc(SKOPT,NSK,s([1:ns ns+iz+1:m],[1:ns ns+iw+1:n]),ns);
//
retf
```

HUV.UDF

This routine determines the Q parametrization of the closed loop system. Gains are chosen to place all poles at the origin. An alternate routine, HUVTMP.UDF, allows the input of weights and then solves the corresponding regulator and estimator problems. HUVTMP must be used if the poles cannot be placed arbitrarily and should also be used if excessive gains are needed to place the poles at the origin.

```
//[T1,T2,T3]=huv(S,NS,IW,IZ)
//
//   CALCULATE T1,T2,T3 (aka H,U,V)
//
[a,b,c,d]=split(s,ns);
[i,m1]=size(b);
  b1=b(:,[1:iw]); b2=b(:,[iw+1:m1]);
[n2,i]=size(c);
  c1=c([1:iz],:); c2=c([iz+1:n2],:);
  d11=d([1:iz],[1:iw]); d12=d([1:iz],[iw+1:m1]);
  d21=d([iz+1:n2],[1:iw]); d22=d([iz+1:n2],[iw+1:m1]);
//
//  TRY PLACING POLES AT ORIGIN (SO MATRIX WILL
//   BE POLYNOMIAL)
//
p=real(poly(a)); for i=1:ns, pm(i,:)=p'; end
bp=0*ones(ns,1); for i=iw+1:m1, bp=bp+b(:,i); end
sc=bp; scc=eye(ns);
for i=2:ns, ...
  sc(:,i) = ...
   a*sc(:,(i-1)); scc=scc+diag(pm([1:ns+1-i],i),(i-1)); end
t=sc*scc; tinv=inv(t);
f1=p(2:ns+1)'*tinv,
for i=iw+1:m1, f(i-iw,:)=f1; end
evcnt=eig(a+b2*f)
```

```
//
cp=0*ones(1,ns); for i=iz+1:n2, cp=cp+c(i,:); end
sc=cp'; scc=eye(ns);
for i=2:ns, ...
  sc(:,i) = ...
   a'*sc(:,(i-1)); scc=scc+diag(pm([1:ns+1-i],i),(i-1)); end
t=sc*scc; tinv=inv(t);
h1=tinv'*p(2:ns+1),
for i=iz+1:n2, h(:,i-iz)=h1; end
evobs=eig(a+h*c2)
  %%%
  %%% The following section is part of HUVTMP.UDF
  %%%  and replaces the preceding pole placement
  %%%  algorithm in that routine.
  %%%
  %%%
  %%% //  ASK FOR WEIGHTS AND SOLVE REGULATOR AND
  %%% //   ESTIMATOR PROBLEMS.
  %%% //
  %%% for i=1:ns, ...
  %%%   inquire f 'enter state weight:', ...
  %%%   h(i)=f; ...
  %%% end
  %%% for i=1:m1-iw, ...
  %%%   inquire f 'enter input weight:', ...
  %%%   g(i)=f; ...
  %%% end
  %%% [e,f]=dregulator(a,b2,diag(h),diag(g));
  %%% f=-f
  %%% evcnt=eig(a+b2*f)
  %%% //
  %%% for i=1:ns, ...
  %%%   inquire e 'enter state weight:', ...
  %%%   h(i)=e; ...
  %%% end
  %%% for i=1:n2-iz, ...
  %%%   inquire e 'enter output weight:', ...
  %%%   g(i)=e; ...
  %%% end
  %%% [e,h]=destimator(a,c2,diag(h),diag(g));
  %%% h=-h
  %%% evobs=eig(a+h*c2)
  %%%
//
//  CALCULATE H,U,V for G = T1 - T2*Q*T3
//
af=a+b2*f;
ah=a+h*c2;
t1=[[af -b2*f ; 0*a ah] [b1 ; b1+h*d21]
    c1+d12*f  -d12*f      d11];
t2=[af b2 ; c1+d12*f d12];
t3=[ah b1+h*d21 ; c2 d21];
retf
```

ZERCON.UDF

```
//[NZ,ZO,ZVEC,NCON]=ZERCON(S1,S2,NS,IFL,LENPHI,NCON)
//
// GET THE ZEROS OF A MATRIX
//
nz=0;
if ifl=1, s2=s2'; end
[a,b,c,d]=split(s2,ns);
[nrow,i]=size(c); [i,ncol]=size(b);
eval=eig(s2([1:ns],[1:ns]));
clear s2
if ncol<nrow, ...
  z1=zeros([a b;c d],ns); ...
  if exist('z1')=0; ...
    zo=0; zvec=0; retf, ...
end
[row,col]=size(s1);
row=row-2*ns;
col=col-2*ns;
nphi=lenphi*row*col;
if ifl=1, n1=col; n2=row; ...
else      n1=row; n2=col; end
//
//   Get zeros and associated vectors.
//
tol=1d-10;
ncns=ncol+ns;
nrns=nrow+ns;
nz1=0;
n=ncol/nrow;
if nrow=ncol, n=1; end
if nrow>ncol, n=0; end
for i=1:n+1, ...
  if n>0, i1 = (i-1)*nrow+1; in = i*nrow; end, ...
  if i=(n+1), ...
    if ncol>(i-1)*nrow, i1=(i-1)*nrow+1; in=ncol; ...
    else i1=0; in=0; end, end, ...
  if i1>0, ...
if nrow=1, ...
  z1=zeros([a b(:,[i1:in]); c d(:,[i1:in])]',ns); ...
else, ...
  z1=zeros([a b(:,[i1,in]); c d(:,[i1,in])],ns); ...
end, ...
if exist('z1')<>0, ...
  [m,j]=size(z1); ...
  z1=[z1;100]; ii=0; ...
  for j=1:m, ...
    x1=z1(j); x2=z1(j+1); ...
    if abs(imag(x1))<1d-15, x1=real(x1); end, ...
    z1(j)=x1; ...
  end, ...
  clear zblk, ...
  for j=1:m, ...
    x1=z1(j); x2=z1(j+1); ...
    for k=1:ns, ...
      if check(x1,eval(i),2,1e-6)=1, x1=0; end, ...
    end, ...
    if check(x1,1,4,1e-6)=1, if abs(imag(x1))<5e-4, ...
      x1=real(x1)/abs(real(x1)); end, end, ...
    if check(x2,1,4,1e-6)=1, if abs(imag(x2))<5e-4, ...
      x2=real(x2)/abs(real(x2)); end, end, ...
    if norm(x1)>=(1-tol), ...
      if ii=0, ...
        if check(x1,x2,1,tol)=0, ...
          ii=1; zblk(ii,:)=[x1 1 0]; end, ...
      else, ...
        nz=1; ...
        for k=1:ii, ...
          if check(x1,zblk(k,1),2,tol)=1, ...
            nz=zblk(k,2)+1; end, ...
        end, ...
        if check(x1,x2,1,tol)=0, ...
          ii=ii+1;zblk(ii,:)=[x1 nz 0]; ...
    end, end, end, ...
  end, ...
  m=ii; ...
  for j=1:m, ...
    z=zblk(j); ...
```

```
[sv,v] = ...
 getsvd([a b(:,[i1:in]);c d(:,[i1:in])],ns,zblk(j,:));...
kr=0; ...
ndim=max(size(v)); ...
for ii=1:ndim, if sv(ii,ii)>1e-12, kr=kr+1; end, end, ...
if kr=0, v=eye(ndim); end, ...
for jj=(kr+1):ndim, ...
  tmp=0*ones(ncol,1); ...
  tmp(i1:in)=v(:,jj); ...
  tmp=tmp/max(abs(tmp)); ...
  for ii=1:ncol, ...
    if norm(tmp(ii))<1e-15, tmp(ii)=0; end, ...
    if abs(imag(tmp(ii)))>0, ...
      l=abs(real(tmp(ii))/imag(tmp(ii))); ...
      if l>1e10, tmp(ii)=real(tmp(ii)); end, ...
      if l<1e-10, tmp(ii)=imag(tmp(ii))*jay; end, ...
  end, end, ...
  ii=zblk(j,2); ...
  rhs=getrhs(s1,2*ns,tmp,z,ii,ifl); ...
  for l=1:ii, tmp1(l)=0; end, tmp1(ii)=fact(1,ii-1); ...
  for l=ii+1:lenphi, ...
    tmp1(l)=fact(l-ii+1,l-1)*(z**(ii-l)); end, ...
  for kk=1:n1, ...
    if abs(rhs(kk))<1e-15, rhs(kk)=0; end, end, ...
  for kk=1:n1, ...
    const=0*ones(1,nphi); ...
    for k=1:n2, ...
      if ifl=1, off=(k-1)*lenphi*col+1+(kk-1)*lenphi; ...
      else off=(kk-1)*lenphi+1+(k-1)*lenphi; end, ...
      const(1,[off:off-1+lenphi])=conj(tmp1')*tmp(k); ...
    end, ...
    [const rhs(kk)], ...
    ncon=user(real([const rhs(kk)]),0,ncon); ...
    if imag(z)<>0, ...
      ncon=user(imag([const rhs(kk)]),0,ncon); ...
    end, ...
  end, ...
  nz1=nz1+1; zvec(:,nz1)=tmp; ...
  zblk(j,3)=zblk(j,3)+1; ...
end, ...
```

```
      end, ...
      if exist('zblk')<>0, nz=nz+m; ...
        if exist('zo')=0, zo=zblk; ...
        else, zo=[zo;zblk]; end, end, ...
  end, end, ...
end
if exist('zvec')=0, zvec=0; end
if ifl=1, zvec=conj(zvec');
if exist('zo')=0, zo=0; end
if nz=0, display('NO UNSTABLE ZEROS'), end, end
retf
```

GETRHS.UDF

```
// RHS = GETRHS(S,NS,ZVEC,ZRO,DER,F)
//
//  RETURN THE RIGHT HAND SIDE FOR INTERPOLATION EQUALITIES
//
[row,col]=size(s);
row=row-ns; col=col-ns;
if der=1, ...
  [a,b,c,d]=split(s,ns); ...
  rhs=c*inv(zro*eye(a)-a)*b+d; ...
 %%%
 %%%  This section calculates derivatives by
 %%%   using the pulse response.  This cannot be
 %%%   used if the pulse reponse does not decay
 %%%   fast enough.
 %%%
else, rhs=0*ones(row,col); ...
  for j=1:row, for i=1:col, ...
    [t,y]=pulse(s([1:ns ns+j],[1:ns ns+i]),ns,5*ns); ...
    for k=der:5*ns, ...
      rhs(j,i) = ...
       rhs(j,i)+y(k)*fact(k-der+1,k-1)*zro**(der-k); ...
    end, ...
  end, end, ...
end
 %%%
 %%%  An alternate method is the following which
 %%%   uses the numerator and denominator of the
 %%%   transfer function.  IT IS ONLY CAPABLE OF
 %%%   CALCULATING FIRST DERIVATIVES however.
 %%%
\\else, rhs=0*ones(row,col); if der=2, ...
\\  for j=1:row, ...
\\    for i=1:col, ...
\\      [num,den]=tform(s([1:ns ns+j],[1:ns ns+i]),ns); ...
\\       d=0; dd=0; n=0; dn=0; ...
\\       for k=1:ns+1, ...
\\         d=d+den(k)*zro**(1-k); ...
\\          dd=dd+den(k)*(k-1)*zro**(2-k); ...
\\        end, ...
\\        [k,i1]=size(num); ...
\\        for k=1:i1, ...
\\          n=n+num(k)*zro**(i1-ns-k); ...
\\          dn=dn+num(k)*(ns-i1+k)*zro**(i1-ns+1-k); ...
\\        end, ...
\\        rhs(j,i)=(d*dn-n*dd)/d/d; ...
\\   end, end, ...
\\   else, display(' *** Three zeros? ***'), end, ...
\\end
if f=1, rhs=conj(zvec')*rhs; ...
else,   rhs=rhs*zvec; end
retf
```

GETSVD.UDF

```
//[SV,V]=GETSVD(S,NS,Z)
//
//  GET ZERO VECTORS FOR G(S) AT Z
//
zro=z(1);
der=z(2)-1;
clear z
[nrow,ncol]=size(s);
if nrow=ns+1, sv=0; v=1; retf
sv=eye(s)*0;
for i=1:ns, sv(i,i)=1; end
nrns=nrow;
nrow=nrow-ns; ncol=ncol-ns;
if der=0, ...
  [u,sv,v]=svd(zro*sv-s); ...
  dmin=min(size(sv)); ...
  sv=sv([ns+1:dmin],[ns+1:dmin]); ...
  v=v([ns+1:dmin],[ns+1:dmin]); ...
else, ...
  [t,y]=pulse(s,ns,30*ns); ...
  ans=0*ones(nrow,ncol); ...
  for i=der+1:30*ns, ...
    f1=fact(i-der,i-1); ...
    for j=1:nrow, ...
      for k=1:ncol, ...
        ans(j,k) = ...
         ans(j,k)+f1*y(i,(j-1)*ncol+k)*(zro**(der+1-i)); ...
  end, end, end, ...
  [u,sv,v]=svd(ans); ...
end
retf
```

GETCMN.UDF

```
//NCON=GETCMN(S,NS,NZ,ZZ,INFO,VEC1,VEC2)
//
// COMMON ZEROS
//
display('*** COMMON ZEROS OF T2, T3 ***')
nuz=nz(1); nvz=nz(2); clear nz
lenphi=info(1); ncon=info(2); clear info
uz=zz([1:nuz],1); vz=zz([nuz+1:nuz+nvz],1);
uinfo=zz([1:nuz],[2 3]); vinfo=zz([nuz+1:nuz+nvz],[2 3]);
uinfo=[uinfo;-99 0]; vinfo=[vinfo;-99 0];
[row,col]=size(s); row=row-2*ns; col=col-2*ns;
nphi=lenphi*row*col;
totu=0;
for i=1:nuz, ...
  common(i,:)=[0 0 0]; ...
  if uinfo(i,1)>=uinfo(i+1,1), ...
    totv=0; ...
    for j=1:nvz, ...
      if check(uz(i),vz(j),2,1e-10)=1, ...
        if vinfo(j)>=vinfo(j+1), ...
          common(i,:)=[j totu totv]; end, ...
      end, ...
      totv=totv+vinfo(j,2); end, ...
  end, ...
  totu=totu+uinfo(i,2); ...
end
for i=1:nuz, ...
  no=common(i,1); ...
  if no<>0, ...
    uu=common(i,2); vv=common(i,3); ...
    start=max([uinfo(i,1) vinfo(no,1)]); ...
    for j=start+1:uinfo(i,1)+vinfo(no,1), ...
      for k=1:j, tmp(k)=0; end, tmp(j)=fact(1,j-1); ...
      for k=j+1:lenphi, ...
        tmp(k)=fact(k-j+1,k-1)*uz(i)**(j-k); end, ...
      for k=1:uinfo(i,2), ...
        rhs=getrhs(s,2*ns,vec1(uu+k,:),uz(i),j,1); ...
        for l=1:vinfo(no,2), ...
          rhs1=rhs*vec2(:,vv+l); ...
          const=0*ones(1,nphi); ...
          for aa=1:row, ...
            for bb=1:col, ...
              off=(aa-1)*lenphi*col+(bb-1)*lenphi+1; ...
           const(1,[off:off+lenphi-1]) = ...
              conj(tmp')*vec1(uu+k,aa)*vec2(bb,vv+l); ...
          end, end, ...
          [const rhs1], ...
          ncon=user(real([const rhs1]),0,ncon), ...
          if imag(uz(i))<>0, ...
            ncon=user(imag([const rhs1]),0,ncon), ...
end,end,end,end,end
retf
```

ZINF.UDF

```
//NCON=ZINF(S1,S2,S3,NS,LENPHI,NCON)
//
//  CALCULATE ZEROS AT INFINITY
//
[a,b,c,d]=split(s1,2*ns);
[row,i]=size(c);
[i,col]=size(b);
nphi=lenphi*row*col;
display('*** ZEROS AT INFINITY ***')
for i=1:row, ...
  rdt2(i)=reldeg(s2([1:ns ns+i],:)); ...
end
for j=1:col, ...
  rdt3(j)=reldeg(s3(:,[1:ns ns+j])'); ...
end
for i=1:row, ...
  off=(i-1)*lenphi*col; ...
  for j=1:col, ...
  rdtot=rdt2(i)+rdt3(j); ...
    for k=1:rdtot, ...
      tmp=0*ones(1,nphi); ...
      [xx,zz]=pulse([a b(:,j);c(i,:) d(i,j)],2*ns,rdtot); ...
      tmp((j-1)*lenphi+k+off)=1; ...
      ncon=user([tmp zz(k)],3,ncon); ...
    end, ...
end, end
retf
```

RELDEG.UDF

```
//[RD]=RELDEG(S)
//
//  Get maximum relative degree (demon)-(num) of a row
//     of a matrixx of polynomial fractions
//
[m,n]=size(s);
ns=m-1;
n=n-ns;
rd=ns;
for i=1:n, ...
  z=zeros(s(:,[1:ns ns+i]),ns); ...
  if exist('z')=1, [m,p]=size(z); ...
  else, m=0; end, ...
  rd=min(rd,ns-m); end
retf
```

RELAT.UDF

```
// NCON=RELAT(S1,S2,NS,LENPHI,IFL,NCON)
//
//  CALCULATE RELATIONS
//
[nrow,ncol]=size(s2);
nrow=nrow-ns;
ncol=ncol-ns;
n1=nrow; n2=ncol;
if ifl=1, n1=ncol; n2=nrow; end
if n1>=n2, display('NO RELATIONS'), retf
nm=n1;
[a,b,c,d]=split(s2,ns);
mult=poly(a);
md=0;
if max(abs(d))>0, md=1; end
n=ns+md;
z=0*ones(n1,n2*n);
for i=1:nrow, ...
  for j=1:ncol, ..
    [seq,tmp]=tform([a b(:,j);c(i,:) d(i,j)],ns); clear tmp; ...
    [jj,ii]=size(seq); ...
    if ii<ns+1, for jj=ii:ns, seq=[0 seq]; end, end, ...
    if ifl=1, i1=j; i2=i; else, i1=i; i2=j; end, ...
    if j>nm, z(i1,[(i2-1)*n+1:i2*n])=-seq(2-md:ns+1); ...
    else, if i>nm, z(i1,[(i2-1)*n+1:i2*n])=-seq(2-md:ns+1); ...
          else, z(i1,[(i2-1)*n+1:i2*n])=seq(2-md:ns+1); ...
end, end, end, end
z=real(z);
nuldim=n2-n1;
totlen=(n-1)*n1+1;
vec=0*ones(n2,nuldim*totlen);
ind0=n*n1;
vec1=seqdet(z(:,[1:ind0]),n); ...
for i=1:nuldim, ...
  n1i=n1+i; ...
  ind1=(i-1)*totlen+1; ...
  ind2=i*totlen; ...
```

```
    ind3=(n1i-1)*n+1; ...
    ind4=n1i*n; ...
    vec(n1i,[ind1:ind2])=vec1; ...
    if n1=1, vec(1,[ind1:ind2])=z(1,[ind3:ind4]); ...
    else, vec(1,[ind1:ind2]) = ..
            seqdet(z(:,[ind3:ind4 n+1:ind0]),n); end, ...
    if n1>2, ...
      for j=1:(n1-2), ...
        ind5=j*n; ...
        ind6=(j+1)*n+1; ...
        vec(j+1,[ind1:ind2]) = ...
           seqdet(z(:,[1:ind5 ind3:ind4 ind6:ind0]),n); ...
      end, ...
      vec(n1,[ind1:ind2]) = ...
        seqdet(z(:,[1:(ind0-n) ind3:ind4]),n); ...
   end, ...
end
clear z vec1 nrow ncol
n=totlen;
nrhs=lenphi+totlen;
[a,b,c,d]=split(s1,2*ns);
[row,i]=size(c); [i,col]=size(b);
n1=row; n2=col;
nphi=lenphi*row*col;
if ifl=1, n1=col; n2=row; end
for ii=1:nuldim, ...
  for i=1:n1, ...
    tmp1=0*ones(1,nrhs+n-1); ...
    for j=1:n2, ...
      tmp=vec(j,[(ii-1)*n+1:ii*n]); ...
      if ifl=2, ...
        [xx,zz]=pulse([a b(:,j);c(i,:) d(i,j)],2*ns,nrhs); ...
      else, ...
        [xx,zz]=pulse([a b(:,i);c(j,:) d(j,i)],2*ns,nrhs); ...
      end, ...
      tmp1=convolve(tmp,zz)+tmp1; ...
    end, ...
    for jj=1:nrhs+n-1, ...
      if abs(tmp1(jj))<1e-15, tmp1(jj)=0; end, end, ...
    for jj=1:n+lenphi-1, ...
```

```
      tmp=0*ones(1,nphi); ...
      for j=1:n2, ...
        if ifl=1, i1=j; i2=i; else, i1=i; i2=j; end, ...
        off=(i1-1)*lenphi*col+(i2-1)*lenphi; ...
        nn=min(n,jj); ...
        mm=max(1,jj+1-lenphi); ...
        for kk=mm:nn, ...
          tmp(jj-kk+1+off)=vec(j,(ii-1)*n+kk); end, ...
      end, ...
      if max(abs(tmp))>1e-15, ...
        ncon=user([tmp tmp1(jj)],0,ncon); end, ...
    end, ...
  end, ...
end
retf
```

SEQDET.UDF

```
//[A]=seqdet(Z,N)
//  get determinant of z, with n=length of each sequence
//
[row,all]=size(z);
col=all/n;
last=all-n;
if row<>col, display('ERROR - MATRIX NOT SQUARE'),retf
if row=1, a=z; retf
if row<3, ...
  a = convolve(z(1,[1:n]),z(2,[n+1:all])) - ...
        convolve(z(1,[n+1:all]),z(2,[1:n]));...
else ...
  a=convolve(z(1,[1:n]),seqdet(z([2:row],[n+1:all]),n)); ...
  a=a+((-1)**(col-1))*convolve(z(1,[last+1:all]), ...
                           seqdet(z([2:row],[1:last]),n));...
  for i=2:(col-1), ...
    of1=(i-1)*n; ...
    of2=i*n; ...
    a=a+((-1)**(i-1))*convolve(z(1,[of1+1:of2]), ...
                 seqdet(z([2:row],[1:of1 of2+1:all]),n));end,end
retf
```

USER.FOR

This is the FORTRAN routine which links into MATRIXX through the 'USER' function. This version, rather than calling an LP solver, writes a data file which can be read later by an LP solver.

```
      SUBROUTINE USER ( XLOCAL, NRX, NCX, SLOCAL, TLOCAL )
C
C ---------------------------------------------------------------
C |          MATRIXx TM Software V7.0    (C) Copyright 1987   |
C |          INTEGRATED SYSTEMS INC., Santa Clara, California |
C |            Unpublished Work. All Rights Reserved Under    |
C |                   The U.S. Copyright Act.                 |
C |                                                           |
C |        This routine is part of the ANSI subset of MATRIXx.|
C ---------------------------------------------------------------
C |                                                           |
C | Subroutine USER allows personal Fortran subroutines to be |
C | linked into MATRIXx.  The MATRIXx statement               |
C |                                                           |
C |     <> Y = USER ( X, S, T )                               |
C |                                                           |
C | causes subroutine USER to be called in the following way: |
C |                                                           |
C |     CALL USER ( XLOCAL, NRX, NCX, SLOCAL, TLOCAL )        |
C |                                                           |
C | where XLOCAL, SLOCAL, and TLOCAL are local copies of X, S,|
C | and T; and NRX, NCX are the number of rows and number of  |
C | columns of X.  If S and T are omitted in the invocation,  |
C | SLOCAL and TLOCAL are set to zero. The contents of XLOCAL,|
C | NRX, and NCX may be reset within the subroutine.  The     |
C | final contents of XLOCAL (which is NRX-by-NCX) is returned|
C | to the MATRIXx data stack as Y, with dimensions NRX-by-NCX|
C | which may have been changed (i.e. Y can have different    |
C | dimension than X                                          |
C ---------------------------------------------------------------
```

```
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION XLOCAL(NRX,NCX)
      COMMON /USERLP/ MLT,ZZCON(100000)
      DIMENSION IEQ(1000)
      DIMENSION B(1000)
      CHARACTER*10 FNAME
C
C       ------------------------------------------
C            THIS VERSION BUILDS LINE-BY-LINE
C       ------------------------------------------
C
      LEN=2*NCX
      NCONST=TLOCAL
      IOFF=(NCONST+1)*LEN
      IF (SLOCAL.EQ.ODO) THEN
C
C . . .   COPY RHS AND CONSTRAINTS FROM MATRIXX.
C . . .    EXPAND TO PHI(i)=X(i)-Y(i).
C
        RHS=XLOCAL(1,NCX)
        DO 50 I=1,LEN
   50     ZZCON(IOFF+I)=0.0
        NCONST=NCONST+1
        ZZCON(IOFF+1)=DABS(RHS)
        DO 100 J=2,NCX
          IF (RHS .GE. 0.0) THEN
            COEFF=XLOCAL(1,J-1)
          ELSE
            COEFF=-XLOCAL(1,J-1)
          ENDIF
          ZZCON(IOFF+J)=COEFF
          ZZCON(IOFF+J+NCX-1)=-COEFF
  100   CONTINUE
      ENDIF
C
C . . .   NOW PUT IN sum(|phi|) < u
C
      IF (SLOCAL .EQ. 1DO) THEN
        NROW=XLOCAL(1,1)
```

```
       NCOL=XLOCAL(1,2)
       NPHI=(NCX-1)/NROW/NCOL
       DO 350 J=1,NROW
         DO 310 I=1,LEN
 310       ZZCON(IOFF+I)=0.0
         IOF1=(J-1)*NCOL*NPHI+2
         DO 300 I=IOF1,IOF1+NCOL*NPHI-1
           ZZCON(IOFF+I)=1.0
           ZZCON(IOFF+I+NCX-1)=1.0
 300     CONTINUE
         NCONST=NCONST+1
         IOFF=(NCONST+1)*LEN
         ZZCON(IOFF)=-1.0
 350   CONTINUE
C
C   PUT IN OBJECTIVE FUNCTION
C
C . . .WANT TO MINIMIZE U
C
       DO 500 I=1,LEN
 500     ZZCON(I)=0.0
       ZZCON(LEN)=1.0
       MLT = NCONST
     ENDIF
C
C      ------------------------------------------
C                SOLVE THE LINEAR PROGRAM
C      ------------------------------------------
C
      IF (SLOCAL .EQ. 2D0) THEN
C        DO 101 I=1,IOFF
C  101     IF (DABS(ZZCON(I)).LT.1D-11) ZZCON(I)=0.0
        M=NCONST
        N=LEN-1
        MP=M+2
        NP=N+1
        MEQ=NCONST-MLT
C
C . . .WRITE DATA FILE.  INSERT LP SOLVER HERE
C . . .  IF DESIRED.
```

```
C
        PRINT *, M,' ',N
        PRINT *,'WRITING DATA FILE'
        OPEN( UNIT=15, FORM='UNFORMATTED', TYPE='NEW',
     1   ERR=9000, RECORDTYPE='VARIABLE',RECL=2000)
        REWIND 15
        WRITE(15) 8,M,N,8
        DO 10 I=1,M
          IOFF1=I*LEN+2
          IOFF2=IOFF1+LEN-2
          WRITE(15) 8*N,(ZZCON(J),J=IOFF1,IOFF2),8*N
          B(I)=ZZCON(IOFF1-1)
          IF (I.LE.MLT) THEN
            IEQ(I)=-1
          ELSE
            IEQ(I)=0
          ENDIF
   10   CONTINUE
        WRITE(15) 8*N,(ZZCON(I),I=2,LEN),8*N
        WRITE(15) 8*M,(B(I),I=1,M),8*M
        WRITE(15) 4*M,(IEQ(I),I=1,M),4*M
        CLOSE(15)
      ELSE
        NCX=1
        NRX=1
        XLOCAL(1,1)=NCONST
      ENDIF
      RETURN
 9000 PRINT *, 'HORRIBLE ERROR'
      RETURN
      END
```

CLSS.UDF

```
//[SCL,NSCL]=CLSS(PHI,LENPHI,ROW,COL)
//
//  CALCULATE STATE SPACE XFER FUNCTION
//
NSCL=lenphi-1;
nscl1=nscl;
ka=0*eye(NSCL); for i=1:NSCL-1; ka(i+1,i)=1; end
b=0*ones(NSCL,1); b(1)=1;
off=1;
for i=1:col, ...
  for j=1:row, ...
    c(j,[1:NSCL])=phi(off+1:off+lenphi-1); ...
    d(j,1)=phi(off); ...
    off=off+lenphi; ...
  end, ...
  if i=1, SCL=[ka b; c d]; ...
  else, [SCL,nscl1]=bldk(SCL,nscl1,[ka b;c d],NSCL); end, ...
end
NSCL=nscl1;
//
retf
```

BLDK.UDF

```
//[KNEW,NSN]=BLDK(S1,NS1,S2,NS)
//
//  GETS MINIMAL REALIZATION FOR K
//
[a1,b1,c1,d1]=split(s1,ns1);
[a2,b2,c2,d2]=split(s2,ns);
KNEW=[[a1 0*eye(a1);0*eye(a2) a2] [b1;0*b1] [0*b2;b2]
       c1 c2 d1 d2];
nsn=ns1+ns;
[KNEW,nsn]=minimal(KNEW,nsn); end
retf
```

KCALC.UDF

```
//[SK,NSK]=KCALC(Q,NSQ,G22,NS)
//
//  Calculate Controller from G22 and Q
//
[m,n,x,y]=coprim(g22,ns);
if NSQ>0, ...
  display(' -- MQ --'), ...
  [m,nsm]=ssmul(m,ns,q,nsq); ...
  [m1,nsm]=minimal(m,nsm); ...
  if nsm<ns+nsq, m=m1; end, ...
  clear m1, ...
  display(' -- NQ --'), ...
  [n,nsn]=ssmul(n,ns,q,nsq); ...
  [n1,nsn]=minimal(n,nsn); ...
  if nsn<ns+nsq, n=n1; end, ...
  clear n1; ...
  display(' -- Y-MQ --'), ...
  y=ssadd(y,ns,m,nsm,-1); ...
  display(' -- X-NQ --'), ...
  x=ssadd(x,ns,n,nsn,-1); ...
  [x1,nsx]=minimal(x,ns+nsn); ...
  if nsx<ns+nsn, x=x1; end, ...
  clear x1, ...
  [y1,nsy]=minimal(y,ns+nsm); ...
  if nsy<ns+nsm, y=y1; end, ...
  clear y1, ...
end
if exist('nsx')=0, nsx=ns; end
if exist('nsy')=0, nsy=ns; end
x=ssinv(x,nsx);
display(' -- (Y-MQ)/(X-NQ) --')
[SK,NSK]=ssmul(y,nsy,x,nsx);
[SK1,NSK]=minimal(SK,nsx+nsy);
if nsk<nsx+nsy, SK=SK1; end
//
retf
```

KSS.UDF

```
//[SKOPT,NSK]=KSS(T1,NS,PHI,LENPHI,NPHI)
//
//  TURN PHI (FROM LP) INTO STATE SPACE
//
//    if len=lenphi, for k=1:(len-2*ns), yy=[yy;0]; end, end, ...
[row,col]=size(t1);
row=row-2*ns;
col=col-2*ns;
len=max([lenphi 2*ns+1]);
for i=1:row, ...
  for j=1:col, ...
    [xx,yy] = ...
      pulse(t1([1:2*ns 2*ns+i],[1:2*ns 2*ns+j]),2*ns,len); ...
    for k=1:len, ...
      if k<=lenphi, ...
        uqv(i,(j-1)*len+k) = ...
          yy(k)-phi((i-1)*lenphi*col+(j-1)*lenphi+k); ...
      else, uqv(i,(j-1)*len+k)=yy(k); end, ...
  end, end, ...
end
//
//   TURN K INTO STATE SPACE
//
NSK=len-1;
ka=0*eye(NSK); for i=1:NSK-1, ka(i+1,i)=1; end
b=0*ones(NSK,1); b(1)=1;
for i=1:col, ...
  clear c d; ...
  for j=1:row, ...
    c(j,[1:NSK])=uqv(j,[(i-1)*len+2:(i*len)]); ...
    d(j,1)=uqv(j,(i-1)*len+1); ...
  end, ...
  if i=1, [SKOPT,nsk1]=minimal([ka b;c d],NSK); ...
  else, [SKOPT,nsk1]=bldk(SKOPT,nsk1,[ka b;c d],NSK); end, ...
end
NSK=nsk1;
retf
```

COPRIME.UDF

```
//[M,N,X,Y]=COPRIM(S,NS)
//
// GET COPRIME FACTORIZATION OF PLANT
//   FOR  K=(Y-MQ)/(X-MQ)
//
[a,b,c,d]=split(s,ns);
[row,col]=size(d);
//
//  TRY PLACING POLES AT ORIGIN for factorization
//
p=poly(a); for i=1:ns, pm(i,:)=p'; end
bp=0*ones(ns,1); for i=1:col, bp=bp+b(:,i); end
sc=bp; scc=eye(ns);
for i=2:ns, ...
  sc(:,i) = ...
   a*sc(:,(i-1)); scc=scc+diag(pm([1:ns+1-i],i),(i-1)); end
t=sc*scc; tinv=inv(t);
f1=p(2:ns+1)'*tinv;
for i=1:col, f(i,:)=f1; end
evcnt=eig(a+b*f)
cp=0*ones(1,ns); for i=1:row, cp=cp+c(i,:); end
sc=cp'; scc=eye(ns);
for i=2:ns, ...
  sc(:,i) = ...
   a'*sc(:,(i-1)); scc=scc+diag(pm([1:ns+1-i],i),(i-1)); end
t=sc*scc; tinv=inv(t);
h1=tinv'*p(2:ns+1);
for i=1:row, h(:,i)=h1; end
evobs=eig(a+h*c)
n=[a+b*f b; c+d*f d];
m=[a+b*f b; f eye(col)];
x=[a+b*f -h; c+d*f eye(row)];
y=[a+b*f -h; f 0*d'];
retf
```

PTOSS.UDF

```
//[S,NS]=PTOSS(NUM,DEN,IN)
//
//  Put NUM/DEN into State Space doing
//    MINIMAL along the way.  (Because
//    SFORM creates too big matrix.
//
[i,j]=size(den);
NS=j-1;
[i,j]=size(num);
j=j/in;
ka=0*eye(NS); for k=1:NS-1, ka(k+1,k)=1; end
c=0*ones(1,NS); c(ns)=1;
for k=1:i, ...
  b=0*ones(NS,in); ...
  d=0*ones(1,in); ...
  for l=1:in, ...
    if j=NS+1, ist=2; else ist=1; end, ...
    for m=ist:j, ...
      b(m-ist+1,l)=num(k,(j+ist-m-1)*in+l); ...
      if ist=2, d(l)=num(k,l); ...
    end, ...
  end, ...
  if k=1, [S,ns1]=minimal([ka b;c d],NS); ...
  else, [S,ns1]=bldk(S,ns1,[ka,b;c d],NS); end, ...
end
NS=ns1;
retf
```

CHECK.UDF

```
//IFL=CHECK(Z,Z1,F,TOL)
//
//  F=1  CHECK FOR CONJUGATE PAIR
//  F=2  CHECK FOR EQUALITY
//  F=3  CHECK REAL PARTS FOR EQUALITY
//  F=4  CHECK FOR NORM=1
//
if tol<0, tol=1d-14; end
IFL=0;
rz=real(z); riz=imag(z);
rz1=real(z1); riz1=imag(z1);
if f=1, ...
  if abs(rz-rz1)<=tol*abs(rz), ...
    if abs(riz+riz1)<=tol*abs(riz), ...
      if abs(riz)>tol, IFL=1; ...
end,end,end,end
//
if f=2, ...
  if abs(rz-rz1)<=tol*abs(rz), ...
    if abs(riz-riz1)<=tol*abs(riz), IFL=1; ...
end,end,end
//
if f=3, ...
  if abs(rz-rz1)<=tol*abs(rz), IFL=1; ...
end,end
//
if f=4, ...
  if abs(norm(z)-z1)<=tol, IFL=1; ...
end, end
retf
```

FACT.UDF

```
//[NUM]=FACT(N1,N2)
// FINDS N1*(N1+1)* ... *N2
//
if n2<n1, num=1; retf
num=1;
for i=n1:n2, num=num*i; end
retf
```

SSADD.UDF

```
//[SSUM,NSUM]=SSADD(S1,NS1,S2,NS2,FAC)
//
// GIVES [A',B',C',D']=[A1,B1,C1,D1]+FAC*[A2,B2,C2,D2]
//
[a1,b1,c1,d1]=split(s1,ns1);
[a2,b2,c2,d2]=split(s2,ns2);
b2=b2*fac;
[m,n]=size(a2);
[i,n]=size(a1);
ssum = [[a1 0*b1(:,1)*c2(1,:);0*b2(:,1)*c1(1,:) a2]  [b1;b2]; ...
                       [c1 c2]                          d1+fac*d2 ];
nsum=ns1+ns2;
retf
```

SSMUL.UDF

```
//[SPROD,NSP]=SSMUL(S1,NS1,S2,NS2)
// GIVES [A',B',C',D']=[A1,B1,C1,D1]*[A2,B2,C2,D2]
//
[a1,b1,c1,d1]=split(s1,ns1);
[a2,b2,c2,d2]=split(s2,ns2);
[m,n]=size(a2);
[i,n]=size(a1);
sprod = [ [a1 b1*c2;0*b2(:,1)*c1(1,:) a2] [b1*d2; b2];...
                      [c1 d1*c2]                 d1*d2 ];
NSP=ns1+ns2;
retf
```

SSINV.UDF

```
//[TINV,J]=SSINV(S1,NS1)
//
// GIVES [A,B,C,D]=INV([A1,B1,C1,D1])
//   GENERATES AN APPROPRIATE LEFT OR RIGHT
//    INVERSE IF S1 NON-SQUARE
//
[a,b,c,d]=split(s1,ns1);
[nrow,ncol]=size(s1);
nrow=nrow-ns1;
ncol=ncol-ns1;
j=0;
if max(d)=0, ...
  [num,den]=tform(s1,ns1); ...
  [i,j]=size(num); ...
  i=j/ncol; ...
  for j=i:ns1, ...
    num=[num 0*ones(nrow,ncol)]; end, ...
  [s1,ns1]=sform(num,den); ...
  [a,b,c,d]=split(s1,ns1); ...
  j=ns1+1-i; ...
end
if nrow=ncol, tinv=[a-b*inv(d)*c b*inv(d);-inv(d)*c inv(d)]; retf
if nrow>ncol, ...
  n=nrow/ncol; ...
  for i=1:n, ...
    clear cc; ...
    ind1=(i-1)*ncol+1; ...
    ind2=i*ncol; ...
    cc=cond(d([ind1:ind2],:)); ...
    if exist('cc')<>0, ...
      if cc>1e8, ...
        display(' -- Condition number > 1e8 --'), end, ...
      x=c([ind1:ind2],:); ...
      y=d([ind1:ind2],:); ...
      dp=0*d'; dp(:,[ind1:ind2])=inv(y); ...
      bp=0*c'; bp(:,[ind1:ind2])=b*inv(y); ...
      tinv=[a-b*inv(y)*x bp;-inv(y)*x dp]; ...
      retf, ...
  end, ...
end
if ncol>nrow, ...
  n=ncol/nrow; ...
  for i=1:n, ...
    clear cc; ...
    ind1=(i-1)*nrow+1; ...
    ind2=i*nrow; ...
    cc=cond(d(:,[ind1,ind2])); ...
    if exist('cc')<>0, ...
      if cc>1e8, ...
        display(' -- Condition number > 1e8 --'), end, ...
      x=b(:,[ind1:ind2]); ...
      y=d(:,[ind1:ind2]); ...
      dp=0*d'; dp([ind1:ind2],:)=inv(y); ...
      cp=0*b'; cp([ind1:ind2],:)=-inv(y)*c; ...
      tinv=[a-x*inv(y)*c x*inv(y);cp dp]; ...
      retf, ...
  end, ...
end
retf
```

Appendix B

Space Station Pitch Axis System Matrix

Figures B.1 and B.2 show the continuous time matrices describing the space station system, and Figures B.3 and B.4 show the discretized versions with Δt = 190 sec. Both **C** matrices are 4×4 identity matrices and are not listed.

$$\mathbf{A} = \begin{bmatrix} 0 & 2.7864 \times 10^{-6} & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure B.1: Continuous Time Pitch **A** Matrix

$$\mathbf{B}_1 = \begin{bmatrix} 9.2593\times10^{-8} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{B}_2 = \begin{bmatrix} 9.2593\times10^{-8} \\ 0 \\ -1 \\ 0 \end{bmatrix}$$

Figure B.2: Continuous Time Pitch **B** Matrices

$$\mathbf{A} = \begin{bmatrix} 1.0507 & 5.3833\times10^{-4} & 0 & 0 \\ 1.9320\times10^{2} & 1.0507 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1.9000\times10^{2} & 1 \end{bmatrix}$$

Figure B.3: Discrete Time Pitch **A** Matrix

$$\mathbf{B}_1 = \begin{bmatrix} 1.7889\times10^{-5} \\ 1.6854\times10^{-3} \\ 0 \\ 0 \end{bmatrix}, \mathbf{B}_2 = \begin{bmatrix} 1.7889\times10^{-5} \\ 1.6854\times10^{-3} \\ -1.9000\times10^{2} \\ -1.8050\times10^{4} \end{bmatrix}$$

Figure B.4: Discrete Time Pitch **B** Matrices

Appendix C

Space Station Roll and Yaw System

Figures C.1 and C.2 show the **A** and **B** matrices for the discretized space station roll/yaw dynamics. In the **B** matrix, the first two columns are for the disturbance inputs and the last two columns are for the control inputs.The stabilizing regulator gains are listed in Fig. C.3.

Columns 1-4:

$$\begin{bmatrix}
1.0115 \times 10^{0} & 4.1115 \times 10^{-1} & 8.7707 \times 10^{-4} & 3.1719 \times 10^{-5} \\
-3.5296 \times 10^{-1} & 9.4297 \times 10^{-1} & -1.5352 \times 10^{-4} & 1.5203 \times 10^{-4} \\
1.9074 \times 10^{2} & 3.8889 \times 10^{1} & 1.0832 \times 10^{0} & 2.0053 \times 10^{-3} \\
-3.3385 \times 10^{1} & 1.8640 \times 10^{2} & -9.7057 \times 10^{-3} & 1.0146 \times 10^{0} \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}$$

Columns 5-8:

$$\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
9.7824 \times 10^{-1} & 2.0748 \times 10^{-1} & 0 & 0 \\
-2.0748 \times 10^{-1} & 9.7824 \times 10^{-1} & 0 & 0 \\
1.8862 \times 10^{2} & 1.9783 \times 10^{1} & 1 & 0 \\
-1.9783 \times 10^{1} & 1.8862 \times 10^{2} & 0 & 1
\end{bmatrix}$$

Figure C.1: Discrete Time Roll/Yaw A Matrix

$$\begin{bmatrix} 3.7935 \times 10^{-6} & 6.6398 \times 10^{-7} & 3.7935 \times 10^{-6} & 6.6398 \times 10^{-7} \\ -6.6398 \times 10^{-7} & 3.1825 \times 10^{-6} & -6.6398 \times 10^{-7} & 3.1825 \times 10^{-6} \\ 3.5969 \times 10^{-4} & 4.1978 \times 10^{-5} & 3.5969 \times 10^{-4} & 4.1978 \times 10^{-5} \\ -4.1978 \times 10^{-5} & 3.0526 \times 10^{-4} & -4.1978 \times 10^{-5} & 3.0526 \times 10^{-4} \\ 0 & 0 & -1.8862 \times 10^{2} & -1.9783 \times 10^{1} \\ 0 & 0 & 1.9783 \times 10^{1} & -1.8862 \times 10^{2} \\ 0 & 0 & -1.7984 \times 10^{4} & -1.2547 \times 10^{3} \\ 0 & 0 & 1.2547 \times 10^{3} & -1.7984 \times 10^{4} \end{bmatrix}$$

Figure C.2: Discrete Time Roll/Yaw **B** Matrix

Columns 1-4:

$$\left[\begin{matrix} 1.1595 \times 10^{5} & 4.9158 \times 10^{4} & 2.5428 \times 10^{2} & -4.1107 \times 10^{1} \\ 2.5914 \times 10^{4} & 1.0981 \times 10^{5} & 1.8354 \times 10^{2} & 6.0033 \times 10^{1} \end{matrix}\right.$$

Columns 5-8:

$$\left.\begin{matrix} 4.1508 \times 10^{-5} & 5.2889 \times 10^{-5} & -1.0000 \times 10^{-7} & -1.0000 \times 10^{-7} \\ -2.1350 \times 10^{-5} & 3.1989 \times 10^{-5} & 1.0000 \times 10^{-8} & -1.0000 \times 10^{-8} \end{matrix}\right]$$

Figure C.3: G Such That $(A - BG)$ Is Stable

Bibliography

[1] Dahleh, Munther A.,"Design of Multivariable Feedback Controllers: l_1-Optimal Systems," PhD Thesis, Rice University, Houston, Texas, Nov. 1986.

[2] Oglevie, R.E.,"Space Station Attitude Control — Challenges and Options," AAS 83-066, Annual Rocky Mountain Guidance and Control Conference, Keystone, Colorado, Feb. 5-9, 1983.

[3] Johnson, C.D., and Skelton, R.E.,"Optimal Desaturation of Momentum Exchange Control Systems," *AIAA Journal*, Vol. 9, No. 1, 1971.

[4] Bishop, L.R., et al., "Proposed Momentum Management Scheme for Space Station,"

[5] Wie, B., et al., "A New Momentum Management Controller for the Space Station," submitted to *AIAA Journal of Guidance, Control, and Dynamics* Jan. 19, 1988.

[6] Francis, Bruce A., *A Course in H_∞ Control Theory*, Springer-Verlag, Berlin, 1987.

[7] Wertz, James R., ed., *Spacecraft Attitude Determination and Control*, D. Reidel Publishing Company, Boston, 1985.

[8] Kane, Thomas R., et al., *Spacecraft Dynamics*, McGraw-Hill Publishing Company, New York, NY, 1983.

[9] Vidyasagar, M., *Control System Synthesis: A Factorization Approach*, MIT Press, Cambridge, MA, 1987.

[10] Voulgaris, Petros, "High Performance Multivariable Control of the 'Super-maneuverable' F18/HARV Fighter Aircraft," M.S. Thesis, Massachusetts Institute of Technology, 1986.

[11] Quinn, Wilma W., "Multivariable Control of a Forward Swept Wing Aircraft," Laboratory for Information and Decision Systems Report LIDS-TH-1530, Massachusetts Institute of Technology, 1986.