# Applications of Nonlinear Programming for Automated Multivariable Control System Design

Gary W. Machles* and Francis D. Hauser†
*Boeing Aerospace Company, Seattle, Washington*

**Conventional control system design, via the classical concepts of gain and phase stability margins and damping ratios, continues to be a principal design approach. This paper discusses the automation of the conventional control system design process using nonlinear programming. Examples are given for three aerospace vehicles. Modern control theory, including the Linear Quadratic Regulator methods, offers advantages when designing multivariable systems. However, conventional control system design can easily include multivariable systems when nonlinear programming techniques are applied. With nonlinear programming, engineering problems can be solved directly; i.e., the problem does not have to be fit into a canonical form.**

## Introduction

THE problem that motivated development of the automated design program was the expensive, time-consuming, and redundant analysis necessary for generating mission-dependent data for a variable payload vehicle (VPV) thrust vector control system. For each mission, the autopilot hardware was fixed, but new gain and filter values were required because of the different flight and payload conditions. The objective was to develop a technique for selecting control system variables that would be inexpensive, provide a quick turnaround, and be user friendly. Furthermore, it was desired to create a multipurpose program that could be used for selecting control system variables for any proven (hence practical) control system design. This effort resulted in development of the Automated Design and Optimization of Control Systems (ADOCS) software package.

Perhaps the easiest way to explain how the ADOCS package works is to review the typical control system design process (see Fig. 1). Of course, the first steps are to understand the system and its performance requirements. Mathematical modeling and data gathering are part of this initial and often complex effort. Once the mathematical model has been verified, frequency and time domain analyses are needed to establish the system's actual performance characteristics. Next, the designer chooses various forms of compensation that will provide the system with the required performance characteristics. Experienced designers can often choose the appropriate types of filters and feedback networks, but even the best designers will rarely obtain the required system performance in one step. At this point the iterative process begins. The designer repeats the frequency and time response computations, adjusts the compensator, recomputes the frequency and time response, and so on, until all requirements are met. These iterations can often be very time consuming, costly, and technically unchallenging. If the system is multivariable, the iterations are greatly increased, as the designer must also iterate between loops. Similarly, time-varying systems require additional iterations at each time point.

The ADOCS package automates the iterative steps in this process. The frequency and time response of the system are computed by a control system analysis tool. Linked to the analysis tool is an interface program which searches the system's response for all the stability and performance criteria (e.g., gain margins, phase margins, and bandwidth). Constraints (e.g., 6-dB gain margin, 30-deg phase margin) are assigned to each criterion, setting up a matrix of inequalities relating all the criteria, constraints, and control system variables together. The data are then passed to the optimizer, which selects new values for the variables. These new values are passed back to the analysis tool, and the process is repeated until all requirements are met.

References 1-3 are compendia of the broad field of optimization theory, including nonlinear programming and its recent applications. In general, the nonlinear programming problem is to determine values for $n$ variables ($X_n$), which satisfy $m$ nonlinear constraints

$$g_i(X_1,\ldots,X_n)\{\leq,=,\geq\}b_i \qquad i=1,\ldots,m$$

(note that $n$ is not related to $m$) and, in addition, maximize (or minimize) a nonlinear objective or goal function: $Z=f(X_1,\ldots,X_n)$. Many engineering problems fit this formulation, although often the $g$'s and $z$ are only in the engineer's mind as the problem is solved by trial and error. The field of nonlinear programming contains systematic ways to solve the problem if it can be quantified. In most cases it can be. In ADOCS, the $X$'s are the control system variables, the $g$'s are the equations defining the stability margins and performance measures like bandwidth, and the $b$'s are the requirements or constraints on these performance measures. As will be detailed later, the objective function ($Z$) is "built by the user" and, therefore, addresses exactly what the user desires, e.g., maximum rigid body gain margin and phase margin (see Fig. 2).

## The Linear Analysis Program

A linear analysis program is used in the ADOCS software package. Several linear analysis software packages exist throughout industry and at universities. Regardless of what linear analysis tool is used, ADOCS uses it to determine the system's frequency response and/or time response (if time performance parameters like percent overshoot and rise time are to be optimized).

## The Interface Program

The interface program is key to automating the design process. It provides an intelligent link between the linear analysis program and the optimizer by setting up the exact engineering problem to be solved. It searches for the system's stability and

Presented as Paper 83-2275 at the AIAA Guidance and Control Conference, Gatlinburg, Tenn., Aug. 15-17, 1983; submitted April 5, 1984; revision received Sept. 18, 1984. Copyright © 1984 by the Boeing Company. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission.

*Senior Engineer.

†Senior Engineering Specialist.

performance characteristics and calculates the objective function and vector of constraints needed by the optimizer. The stability and performance characteristics are found by searching the open- and closed-loop frequency response for critical points (e.g., 0-dB and 180-deg crossover points, bending modes, etc). The constraints are established by the user in the form of data statements and subroutines. For instance, the user may have stability requirements of 6-dB gain and 30-deg phase margin for the rigid mode, with bending modes to be gain stabilized greater than 20 dB. In this case, the constraint values are merely placed in data statements. However, if the criteria vary (e.g., requiring the bending modes to be both gain and phase stabilized outside of some region) an evaluation subroutine may be necessary. Figure 3 shows how the bending mode constraints were established for the variable payload vehicle (VPV). Here the system frequency response is excluded from a parabolic region centered at the 180-deg, 0-dB stability point. The closest approach is then defined as the shortest distance between the frequency response and the parabola, normal to the parabola. If the frequency response enters the parabola, a negative value with the magnitude of its deepest penetration is assigned as its actual margin.

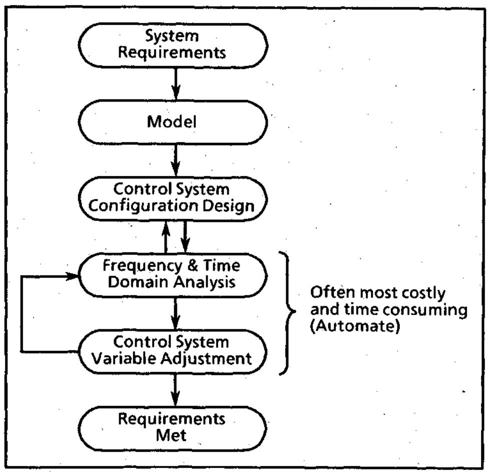
System Requirements
Model
Control System Configuration Design
Frequency & Time Domain Analysis
Control System Variable Adjustment
Requirements Met
Often most costly and time consuming (Automate)

Fig. 1 Conventional control system design process.

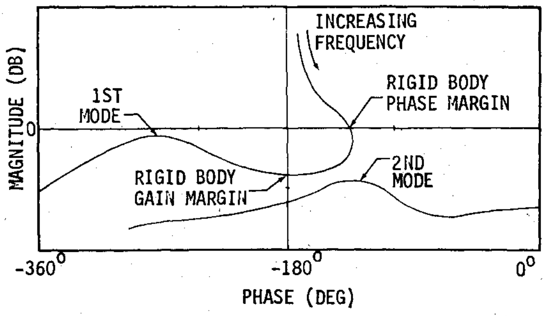

Fig. 2 Typical autopilot frequency response.

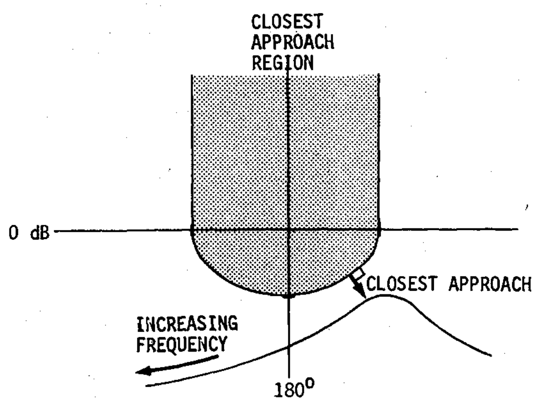

Fig. 3 VPV bending mode constraints.

The multivariable design problem is solved as follows: the interface program prompts the linear analysis program to perform a frequency response for each loop combination. It then searches each frequency response, storing each margin or performance measure with its requirement. The data obtained from each frequency response are combined into composite criteria and constraint arrays for the entire system. These composite arrays are then passed to the optimizer allowing it to optimize the variables for all loops simultaneously.

The problem of choosing constant control parameters for time-varying systems or for robust systems is also solved by the interface program. Similar to the multivariable design approach, the interface program forms composite arrays from several time steps (for time-varying systems) or from several vehicle conditions (for robust systems). These arrays are then passed to the optimizer for optimization over all time points or conditions.

Along with the performance criteria and constraints, the interface program calculates an objective function which the optimizer maximizes. This function is completely user defined and relatively simple to set up. If the user wants to maximize the system's bandwidth then—objective function = bandwidth. If several of the system's characteristics are to be maximized, then they may be put into the objective function as a normalized sum. For instance, if the user wants to maximize the rigid mode stability margins, the objective function might be—objective function = $W_1 \times$ gain margin + $W_2 \times$ phase margin, where the $W$'s are normalizing factors. For example, $W_1$ might = 1, and $W_2$ might = 6/30 so that a 30-deg phase margin would be equivalent to a 6-dB gain margin. It is worth noting that any of the constraints can be easily used in the objective function since the margin finder makes them readily available.

## The Optimizer

The optimizer in ADOCS is derived from the work documented in Ref. 4. The fundamental principle is to iterate to the final solution by solving a succession of linear programming problems using the Simplex algorithm.[5] The optimizer is classified as an inequality-constrained optimization method. It is also a parameter optimization method; i.e., it finds the best values for variables whose "first guess" values are input by the user. Two distinctive features of the optimizer are 1) the "first guess" does not have to satisfy any of the constraints (which is often the situation in real engineering problem), and 2) it can converge to local interior as well as exterior optima.

Via the Interface Program, the optimizer receives the following data: 1) an array, $\bar{X}$, containing the "first guess" values for the variables; 2) arrays, $\bar{X}_U$ and $\bar{X}_L$, containing the minimum and maximum allowed values for the variables; 3) an array, $\bar{M}$, containing the current values of each of the performance or stability criteria (for example, the first element in this array might be reserved for the lowest-frequency phase margin, whose current value is 20 deg; the first element of this array would then be 20); 4) an array, $\bar{M}_c$, containing the required values (constraint) for each of the criteria (for example, the first element of this array might be 30 deg); and 5) the current value of the objective function $Z$.

Using the values in the $X$ array, the optimizer forms the following inequality constraint equations, which are first-order Taylor Series expansions.

$$M_i(\bar{X}_0+\Delta\bar{X})=M_i(\bar{X}_0)+\sum_{j=1}^{n}\left(\frac{\partial M_i}{\partial X_j}\right)_0\Delta X_j\left\{\begin{matrix}\le\\=\\\ge\end{matrix}\right\}M_{ic} \quad (1)$$

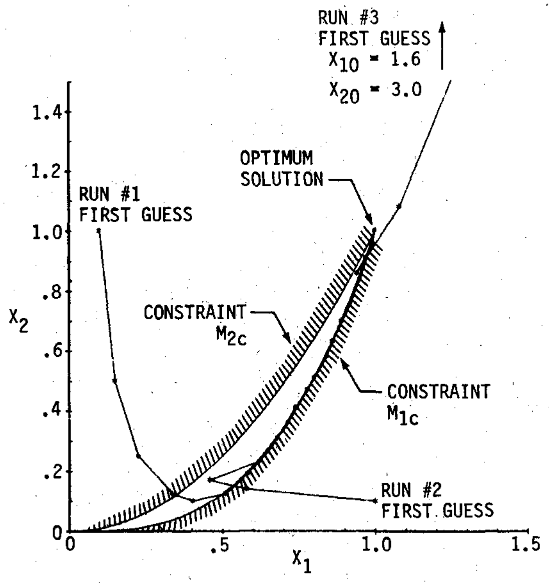

Fig. 4 Academic optimization example.[6]

where $\Delta X_0 = X_j - X_{j0}$ and $X_{j0}$ is the value about which the Taylor Series is formed. The finite differencing method is used to calculate the partial derivatives.

The optimizer forms an additional set of constraint equations, which establish the upper and lower bounds for each variable.

$$X_{jl} \leq X_j \leq X_{ju} \; (j=1,\ldots,n) \tag{2}$$

The objective function is also expanded in a first-order Taylor Series.

$$Z(\bar{X}_0 + \Delta\bar{X}) = Z(\bar{X}_0) + \sum_{j=1}^{n} \left(\frac{\partial Z}{\partial X_j}\right)_0 \Delta X_j \tag{3}$$

Equations (1-3) form a linear programming problem. Reference 4 illustrates how the optimizer iterates from the "first guess" to the desired solution by solving successive linear programming problems.

Figure 4 shows the application of the optimizer subroutine to an academic problem given in Ref. 6.

Maximize: $Z = X_1$

Subject to: $M_{1c} = X_2 - X_1^3 > 0$

$M_{2c} = X_1^2 - X_2 > 0$

$X_1, X_2 > 0$

It shows how the optimizer iterated to the solution from three different infeasible "first guesses." It also shows another distinctive feature of the optimizer: it concentrates on the constraints. Run 2 shows how the objective function decreases during the first few iterations to reach a feasible solution. The optimizer moves in a "deflected gradient" direction (i.e., the objective function gradient is deflected by the constraints). Often this is exactly what the engineer does, since most real engineering problems are dominated by the constraints with the objective function providing only a general direction.

## Examples

The following examples illustrate the application of ADOCS to real hardware design. The results from one application were coded into the onboard software of a vehicle that actually flew. The other applications are from vehicles that will be test flown in the near future.

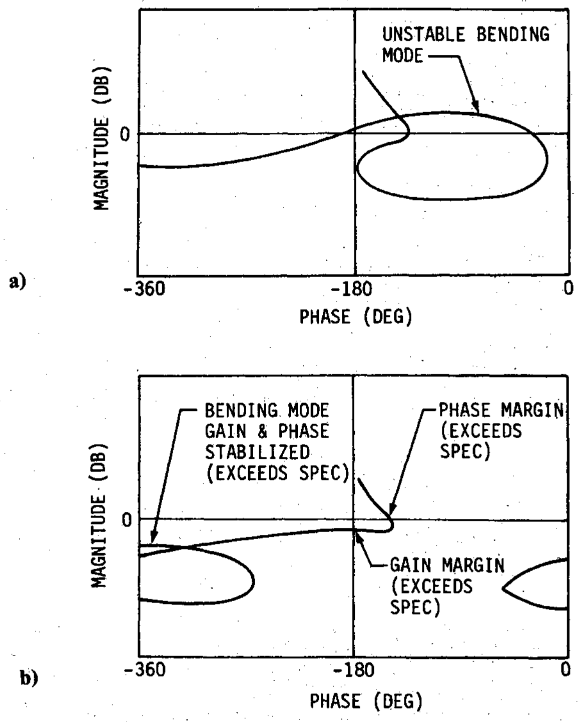

Fig. 5 VPV optimization: a) unstable VPV with bending mode; b) 6 iterations later, stabilized with all margins met.

The examples demonstrate that the automated process can be more cost-effective than the manual process. However, it should be noted that the ADOCS computational and storage requirements are primarily dependent on the details of the model, the linear analysis program, and the number of times it is executed. (The interface program and optimizer combined use less than 10 pages of Fortran code.)

### Example 1

The ADOCS package was applied to the control system of a gimballed-thrust variable payload vehicle. The mathematical model included both rigid and structural modes. The control system variables to be optimized were the position and rate gains and the frequency and damping of the body bending and rate filters. The constraints included a rigid-body gain margin of 6 dB, a rigid-body phase margin of 30 deg, a bending mode closest-approach margin (see Fig. 3), a bandwidth of 0.5 Hz, and a rigid-mode damping ratio of greater than 0.7. Results are shown in Fig. 5. Figure 5a shows that the initial user-defined variables resulted in an unstable system. In addition, the bending mode was grossly out of place. For this reason, the objective function was set equal to the closest approach margin, allowing the optimizer to weigh that constraint more heavily than the others. In six iterations, variables were chosen that stabilized the system and exceeded the required stability margins (see Fig. 5b). In addition, the system bandwidth and damping maintained their constrained values. The final variable selection resulted in a more optimum system than the system whose variables were selected independently by manual techniques requiring significantly more engineering and computer time.

### Example 2

A standard solid-rocket missile autopilot was used for a second test case. The ADOCS package was used on this problem because the structural properties of the missile had changed drastically and a fast "retune" of the autopilot was needed to decide if an autopilot redesign was necessary. Figure 6 shows the stability margin requirements for this vehicle. Here, somewhat conservative rigid-mode requirements were desired because of the uncertainties in the gain computations and fin effectiveness. The margin finder was easily modified to obtain the two 150-deg crossover points and to calculate the $\Delta K$'s. Because there were three bending modes (each requiring slightly different gain margins), the bending mode margins were searched for on the basis of frequency as well as their peaks and locations. The control system variables were notch-filter damping coefficients, lead and lead-lag compensator frequencies, rate gain, and position gain. All of the variables were scheduled as functions of dynamic pressure and weight, and consequently were time varying. Because the variables were to be interpolated from tables in the onboard computer, it was desirable to make their scheduling as smooth as possible so that the stability of the system at time points occurring between the tabulated points was predictable. Thus, the approach was to optimize the system at critical time points during flight. Then, the optimized variables were plotted with curves faired through the points. The remaining schedule was then optimized using the fixed control variable values as the "initial guesses." The objective function was to maximize $\Delta K_{\text{RIGID}}$ and bandwidth under maximum $M_a$ (pitch moment derivative with respect to angle of attack) conditions.

Figure 7 shows a before and after schedule of the notch filter damping coefficients. The initial design, which was "hand-tuned," had a very nonsmooth schedule. By allowing ADOCS to perform the iterations, the designer was able to direct some attention to smoothing the curves. This was accomplished as shown in Fig. 7. Figure 8 shows how the second mode stability margin varies over flight. As shown in the figure, the original autopilot design was unable to meet the required margins (this was the case for all three modes). The other two curves compare the margins obtained by a designer using ADOCS with the margin obtained by a designer "hand-tuning" the variable. Each designer was allowed the same number of days. Besides providing more stability, the ADOCS package increased the system's bandwidth and $M_a$ capability. Moreover, each designer required essentially the same amount of computer time.

**Example 3**

Example 3 is the design of a roll-yaw autopilot on a variable trajectory vehicle. Figure 9 shows some results of the ADOCS package applied to this multivariable system. The difficulties with multivariable systems arise from interactive effects (i.e., the action of one feedback loop affects the action of the others). There have always been discussions about the shortcomings of classical frequency-response design methods when dealing with multi-input multi-output systems. The standard practice used successfully by the "classical" engineer is simply to design all loops simultaneously. This iterative design approach includes an extensive tolerance analysis and yields optimum systems. ADOCS automates this classical approach.

On every step, ADOCS simultaneously considered 1) stability margins of the roll loop with the yaw loop closed; 2) stability margins of the yaw loop with the roll loop closed; 3) shape of the frequency response of roll due to command with all loops closed; and 4) shape of the frequency response of yaw due to command with all loops closed.

Disturbance rejection could also have been simultaneously considered by reducing the peak value of the frequency response of roll, yaw, and all fin angles due to a disturbance like the wind.

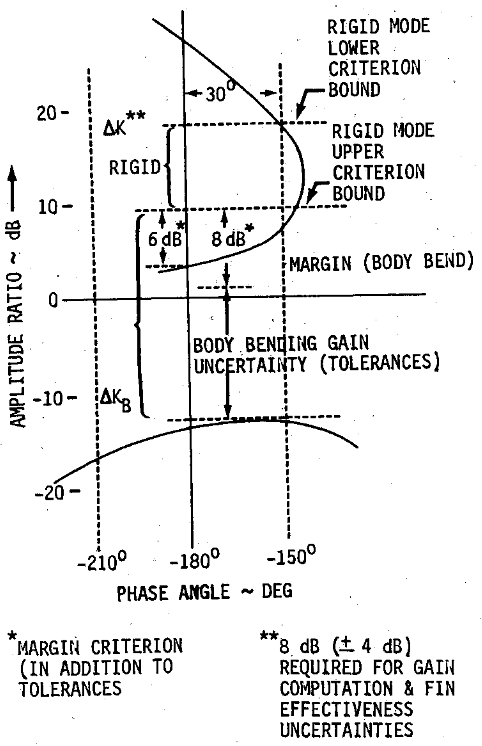
RIGID MODE LOWER CRITERION BOUND
RIGID MODE UPPER CRITERION BOUND
30°
ΔK** RIGID
6 dB*
8 dB*
MARGIN (BODY BEND)
BODY BENDING GAIN UNCERTAINTY (TOLERANCES)
ΔK_B
AMPLITUDE RATIO ~ dB
20
10
0
-10
-20
-210°
-180°
-150°
PHASE ANGLE ~ DEG
*MARGIN CRITERION (IN ADDITION TO TOLERANCES
**8 dB (± 4 dB) REQUIRED FOR GAIN COMPUTATION & FIN EFFECTIVENESS UNCERTAINTIES

Fig. 6 Missile stability requirements.

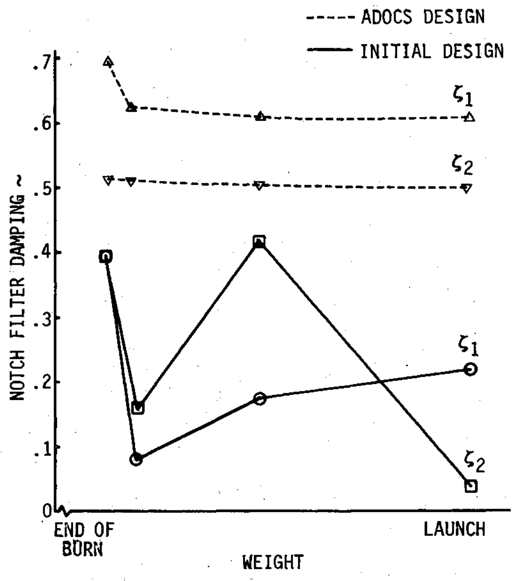

Fig. 7 Notch filter scheduling.

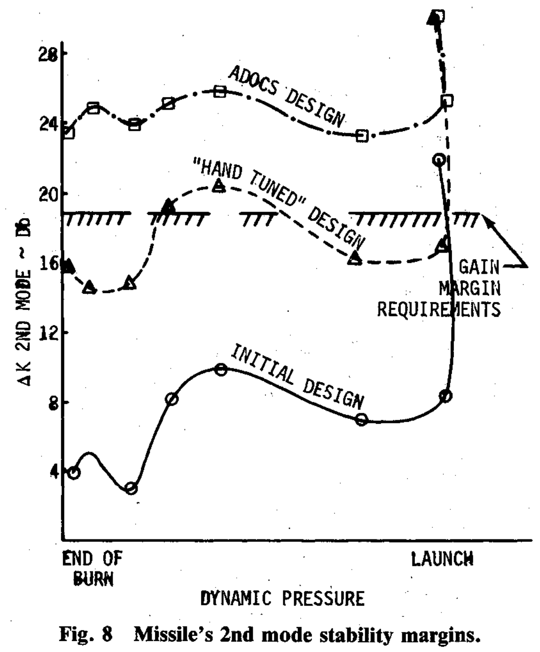

Fig. 8 Missile's 2nd mode stability margins.

| PHASE MARGIN (DEGREES) | | |
|---|---|---|
| | INITIAL | FINAL |
| ROLL OPEN, YAW CLOSED | 22. | 60. |

| FREQUENCY RESPONSE, PEAK RESONANCE (db), ALL LOOPS CLOSED | | |
|---|---|---|
| | INITIAL | FINAL |
| ROLL DUE TO COMMAND | 9. | 0. |
| YAW DUE TO COMMAND | 5. | 0. |

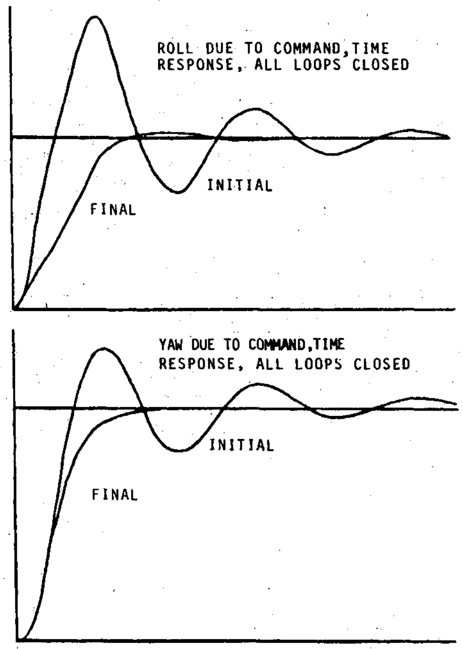

**Fig. 9 Variable trajectory vehicle optimization.**

## Summary

A computer program has been developed wherein a conventional and proven control system design method has been automated through nonlinear programming. The program replaces the engineer in the technically unchallenging iterative steps of the design process. This allows the engineer to concentrate on the most important jobs: developing system requirements, mathematical modeling, and choosing control system configurations. Because the program automates conventional design methodology, it is easily understood and incorporated into existing projects. In fact, results have been coded into the onboard software of aerospace vehicles that have actually flown.

## Acknowledgments

The authors would like to thank James H. Miller, Richard D. Jones, John F. Pflug, and John L. Tietze for their contributions to developing and implementing the ADOCS software package; also Virginia M. Riske and Susan E. Hatch for their contributions to document preparation.

## References

[1]Wilde, D.J. and Beightler, C.S., *Foundations of Optimiztion,* Prentice-Hall, Englewood Cliffs, New Jersey, 1967.

[2]Hadley, G., *Nonlinear and Dynamic Programming,* Addison-Wesley Publishing Co., Inc., Reading, Mass., 1964.

[3]"Proceedings of the Symposium on Recent Experiences in Multi Disciplinary Analysis and Optimization," NASA Conference Publication 2327, Parts 1 and 2, compiled by J. Sobieski, Langley Research Center, April 1984.

[4]Hauser, F.D., "A Nonlinear Programming Algorithm for the Automated Design and Optimization of Flexible Space Vehicle Autopilots," AIAA Paper 73-892, Aug. 1973.

[5]Dantzig, G.B., "Maximization of a Linear Function of Variables Subject to Linear Inequalities," *Activity Analysis of Production and Allocation,* John Wiley and Sons, Inc., New York, 1951.

[6]Miele, A., Calabro, A., Rossi, F., and Wu, A.K., "A Modification of the Sequential Gradient-Restoration Algorithm for Mathematical Programming Problems with Inequality Constraints," Aero-Astronautics Rept. 124, Rice University, Houston, 1975.