JOURNAL OF SPACECRAFT AND ROCKETS
Vol. 39, No. 3, May–June 2002

Optimized Solutions for Kistler K-1 Branching Trajectories Using Multidisciplinary Design Optimization Techniques

Laura A. Ledsinger* and John R. Olds†
Georgia Institute of Technology, Atlanta, Georgia 30332-0150

Fully reusable two-stage-to-orbit launch vehicle designs that incorporate branching trajectories during their ascent are of current interest in the advanced launch vehicle design community. Unlike expendable vehicle designs, the booster of a two-stage reusable system must fly to a designated landing site after staging. Because of a mutual dependence on the staging conditions, both the booster flyback branch and the orbital branch of the ascent trajectory must be simultaneously optimized to achieve an overall system objective. The optimum solution is often a compromise between the local objectives of the two branches. Current and notable designs in this class include the U.S. Air Force Space Operations Vehicle designs, the Kelly Astroliner, the Kistler K-1, and NASA's proposed liquid flyback booster designs (space shuttle booster replacement). Solution techniques are introduced that are well suited to solving this class of problem with existing single-segment trajectory optimization codes. In particular, these methods originate from the field of multidisciplinary design optimization and include optimization-based decomposition and collaborative optimization. The results of applying these techniques to the branching trajectory optimization problem for the Kistler K-1 launch vehicle are given and conclusions are drawn with respect to computational efficiency and quality of the results. In general, partial optimization-based decomposition was preferred due to its superior robustness, ease of setup, fast execution time, and optimality of the results.

Nomenclature

g_i = system-level constraint, compatibility constraint
J_i = local error function used in collaborative optimization
x = feedforward or feedback coupling variable between trajectory branches
x' = intermediate or guessed value of x

Introduction

TO lower costs, designers of advanced two-stage-to-orbit (TSTO) launch vehicles are considering launch systems in which the booster stage can be recovered, serviced, and reflown. Often the reusable booster is required to land at a predesignated recovery site either near the original launch site [return-to-launch-site (RTLS) trajectory; Fig. 1] or downrange of the staging point. In these cases, the entire trajectory is composed of three parts. The ascent portion follows the mated vehicle from launch to staging. At this point, the trajectory is assumed to split into two branches. One is the orbital branch beginning at staging and following the orbital upper stage to orbit. The second branch, or flyback branch, starts at staging and follows the reusable booster to its landing site. Because of long recovery distances or required out-of-plane maneuvers, the booster is often powered for its flight to the landing site.

In general, both the orbital branch and the flyback branch rely on the (mated) ascent trajectory for their respective initial conditions. These initial conditions form a state vector comprising geographical position, altitude, velocity, flight-path angle, velocity azimuth, and possibly staging weight. In the most complex cases, the ascent trajectory will also depend on the results of both subsequent branches. When it is assumed that the booster is powered, the amount of flyback propellant required by the booster influences the gross liftoff weight of the vehicle and, thus, the ascent path. The weight of the upper stage (which is dependent on initial staging conditions and the payload) also affects the gross liftoff weight of the vehicle and, thus, the ascent path. Consequently, all three segments of the entire trajectory are nonhierarchically coupled or interdependent (Fig. 2). In certain numerical simulations, it may be computationally convenient to combine the ascent trajectory and the orbital branch to create one computer job, as is the case with the Kistler K-1. In this case, the problem can be reduced to two nonhierarchically coupled segments.

Optimization of branching trajectories differs greatly from that of single-segment trajectories. The fact that there are two, or even three, different parts of the overall branching trajectory makes the optimization more complex. The mutual dependence of the branches on the staging condition typically means that compromises must be made between the orbital and flyback branches. For example, an upper stage typically prefers a larger flight-path angle at staging. A steep flight-path angle helps it reach its orbital destination in a shorter amount of time and, thus, aids in maximizing the payload delivered to orbit. At the same time, a typical booster prefers a smaller flight-path angle at staging. The closer the velocity vector is to the horizontal, the faster the booster can achieve the negative flight-path angle that is needed to aim the vehicle back to the Earth. Thus, a shallow flight-path angle helps to minimize flyback propellant. It is evident that a compromise is needed between these two local objectives when the overall trajectory is considered.

In many trajectory optimization problems, the overall objective is to maximize the payload to orbit for a given launch mass. If the problem has a powered flyback branch, a further constraint is that the booster have sufficient flyback propellant to safely reach its landing area. Carrying excess flyback propellant will reduce the vehicle's payload delivery capability to orbit. Insufficient flyback propellant is equally unacceptable. Therefore, correctly calculating the flyback propellant requirement (with any required performance reserve margin) and providing that information to the overall gross mass calculation at the beginning of the ascent segment represents a feedback link from a downstream simulation to an upstream simulation and introduces the added complexity of nonhierarchic coupling to this class of branching trajectories.

Traditional Solution Methods

Unfortunately, a common method currently used in industry for optimizing a branching trajectory problem (henceforth the

Received 19 January 2001; revision received 26 October 2001; accepted for publication 29 October 2001.

*Ph.D. Candidate, Space Systems Design Laboratory, School of Aerospace Engineering; currently Senior Member of the Technical Staff, Space Architecture Department, The Aerospace Corporation, P.O. Box 92957, M4/940, Los Angeles, CA 90009-2957. Member AIAA.

†Associate Professor, Space Systems Design Laboratory, School of Aerospace Engineering. Senior Member AIAA.

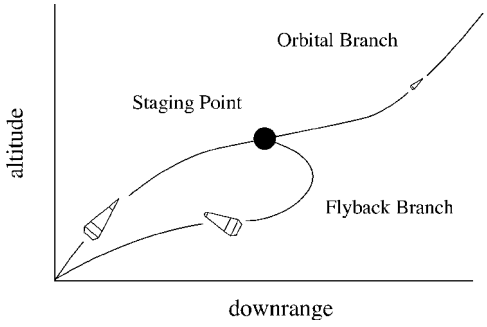

Fig. 1 RTLS branching trajectory.

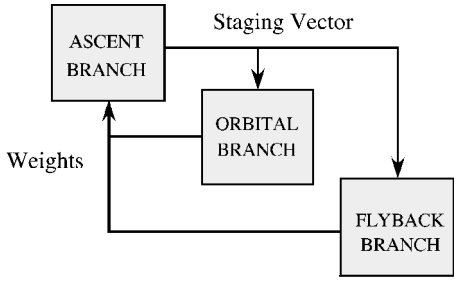

Fig. 2 Nonhierarchic coupling for a typical branching trajectory.

one-and-done method), although recognizing the feedforward coupling of the ascent trajectory to the orbital and flyback branches, ignores the flyback propellant feedback from the flyback branch to the ascent trajectory. The ascent trajectory, orbital branch, and flyback branch are treated as separate but sequential optimization subproblems. A reasonable guess at upper stage mass, flyback propellant, and associated structure is made to establish an initial booster weight. Then the ascent is optimized for maximum weight at staging (or some similar criterion). The ascent trajectory will produce a staging state vector used to initiate the orbital branch and the flyback branch. This vector includes altitude, velocity, flight-path angle, velocity azimuth, latitude, longitude, and sometimes staging weight. The orbital branch will typically be optimized with respect to maximizing the upper stage burnout weight, whereas the flyback branch will typically be optimized with respect to minimizing the flyback propellant consumed.

There are a number of deficiencies in the one-and-done method. A major deficiency is that the final solution is not internally consistent. In other words, it is not guaranteed to be converged between the subproblems. For example, the initial estimate for upper stage payload used to begin the process may not match the calculated payload determined by the subsequent orbital branch. Equally important, the booster flyback propellant feedback is not present. Because the analyst must guess the flyback propellant required before optimizing the ascent branch, the potential for having either excess or insufficient flyback propellant is high.

If time allows, the aforementioned deficiencies can be eliminated through simple iteration between the ascent, upper stage, and the flyback branches. That is, the results from the downstream solutions can be used to update the initial guesses used to start the ascent trajectory, and the process can be iterated until the coupling variables are converged. From this point on, this method will be referred to as the manual iteration method. The manual iteration method is common in practice to eliminate internal consistency problems between segments of a nonhierarchically coupled branching trajectory. However, a significant deficiency still exists with this method as with the one-and-done method.

At a fundamental level, these methods are inherently flawed. The objective functions of the subproblems are not the same; therefore, they can be in conflict. If the system-level objective is to deliver the maximum payload to orbit with a given size booster, then why should one expect an optimum solution from a method that first determines a staging condition by maximizing the payload to orbit for the orbital branch, then minimizes the flyback propellant from that point for the flyback branch? Assuming the ascent and flyback propulsion systems draw their propellants from a common tank, could not a compromise in the staging condition be made such that the flyback propellant is reduced, and thus the booster propellant available for ascent is increased? A rigorous solution to this problem requires simultaneous and coupled treatment of all branches of the trajectory and the establishment of a single, consistent objective function between them, that is, a system-level optimization.

Branching Trajectory Optimization in the Launch Vehicle Community

Many in industry have recognized the deficiencies of the one-and-done and manual iteration methods. Some have employed trajectory optimization tools that solve the branching trajectory problem as a nondistributed problem. For example, the computer code OTIS has the ability to simulate the entire branching trajectory as a single simulation.[1] However, descriptions and results for applications of OTIS to nonhierarchically coupled branching trajectories are not currently available in the public domain, that is, branching trajectories in which the results of a downstream branch must be used to update the initial conditions of the ascent branch.

Shuttle inertial upper stage (IUS) trajectories with branches have been simulated by previous researchers.[2] An ascent trajectory was simulated for the orbiter from launch to an intermediate parking orbit. From this orbit, trajectories required for the shuttle deorbit and IUS mission were generated. The overall problem was a parameter optimization problem with variables that corresponded to the magnitude of the shuttle and IUS propulsive maneuvers and constraints for the mission requirements. An overall optimization using a sequential quadratic programming algorithm was performed to satisfy the mission constraints of the two branches while maximizing the payload weight of the IUS. Although these trajectories differ from the branching trajectory definition of the Introduction, the solution with an overall optimizer found that compromises in shuttle deorbit requirements and IUS performance were necessary to find the desired system optimum of maximum payload. Feedback of the propellants was not considered in Ref. 2.

POST[3] has been used by other researchers to solve some branching trajectories. Branching trajectory research has included investigations of a bimese-type two-stage launch vehicle with a glideback, or nonpowered, RTLS booster.[4] Separately optimized ascent and glideback POST subproblems were used for these nonhierarchically coupled simulations.

A Sänger-like vehicle with an orbiter and a ramjet-powered RTLS booster has also been investigated.[5] The study was used to analyze the effects of various amounts of airbreathing and rocket propulsion during ascent. Thus, optimization for an overall objective was not the goal of the analysis and was not addressed. The trajectory simulations for the ascent, orbital, and flyback branches were run separately with the staging conditions (including altitude, velocity, and flight-path angle) fixed for staging at Mach 6. The sizes of the booster and the orbiter were also fixed. Internal propellant volume in the booster could vary, however. While trying to find the booster gross weight needed to lift the orbiter, cruiseback propellant weight was estimated. Feedback of the actual flyback, or cruiseback, propellant weight required was not modeled, but would have been beneficial because the cruise distance and booster staging weight changed for each different gross weight.

Trajectory Optimization with POST

POST I, widely used in conceptual launch vehicle design, is not capable of simultaneously treating and optimizing all parts of a branching trajectory. POST I is a generalized event-oriented trajectory analysis code that numerically integrates the equations of motion of a flight vehicle given definitions of aerodynamic coefficients, propulsion system characteristics, atmosphere tables, and gravitational models. Guidance algorithms used in each phase are user defined. Numerical optimization is used to satisfy trajectory constraints and minimize a user-defined objective function by changing independent steering and propulsion variables along the flight path. POST I runs in a batch execution mode and depends on an input file (or input deck) to define the initial trajectory, event structure, vehicle parameters, independent variables, constraints, and objective function. Three-degree-of-freedom and six-degree-of-freedom

versions are available to qualified researchers. Multiple objective functions and simultaneous trajectory branches cannot currently be defined in POST I.

The POST I sequel, POST II,[6] is currently being distributed to qualified researchers on a limited basis for beta-test, or prerelease testing, but was not available for this study. Like OTIS, POST II can simulate multiple vehicles and thus branching trajectories as a single nondistributed trajectory simulation. At this time, flyback propellant feedback is not a direct option of the code, but has the potential to be included as part of the code's user-defined calculations.

Overview of the Present Research

For this research, the optimization of nonhierarchically coupled branching trajectories has been solved using the three-degree-of-freedom version of the POST I code. One of the ground rules of the present study was to develop a solution approach to this problem that could be introduced into the design community without a significant change to the incumbent toolset. The incumbent toolset is assumed to be limited to a single-segment trajectory optimization code. The traditional solution approaches of the one-and-done method and manual iteration method used for comparison rely on at most three separate POST input subproblems: one for the ascent trajectory subproblem, one for the orbital branch subproblem, and one for the flyback branch subproblem. Each subproblem is a separate computational job, with its own input files, independent variables, constraints, and objective function. The multidisciplinary design optimization (MDO) approaches introduced in this paper have retained the POST I code and the use of at most three separate computational jobs (one job for each subproblem or branch) but eliminated any objective function conflict and lack of data consistency between jobs. As a result, a solution with internally consistent data (the feedback propellant is reflected in the initial gross weight, etc.) and with a single system-level objective function (without conflicting objective functions for each subproblem) has been obtained.

Kistler K-1

To apply this research, the missions of candidate TSTO launch vehicle designs were chosen to serve as reference missions. In this paper, only the analysis and results for the Kistler K-1 are presented. Ledsinger gives a second example for a small payload horizontal takeoff two-stage vehicle with an airbreathing booster and a rocket upper stage.[7]

Many reusable launch vehicles are currently being developed by commercial industries with the goal of capturing a profitable share of the growing satellite launch market. Such is the case for the Kistler K-1 launch vehicle.[8] The K-1 (Fig. 3) (Ref. 9) will be a fully reusable, two-stage vehicle that incorporates branching trajectories. The vehicle's booster will use three Aerojet-modified Russian NK-33 engines, and the upper stage will be propelled by one Aerojet-modified NK-43 engine.[10,11] There will be different versions of the vehicle to accommodate various payload classes. One of its missions will be to deliver a 3400-lb (1542-kg) payload to a 51 n mile (94.5 km) × 486 n mile (900 km) × 51 deg orbit. This mission will be analyzed in this study. The data (stage weights, trajectory constraints, engine data, etc.) pertaining to that mission were provided directly by Kistler Aerospace.

The trajectory events of the K-1 launch vehicle are shown in Fig. 4. The K-1 trajectory is an RTLS branching type trajectory as in Fig. 1. After launch from the site at Woomera, Australia, the K-1 booster will power the vehicle until staging approximately 120 s later. After staging, the booster performs a pitcharound maneuver that will guide itself back to within 10,000 ft (3048 m) of the launch site, to land with airbags and parachutes. The upper stage will continue on to the designated orbit. For the purposes of this study, the simulation will end when the orbital insertion conditions have been attained. In reality, the K-1's upper stage will deorbit and return to the launch site.

For simplicity, the K-1 trajectory has been simulated through two POST subproblems. The first follows the vehicle from launch to orbital injection of the upper stage. Note that this specific simulation combines the ascent (to staging) and orbital paths (beyond staging) into a single job. The second, or flyback branch, follows just the booster from staging to its RTLS. The independent variables of reference K-1 ascent subproblem are 12 pitch angles spaced along the trajectory at various times and the payload weight. The actual vehicle pitch is linearly interpolated between these 12 control points. The ascent has five constraints including orbital insertion criteria (target altitude, flight-path angle, inclination, velocity) and nominally maximizes the payload for a given set of propulsion characteristics, vehicle aerodynamics, K-1 weights, and ascent propellant.

Fig. 3 K-1 launch vehicle.

The reference booster flyback subproblem uses six independent variables: four pitch angles, azimuth of the pitchover maneuver needed initially to head the vehicle in a direction back to the launch site, and engine burn time. Two constraints guarantee a smooth rocket pull up at staging and landing within a certain downrange distance. Given a set of engine propulsion characteristics, aerodynamics, and the staging conditions, the flyback trajectory nominally tries to minimize flyback propellant weight. The staging conditions from the ascent branch include altitude, flight-path angle, latitude, longitude, velocity, and velocity azimuth.

Solution Approach

Solution techniques from the field of MDO were advantageously applied to the branching trajectory problem posed as a coupled set of subproblems. Table 1 lists the characteristics of the MDO solution techniques that were used: fixed-point iteration (FPI), two variations of optimization-based decomposition (OBD), and collaborative optimization (CO). Partial OBD (PODB) is a technique in which only the feedback links between subproblems are broken by introducing compatibility constraints and intermediate variables to the system-level optimization problem. For example, a coupling variable x from a downstream analysis is passed directly to the optimizer instead and replaced with a new intermediate variable x' under the control of the optimizer. This new intermediate variable serves as the guess into the upstream discipline, and thus, the need for iteration is eliminated. Compatibility constraints ensure that the actual coupling variable and the intermediate variable are the same at the final solution.

Full OBD (FOBD) uses this same decomposition strategy to break both feedforward and feedback links, resulting in a completely parallel subproblem execution. CO is a multilevel optimization scheme. These methods have been used successfully by other researchers for preliminary aircraft design[12] and launch vehicle design.[13] The FPI method is a single-level, serial execution technique among the

Table 1 Solution techniques to branching problems

Method	Internally consistent data	Iteration between branches	Conflicting objective functions	System-level optimizer	Subproblem execution	Optimizer strategy
Manual iteration	Yes	Yes	Yes	No	Sequential	Distributed
FPI	Yes	Yes	No	Yes	Sequential	System level (large)
POBD	Yes	No	No	Yes	Sequential	System level (very large)
FOBD	Yes	No	No	Yes	Parallel	System level (extremely large)
CO	Yes	No	No	Yes	Parallel	Distributed

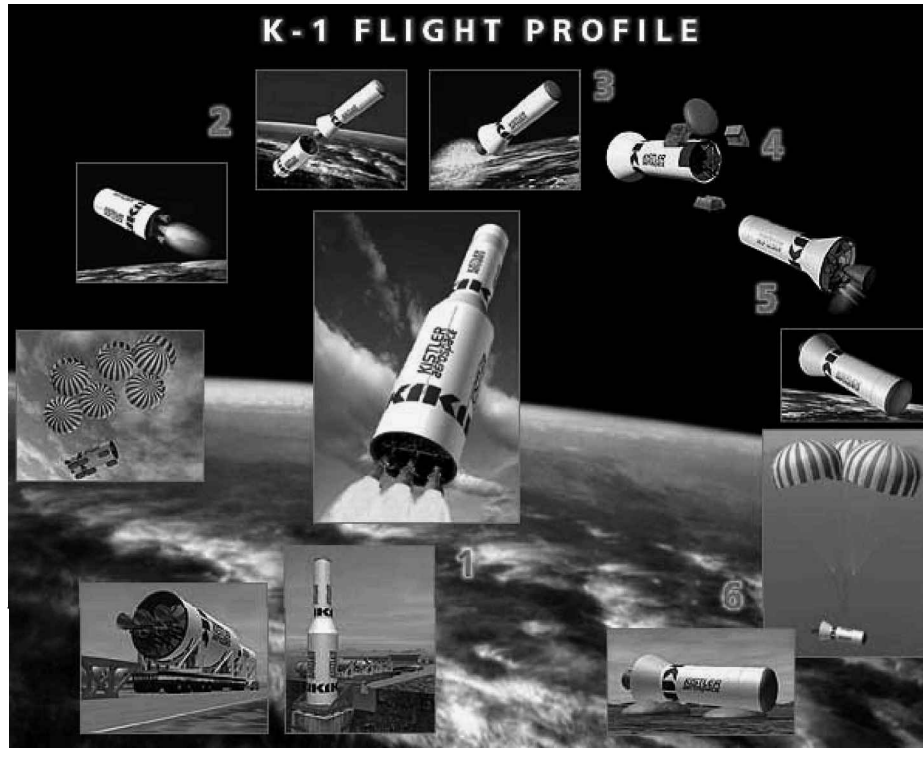

Fig. 4 K-1 trajectory.

subproblems that uses an overall system optimizer. Ledsinger provides supporting detail on the formulation and characteristics of each of these MDO techniques for the specific problem considered here. The reader is referred to Ref. 7 for more information. In addition, the earlier introduced manual iteration method is included for comparison.

Note that there are many metrics against which to optimize the trajectories of both the upper stage and the booster. In this research, all of the methods analyzed for the K-1 had a system-level objective of maximizing orbital payload weight. For the K-1 simulation, fixed-stage inert and propellant weights were used for all weights except for booster ascent propellant weight, flyback propellant weight, and payload weight. The sum of the booster ascent propellant and the flyback propellant was assumed to be constant; thus, an increased flyback propellant requirement would reduce booster ascent propellant and vice versa. This flyback propellant requirement creates a feedback link back to the ascent branch. Note that, in the actual K-1 configuration, the ascent booster propellant and the flyback propellant are separately maintained and are not interchangeable once the vehicle is past the design phase.

Results for the Kistler K-1

One-and-Done Method

The one-and-done method does not account for the iterative, coupled nature of the ascent and flyback branches. The data extraction and insertion from the ascent POST subproblem to the flyback subproblem were performed manually. The results for this method appear in Table 2. The solution for this method will be the starting point for all of the methods following this one. As a result, computational time is not listed for this method. The main reason to show the results of this method is to highlight the large difference in the objective function (recall that the goal is to maximize payload weight) that can be achieved when iteration occurs to converge the coupling variables between the branches.

For one-and-done method, the initial guess for booster ascent propellant is extremely significant. The percentage of booster ascent propellant with respect to the total available assumed for this simulation was 92.88%. This assumption left 7.12% for the flyback propellant. After the serial execution of the two POST subproblems, it was found that several hundred pounds initial flyback propellant was left over, or not used, even after considering the required margins. Only 6.08% of the original propellant was needed for flyback for this unconvergred case. Making some of this propellant available to the ascent branch could improve payload delivered to orbit. On the other hand, if the initial guess for booster ascent propellant percentage was too high, then the possibility of not having enough flyback propellant would have existed. This scenario highlights an example of one of the many deficiencies of this method.

As already stated, the next method and the MDO methods will all begin with the solution to the one-and-done method. Thus, the initial guesses for the following methods are as follows: percentage of booster ascent propellant 93.92%, percentage of flyback propellant 6.08%, and payload weight 3314.79 lb (1503.6 kg).

Manual Iteration Method

The manual iteration method uses local optimization in both subproblems and no system-level optimizer. Execution is sequential and iterative between the ascent subproblem and the flyback subproblem. Flyback propellant (and therefore the split with ascent booster propellant) and payload from the end of both simulations are

Table 2 One-and-done and manual iteration method results for K-1

Method	Payload weight, lb (kg)	POST CPU time, s	Iterations
One-and-done	3315 (1503.6)	——	0
Manual iteration	3529 (1600.7)	201	6

Table 3 Size of system optimizer for the K-1 branching case

Method	Variables	Constraints
Manual iteration	——	——
FPI	19	10
POBD	20	12/11
FOBD	26	24
CO	8	2

provided to initialize the ascent subproblem of the subsequent iteration. The resultant staging state vector from the ascent subproblem is used to initialize the subsequent flyback subproblem. The booster weight at staging is also updated as the iterations occur. Again, the data extraction and insertion steps are manual. Iteration information and execution time results are shown in Table 2. The convergence criterion for the manual iteration method was flyback propellant weight. The K-1 iteration was considered converged when the change in this variable was less than 0.01% of the result from the previous iteration. Six iterations were required. Note that the payload result of 3529 lb (1600.7 kg) is about 129 lb (58.7 kg) higher than the advertised capability of the K-1 to this orbit. This result will be used as a comparison case in the MDO method assessment. Whereas iteration was performed between the two basic subproblems to ensure data consistency (unlike the one-and-done method), the conflicting objective functions between the two branches were not addressed. That is, the ascent branch was configured to maximize orbital payload, whereas the flyback branch was configured to minimize flyback fuel.

MDO Methods

The MDO methods of FPI, OBD, and CO all require the introduction of a system-level optimizer. The size of the system-level optimization problem for each of the MDO methods is listed in Table 3. The system-level optimizer used for all of the MDO methods was the modified method of feasible directions implemented by the software program DOT™. For the FPI and OBD methods, the gradients for the system-level optimizer were calculated using central finite differences. The entire process for each MDO method, including data extraction and insertion and gradient calculation, was automated using PERL and C++ codes.

DOT requires that equality constraints be formulated as two separate inequality constraints. Three of the trajectory constraints for the K-1 POST subproblems were equalities that, when reformulated as two inequality constraints, brought the total number of system-level constraints for the FPI method to 10. Note that for the POBD method there were either one or two more constraints in addition to those of the FPI method. In this method, a new intermediate variable representing the guessed flyback propellant is created in the system-level optimizer and passed to the ascent trajectory simulation when needed. A system-level compatibility constraint is used to ensure that the guessed flyback propellant, x', and the actual flyback propellant from the flyback branch trajectory simulation, x, are identical at the final solution. The compatibility constraint g, the difference between x and x' being zero, can be represented as either a single quadratic inequality constraint [Eq. (1)] leading to 11 total constraints or 2 linear inequality constraints [Eqs. (2) and (3)] leading to 12 total constraints:

$$g = (x' - x)^2 \leq 0 \tag{1}$$

$$g_1 = (x' - x) \leq 0 \tag{2}$$

$$g_2 = (x - x') \leq 0 \tag{3}$$

The compatibility constraints for the FOBD method were posed as 7 pairs of linear inequality constraints, thus the total number of constraints was 24 (14 above the 10 present in the FPI method). The coupling variables between the two POST simulations are given in Table 4. The ODB method is also referred to as a simultaneous analysis-and-designmethod.

FPI

The FPI method involves use of a system-level optimizer and POST subproblem iteration. The individual POST subproblems are not locally optimized for this method. The individual POST subproblems are used simply to integrate the equations of motion using the set of controls given by the system-level optimizer. Thus, this method differs fundamentally from the manual iteration method used for comparison. In FPI, the approach used to eliminate the conflicting local objective functions is to elevate all of the trajectory variables for both branches to the system level where the single objective is to maximize payload weight delivered to orbit. This method is also called the all-at-once method or nested analysis and design for this reason. FPI ensures internal data consistency among the coupling variables used between the two POST jobs by iterating between them for each function call from the system-level optimizer. The POST subproblem iterations are considered converged when the flyback propellant weight was within 0.01% of its previous value, just like in the manual iteration method.

Detailed quantitative results can be seen in Tables 5 and 6. The FPI method gave an optimized solution of 3544 lb (1607.5 kg) of payload weight in 16.3 CPU min with 18 system-level iterations on an SGI Octane computer with a 250-MHz R10000 CPU. Figure 5 shows how the payload weight varied with each function call. Figure 5 also shows the number of method of feasible directions iteration counter, the system-level line searches needed to move toward the solution, vs function calls. Several function calls are needed to complete a full system-level iteration. Figure 6 shows the constraint convergence history for the FPI method, plotting the logarithm of the norm of the nonnegative constraints at the end of each system-level iteration. Convergence for this and subsequent methods was considered reached when the change in payload was less than 0.5 lb (0.2 kg) or 0.1% from the previous iteration.

Table 4 Coupling variables between two branches

Parameter	Ascent subproblem	Flyback subproblem
Flyback propellant weight	Input	Output
Altitude	Output	Input
Velocity	Output	Input
Azimuth of velocity	Output	Input
Flight-path angle	Output	Input
Latitude	Output	Input
Longitude	Output	Input

Table 5 K-1 results comparison (MDO)

Method	Optimized payload weight, lb (kg)	POST computational time, s
FPI	3543.64 (1607.37)	976.8 (16.3 min)
POBD 1	3566.82 (1617.88)	939.5 (15.7 min)
POBD 2	3566.85 (1617.90)	941.4 (15.7 min)
FOBD	3585.06 (1626.16)	1808.5 (30.14 min)
CO	3569.04 (1618.89)	6627.0 (1.84 h)

Table 6 K-1 detailed results comparison (MDO)

Method	POST calls	System-level iterations	Average CPU time/ function call
FPI	884	18	1.105 s
POBD 1	843	20	1.114 s
POBD 2	843	20	1.117 s
FOBD	1622	34	0.911 s
CO	2271/4745	7	1.326 min

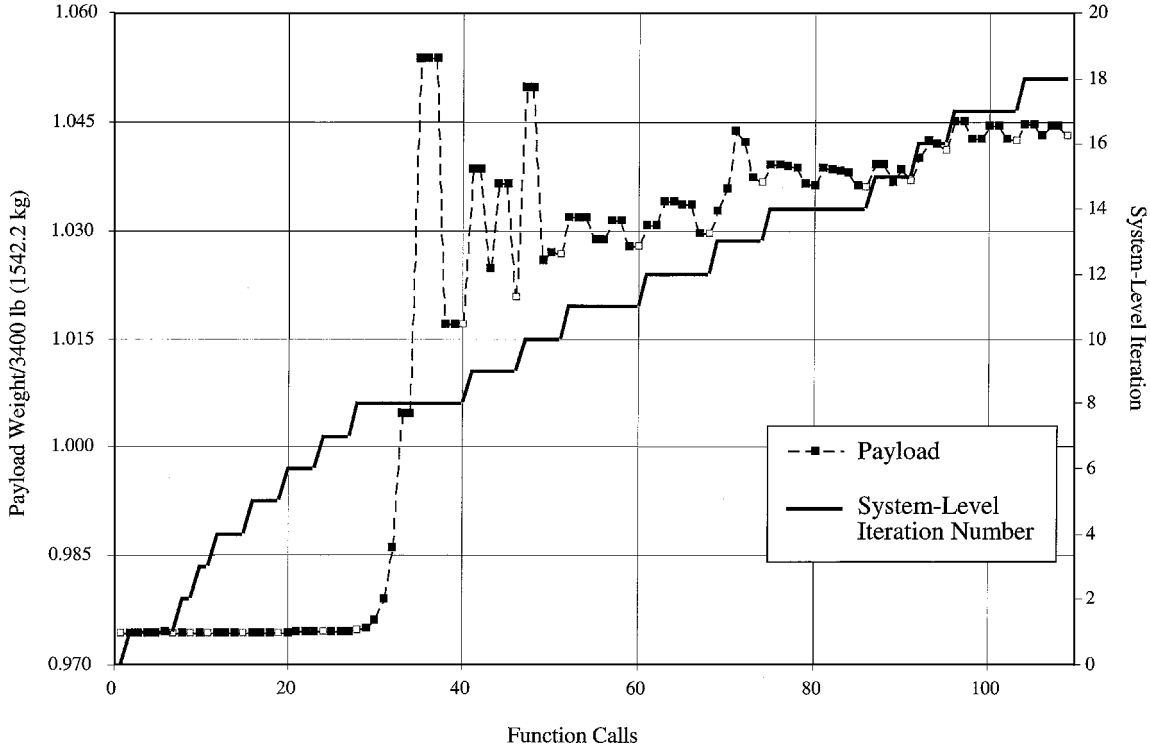

Fig. 5 Payload weight and system-level iterations vs function call for FPI method.

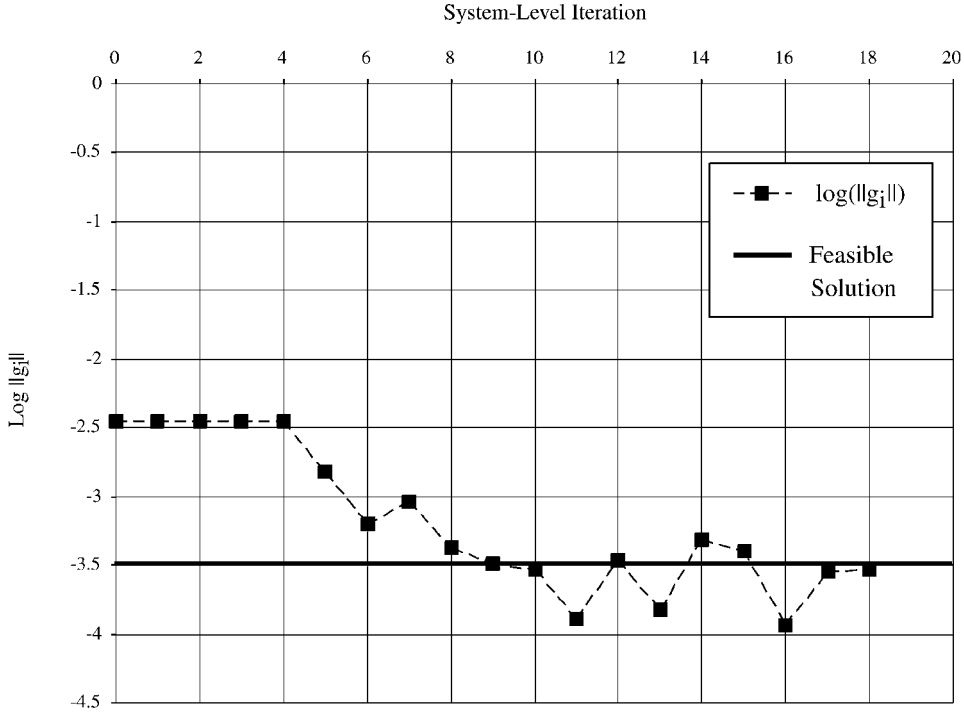

Fig. 6 Active constraint history per system-level iteration for FPI method.

OBD

Like the FPI method, the POBD method introduces a system-level optimizer, and the individual POST subproblems are not locally optimized. However, the feedback loops are eliminated by this method with a compatibility constraint as already discussed. The POBD method resulted in an optimized payload weight of 3567 lb (1617.90 kg) in 20 system-level iterations requiring 15.7 min. Additional numerical results are listed in Tables 5 and 6.

The FOBD method also involves the use of a system-level optimizer and POST subproblems that are not optimized. For this method, all of the coupling variables in Table 4 are replaced with intermediate variables (guessed values controlled by the system-level optimizer) and compatibility constraints so there are neither feedback nor feedforward links, and the two POST subproblems can be executed in parallel if desired. For the FOBD method, the optimization had to be restarted once due to a lack of progress from the original starting point. After this restart from a new starting point, an optimal solution of 3585 lb (1626.16 kg) of payload weight was found. This method took about 16 CPU min total and 34 system-level iterations for the entire problem. More quantitative results are listed in Tables 5 and 6.

CO

The CO method[7,12] involves the use of a system-level optimizer and a parallel analysis structure. However, for this multilevel optimization scheme, the POST subproblems are locally optimized

using the NPSOL optimizer included in the POST software, allowing the size of the system-level optimizer to be substantially reduced (Table 3). There were eight target variables required for this method. These included the payload and flyback propellant weights and the six other variables that comprised the staging state vector. Target variables are analogous to intermediate variables in the OBD methods and are independent variables under control of the system-level optimizer. However, unlike intermediate variables, the local subproblems do not use the target variables directly but rather try to match their local versions of the coupling variables to the targets supplied by the system-level optimizer.[7,12] The two system-level constraints J were calculated using a sum of the squared errors formulation between the target variables and the local versions in each POST subproblem. System-level constraint gradient calculations were performed using postoptimality sensitivity analysis.[14] At the solution, the system-level constraints will be zero, indicating that the targets and the local versions of the coupling variables are consistent for each subproblem. Note that practical[15] and theoretical[16] issues with the convergence of CO when using a sum-squared formulation for local J have been observed by some researchers, but this approach has been shown to be acceptable for some problems.[13]

The objective function, payload weight, was also included as a target variable (independent variable at the system level). Consequently, the objective function gradient was easily and analytically derived. The target variable for payload weight was passed to the ascent subproblem, where it and the actual payload weight from the POST simulation were used in the calculation of the error function J_1 for the ascent.

The CO method gave an optimal solution of 3569 lb (1618.89 kg) of payload weight. This was attained in six system-level iterations requiring 1.84 CPU h. More results are given in the next section. Figure 7 shows how the payload weight (both the system-level target and the actual payload calculated by the ascent branch) changed per system-level function call. Figure 8 shows that the ascent subproblem is able to match the target payload weight for the first three system-level function calls shown (1–3). Indeed, the ascent

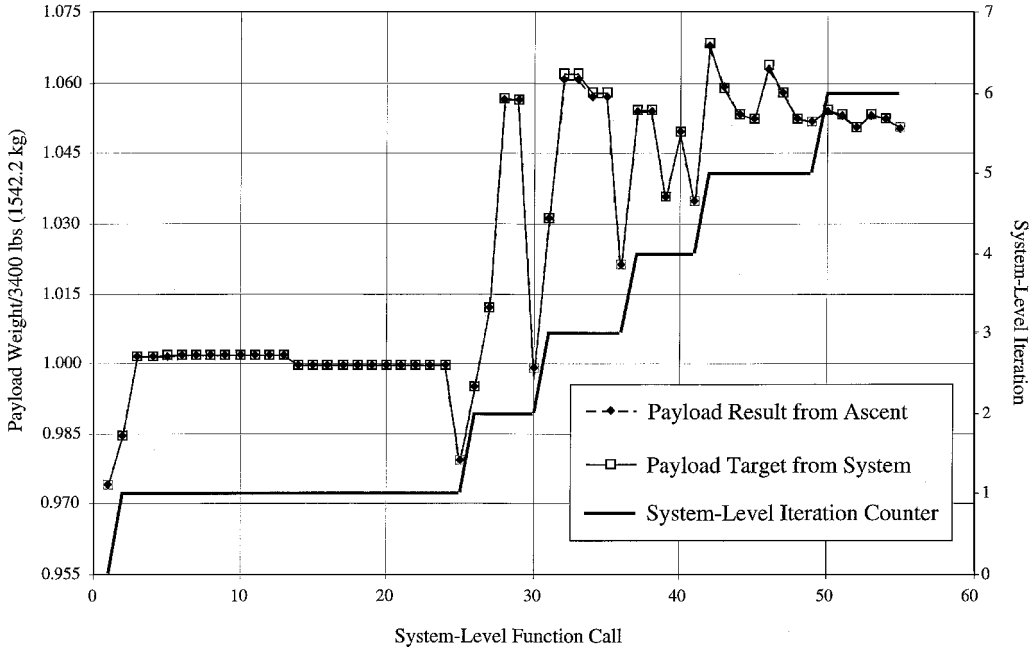

Fig. 7 Payload weight vs system-level function call for the CO method.

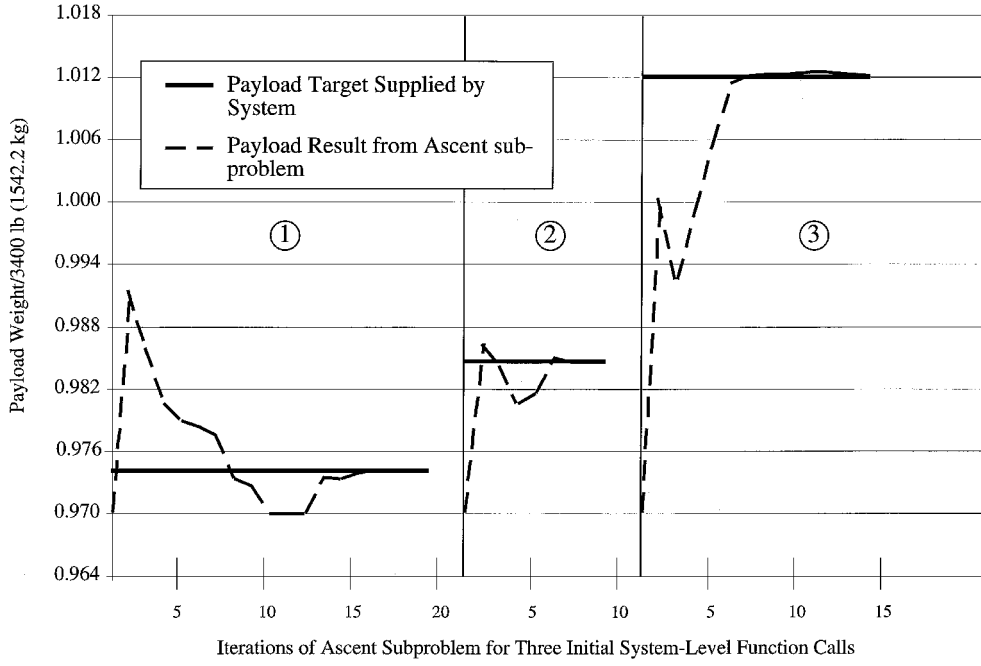

Fig. 8 System-level coordination for payload weight.

subproblem had little trouble matching the target payload for all function calls. Note also that the system-level optimizer increased the target payload weight at every iteration to maximize its objective function. Earlier noted convergence issues were not observed for this problem.

Results Summary

Tables 5 and 6 give quantitative results for the Kistler K-1 MDO methods. In Table 5, POST computational time refers to the total amount of time to run the POST subproblems for all calls, including function calls and gradient calls.

As indicated in Table 5, all MDO methods increased the payload weight of the K-1, relative to the manual iteration method [3529 lb (1600.7 kg); Table 2]. The FPI method improved payload by about 15 lb (7 kg) over the manual iteration method. The POBD methods further improved the payload weight of the K-1 by approximately 38 lb (17 kg) over the manual iteration method (results from both formulations of the compatibility constraint are shown with POBD 2 representing the separate inequality constraint formulation). The FOBD method results in about a 56-lb (25.5-kg) increase in the payload weight over the manual iteration method. Use of the CO method increased the payload by approximately 40 lb over the manual iteration method. The MDO methods gave an approximate 40–55-lb (18–26-kg) increase over the manual iteration method, a 1.5% increase, which is relevant considering the high costs per pound to launch payloads.

It was expected that the FPI, POBD, FOBD, and CO methods would result in the same objective function. That is, all four of these MDO methods are designed to produce an overall optimization problem that does not have any of the conflicting local objective functions that are apparent in the manual iteration method and does not have the internal data consistency problems (lack of convergence) of the one-and-done method. No local extrema are known to exist in the test problem. However, the implementation strategy is different among the MDO methods.

The suboptimal payload weight produced by the FPI method is attributed to the numerical noise introduced by the internal iterations required between the two POST subproblems for each function call from the system-level optimizer. This problem is known to hamper FPI solutions in general. As the solution is approached, the accuracy of the gradients required to advance the solution becomes very critical. Poor gradients in the objective function and constraints will cause the optimizer to stop prematurely. Tightening the convergence tolerance on the flyback propellant in the internal POST iteration might improve the situation, but would obviously increase the computational effort required to solve the problem.

The marked difference in the payload found by the FOBD method relative to the POBD methods may partially be attributed to the existence of some degree of numerical slack allowed for intermediate variable values in the staging vector via the enforcement of compatibility constraints. For example, in the FODB methods, a feedforward staging vector value such as velocity is under the control of the optimizer as an intermediate variable. Within the tolerance allowed by the compatibility constraint, the intermediate velocity can be slightly reduced at the system-level and provided to the flyback subproblem to the overall benefit of system. Tightening the tolerances on the compatibility constraints will likely bring the POBD and FOBD methods into closer agreement by reducing the payload found by FOBD, but with a still further increase in computational effort required for the FOBD method.

Similarly, the CO method can take advantage of any numerical slack in the construction of its local error functions to improve the system performance. Tightening the tolerances at the local level will eliminate this extra degree of freedom, but this method is already much more computationally demanding than the other MDO alternatives for this problem and requires a more complex setup. CO is typically favored in multidisciplinary problems where distributed, even remote, execution of the individual subproblems is needed or when the problem size makes it prohibitive to elevate all of the local independent variables to the system-level. For this problem, the application of PODB produces a manageable number of design variables and constraints at the system level (Table 3) and is a relatively straightforward implementation.

Some definitions for the columns in Table 6 are required. The second column, POST calls, refers to the cumulative times a trajectory analysis evaluation occurred, including system-level gradient (except for the CO method, which were analytic), line search evaluations, and internal FPIs. Note that the two POBD methods are the most efficient by this metric. FOBD requires nearly twice as many function calls and thus nearly twice the solution time (Table 5). There are two POST call numbers given for the CO method. The first is the number of POST calls to the ascent subproblem, the second is for the flyback subproblem. As would be expected, the number of POST calls for the CO method is significantly larger than that for the other methods (7016 total). Although the number of system-level iterations is smaller for CO, many POST calls within the subproblems at each iteration were required.

In the fourth column, CPU time refers to the average time it took for one system-level function call to run, be that iteratively, sequentially, or in a parallel manner. At first glance, the average CPU times per function call listed in Table 6 are not what one would expect. The average time for the FPI method is expected to be larger than that for the POBD methods because the internal iterations for convergence are many. For the Kistler case and the baseline convergence tolerance specified, internal convergence often occurred in zero or one iteration, usually zero for gradient calculation (see earlier comments on poor gradients for the FPI method). Thus, for the majority of function calls, the FPI method would take approximately the same amount of time as the POBD method. The time per function call for the FOBD method was smaller because the POST subproblems are executed in a parallel manner. The ascent POST subproblem required a longer amount of execution time than the flyback subproblem, and its average execution time is the limiting factor in Table 6. Even though the two POST subproblems for the CO method were executed in parallel, the CPU time per function call was significantly longer than the PODB and FOBD methods. This was because local optimization of the POST subproblems to minimize the J error functions required more computational effort relative to a simple integration of the equations of motion.

Note that the results for the POBD method were the same regardless of which way the compatibility constraint was posed [either Eq. (1) or the combination of Eqs. (2) and (3)]. This observation was not the case, however, for the FOBD method. When the compatibility constraints were formed as one quadratic inequality per design variable, a feasible solution could not be found. As already mentioned, the FOBD method (with separated compatibility constraints) had to be restarted to arrive at its solution, whereas the PODB methods made good progress toward the minimum. The POBD methods were, therefore, the most robust and were generally preferred for this class of branching trajectory problem characterized by nonhierarchical coupling between the branches, but only a limited number of feedback variables (in this case only one, the flyback fuel required).

Additional quantitative results can be seen in Table 7, which shows the difference in selected values in the final staging vector for the methods. Included for comparison is the manual iteration method. Because the results for the two POBD methods were almost exactly the same, just one is included in Table 7. When it is assumed that internal convergence tolerances negatively affected the FPI method staging vector, the results from the other MDO methods are taken to be more credible. The results imply that a smaller flight-path angle (gamma) at staging can reduce flyback propellant weight consumed and, thus, increase the payload. (In the FOBD case, the lower staging altitude also has a similar effect.) The

Table 7 Staging data results for K-1

Method	Altitude, ft (m)	Velocity, ft/s (m/s)	Gamma, deg
Manual iteration	138,028 (42,070.9)	4,172.28 (1271.71)	33.337
FPI	138,019 (42,068.2)	4,169.27 (1270.79)	33.520
POBD 2	138,014 (42,066.7)	4,166.92 (1270.07)	33.272
FOBD	137,542 (41,922.8)	4,160.34 (1268.07)	32.892
CO	138,028 (42,070.9)	4,171.15 (1271.37)	32.667

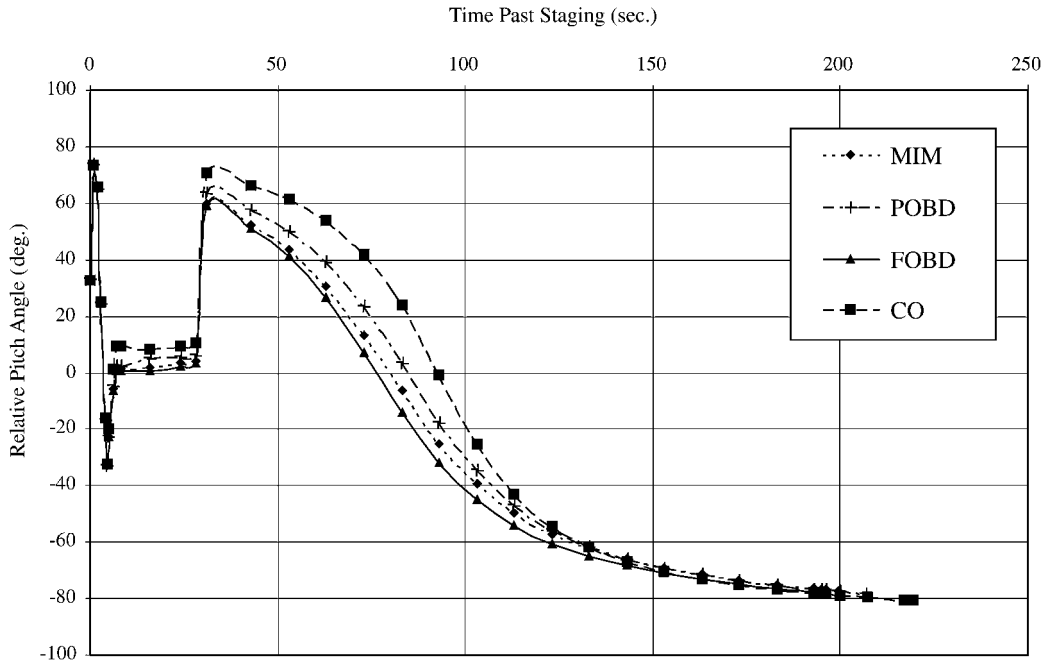

Fig. 9 Pitch angle vs time for the K-1 flyback trajectory.

returning booster desires this lower angle so that it can perform its pitchover maneuver more efficiently. In contrast, the flight-path angle at staging in the manual iteration method is under the exclusive control of the ascent branch. The manual iteration method increases the flight-path angle slightly relative to the FOBD and CO methods to improve the orbital stage performance unaware of the flyback and overall system benefits to be gained through a compromise.

Through the ascent, the trajectories are very similar with regard to altitude and velocity. In fact, as can be seen in Table 7, the staging points chosen by each method are relatively close to one another. However, the flyback trajectory shows more pronounced differences. Differing pitch angles during the flyback pitcharound maneuver are highlighted in Fig. 9. For example, the CO method selects a higher pitch angle history relative to the other methods between 30 and 150 s. Although Fig. 9 does highlight the differences in the solutions reached by each method, the effect of pitch angle schedule during the portions of the flyback just after flyback engine cutoff on orbital payload is small due to the extremely low aerodynamic forces at this altitude.

Conclusions

This paper has provided an introduction to the nonhierarchically coupled branching trajectory problem, using the Kistler K-1 branching trajectory as an example. One ground rule of the study was to develop a solution approach that would retain the existing single-segment trajectory optimization codes widely used in conceptual design. POST was used to represent this type of code. The deficiencies of the traditional methods of the one-and-done and manual iteration methods were discussed. The main deficiency was that conflicting local objective functions existed in these methods. Each POST subproblem had its own objective to fullfill, but compromises in each individual trajectory segment that could benefit the entire trajectory went unexploited. MDO methods introduced a system-level optimizer that resulted in an overall, system-level objective being met for the branching trajectory problem. The use of these methods for the Kistler K-1 problem showed that an increase in payload weight of 1.5%, on average, could be obtained. When one considers the costs of launching payloads is about $3000–$5000/lb ($6600–$11,000/kg), small payload gains can be significant. More payload can be put into orbit per launch, and thus, more profits can be achieved.

The MDO techniques used to solve the branching trajectory problem were FPI (an all-at-once approach), OBD to eliminate iteration between the branches (two different formulations), and CO to enable parallel subproblem execution with distributed, coordinated optimizers. The results of solving the K-1 branching trajectory with MDO methods indicated a preference for POBD when simultaneously considering its superior robustness, ease of setup, straightforward implementation, fast execution time, and optimality of the results. A specific conclusion from the K-1 trajectory also showed that a small decrease in the flight-path angle at staging aided in reducing the booster's flyback propellant. When it was assumed that more ascent propellant would be available as a result, this trajectory compromise allowed more payload weight to be flown to orbit.

Acknowledgments

This work was partially sponsored by NASA via the Georgia Space Grant Consortium and Marshall Space Flight Center under NAG8-1597 to the Georgia Institute of Technology. The authors would like to thank the members of the Space Systems Design Laboratory at the Georgia Institute of Technology for their support. Thanks are extended to Richard Kohrs of Kistler Aerospace for his assistance with the K-1 data.

References

[1]Vlases, W. G., Paris, S. W., Lajoie, R. M., Martens, P. J., and Hargraves, C. R., "Optimal Trajectories by Implicit Simulation, Ver. 3.0, User's Manual," WRDC-TR-90-3506, Wright–Patterson AFB, OH, 1990.

[2]Hallman, W. P., "Mission Timeline for a Shuttle-IUS Flight Using a Nonstandard Shuttle Park Orbit," The Aerospace Corporation, Rept. TOR-0083 (3493-14)-1, El Segundo, CA, Oct. 1982.

[3]Brauer, G. L., Cornick, D. E., and Stevenson, R., "Capabilities and Applications of the Program to Optimize Simulated Trajectories," NASA CR-2770, Feb. 1977.

[4]Naftel, J. C., and Powell, R. W., "Flight Analysis for a Two-Stage Launch Vehicle with a Glideback Booster," *Journal of Guidance, Control, and Dynamics*, Vol. 8, No. 3, 1985, pp. 340–343.

[5]Lepsch, R. A., and Naftel, J. C., "Winged Booster Performance with Combined Rocket and Airbreathing Propulsion," *Journal of Spacecraft and Rockets*, Vol. 30, No. 6, 1993, pp. 641–646.

[6]Anderson, R. L., Aguirre, J. T., Striepe, S. A., Brauer, G. L., and Engel, M. C., "Program to Optimize Simulated Trajectories (POST II), Vol. I. Guide for New Users," Ver. 1.0.5.G, NASA Langley Research Center, Hampton, VA, April 1999.

[7]Ledsinger, L. A., "Solutions to Decomposed Branching Trajectories with Powered Flyback Using Multidisciplinary Design Optimization," Ph.D. Dissertation, Space Systems Design Lab., School of Aerospace Engineering, Georgia Inst. of Technology, Atlanta, June 2000.

[8]Kohrs, R., and Petersen, R., "Development of the K-1 Two-Stage, Fully-Reusable Launch Vehicle," AIAA Paper 98-1540, April 1998.

[9]Associate Administrator for Commercial Space Transportation (AST), "2000 Reusable Launch Vehicle Programs & Concepts," Federal Aviation Administration, Washington, DC, Jan. 2000.

[10]Anisimov, V. S., Lacefield T. C., and Andrews J., "Evolution of the NK-33 and NK-43 LOX/Kerosene Engines," AIAA Paper 97-2680, July 1997.

[11]Mecham, M., "Aerojet Acquires Major K-1 Engine Shipment," *Aviation Week and Space Technology*, Vol. 147, No. 10, 1997, pp. 61, 62.

[12]Kroo, I., Altus, S., Braun, R., Gage, P., and Sobieski I., "Multidisciplinary Optimization Methods for Aircraft Preliminary Design," AIAA Paper 94-4325, Sept. 1994.

[13]Braun, R. D., Powell, R. W., Lepsch, R. A., Stanley, D. O., and Kroo, I. M., "Multidisciplinary Optimization Strategies for Launch Vehicle Design," AIAA Paper 94-4341, Sept. 1994.

[14]Braun, R. D., Kroo, I. M., and Gage, P. J., "Post-Optimality Analysis in Aerospace Vehicle Design," AIAA Paper 93-3932, Aug. 1993.

[15]Cormier, T., Scott, A., Ledsinger, L., McCormick, D., Way, D., and Olds, J., "Comparison of Collaborative Optimization to Conventional Design Techniques for a Conceptual RLV," AIAA Paper 2000-4885, Sept. 2000.

[16]Alexandrov, N. M., and Lewis, R. M., "Analytical and Computational Properties of Distributed Approached to MDO," AIAA Paper 2000-4718, Sept. 2000.

D. B. Spencer
Associate Editor