Trans. JSASS Aerospace Tech. Japan
Vol. 14, No. ists31, pp.Pd_1 - Pd_8, 2017

Automated Sensitivity Analysis of Interplanetary Trajectories for Optimal Mission Design

By Jeremy KNITTEL,[1] Kyle HUGHES,[1], Jacob ENGLANDER,[1] and Bruno SARLI[2]

[1)] *NASA Goddard Space Flight Center, USA*
[2)] *Catholic University of America, USA*

(Received April 17th, 2017)

This work describes a suite of Python tools known as the Python EMTG Automated Trade Study Application (PEATSA). PEATSA was written to automate the operation of trajectory optimization software, simplify the process of performing sensitivity analysis, and was ultimately found to out-perform a human trajectory designer in unexpected ways. These benefits will be discussed and demonstrated on sample mission designs.

Key Words: mission design, trajectory optimization, missed-thrust analysis

1. Introduction

The first task of the flight dynamicist on an interplanetary mission is typically to create an optimized (or at least feasible) trajectory. This is also the primary focus of presented work at flight dynamics conferences and symposia. However, the second task for the flight dynamicist often occupies far more time: sensitivity analysis. We use this phrase to describe any change to the nominal and optimized trajectory due to considerations of operational constraints, subsystem requirements, or spacecraft faults. This paper presents a new method of automating sensitivity analysis and the resulting impact on global trajectory optimization and improved overall mission design.

The trajectory solver used in this work is the Evolutionary Mission Trajectory Generator (EMTG).[1)] It is capable of solving for electric propulsion (EP) trajectories using a slightly modified version of the Sims-Flanagan transcription[2)] and trajectories using impulsive maneuvering via a two point shooting method.[3)] EMTG generates trajectories using a user-supplied or random initial guess, a non-linear programming (NLP) gradient-based optimizer and monotonic basin hopping (MBH).

If given a good user-supplied initial guess, EMTG converges extremely quickly. If a poor or no initial guess is provided, EMTG is still fully capable of converging to the same optimal trajectory, however the process is stochastic and requires sufficient time for MBH to provide the NLP solver with a good enough starting point, in the desired region of the design space. Unfortunately, the trajectory designer does not know *a priori* what the appropriate length of time is. Frequent checking of the solution requires frequent effort from the trajectory designer, and is a very human-labor intensive operation. Further, manually modifying any cases that did not satisfactorily converge is entirely impractical when running the hundreds or thousands of cases often required to perform sensitivity analysis.

Initially written to address only the issue of engineer work-hours, a suite of tools was written and named the Python EMTG Automated Trade Study Application (PEATSA). It was quickly found to be an effective means of tackling additional unforeseen challenges as well. The fundamental principle behind PEATSA is that it iteratively runs a "batch" of sensitivity analysis cases in the trajectory optimization software until some stop criteria has been satisfied for all cases. Between each iteration, PEATSA will use converged cases to give better and better initial guesses to any case that has not yet met the stopping criteria. Introducing iterative running of EMTG has thus far produced four major benefits for sensitivity analysis:

1. It allows shorter run time of each case, thus decreasing wasted computation time.
2. It allows updating of initial guesses, thus increasing the likelihood that the trajectory solver converges.
3. It allows random modification of design options (such as gravity assist sequence) that might be more optimal than those used for the baseline design. This makes this sensitivity analysis method much closer to globally optimal than otherwise would be possible.

The reader should note that this is not the first attempt to introduce a flyby sequence and spacecraft systems trade study optimization capability into EMTG. Vavrina et al.[9)] implemented a multi-objective hybrid optimal control (HOC) trade study optimizer using a cap-and-optimize transcription and a multi-objective genetic algorithm. This technique was reasonably effective in exploring a large trade space of target selection and systems designs but suffered from a significant flaw in that it required the optimization of each individual trajectory sub-problem to be perfectly reliable. As mentioned, the stochastic nature of the trajectory solver means that it can never be perfectly reliable. The approach presented in this work attempts to correct this deficiency by instantiating many trajectory optimization sub-problems that can share information by using each-others solution as a new initial guess.

Results will be shown describing the operation of PEATSA, and demonstrating the above benefits when applied to the following sensitivity analysis tasks: performing a mission systems trade study, developing a launch window, and analyzing missed-thrust during an EP trajectory.

2. Trajectory Solver

While PEATSA was written and currently only works with EMTG, it could be adapted to just about any trajectory optimization software. EMTG is a particularly convenient choice, however, as it has a Python interface and can be run from a command line. EMTG is written in object-oriented C++, with a class named *missionoptions* to hold all of the options specified in an EMTG input file, including the initial guess. Similarly, EMTG has C++ classes named *mission* and *journey* which hold all of the trajectory data encapsulated in an EMTG output file. All of these data structures were replicated in Python for use with EMTG's graphical user interface (GUI). These GUI classes allow Python scripted reading and writing of EMTG input and output files. If PEATSA were to be adapted to a new trajectory solver, Python classes to hold data for input and output files would be needed, or PEATSA itself would need to change languages to match the software. Further, it must be possible to initiate an optimization run in the new trajectory solver from a command line so that PEATSA can run instances of the software using Python code.

The EMTG Python classes will be referenced in pseudocode below. The formulas provided will not be usable, as they do not use exact EMTG or PEATSA variable names, but are instead intended to give understanding of the operation of PEATSA.

3. Methodology

There are four main tasks performed by PEATSA. These tasks and the general structure in which they are completed is shown in Algorithm 1. This section will discuss the overall methodology of PEATSA and each of these tasks in greater detail.

Algorithm 1 Pseudocode of PEATSA

```
parse PEATSA options file
current_iteration = 0
Task 1. Create an options file for each trajectory case in the correct format for EMTG. These are the iteration 0 cases
while there are cases to run do
    Task 2. Run [current_iteration] cases in EMTG and wait until all cases have completed
    Task 3. Post-process [current_iteration] results, saving results to a comma separated value (csv) file
    current_iteration++
    Task 4:
    for all cases in the csv file do
        if case has not met success criteria then
            find another case that would be a good initial guess for this case and re-create the options file for this case using the improved initial guess as an iteration [current_iteration] case
        end if
    end for
end while
```

Table 1. Highlighted PEATSA options

n	number of cpu core's available
$EMTG\_location$	location of trajectory optimization solver
$start\_type$	whether to start from scratch, or from a previously PEATSA run
run_type	type of sensitivity analysis task being conducted
$nominal\_trajectory$	the trajectory being modified
$EMTG\_runtime$	how long to run each case through EMTG
$goal$	the objective that each case is trying to acheive (for example, final mass > 1000 kg)
$guess\_folder$	a list of folders that PEATSA can use to look for initial guesses in addition to the PEATSA results themselves
$wait\_for\_guess$	should PEATSA wait to run a given case until it has an initial guess?
$modify\_flybys$	should PEATSA modify the nominal flyby sequence when re-running cases?
$maximum\_flybys$	maximum number of flybys
$flyby\_bodies$	which bodies are allowable for a flyby
$options\_override$	a list of options to change from the $nominal\_trajectory$
$options\_conditions$	a list of conditional statements determining whether to actually make the changes specified in the matching list entry from $options\_override$. This allows the options to be modified for only a subset of cases
$analysis\_formulas$	a Python list of items to be calculated during post-processing for each trajectory
$sorting\_criteria$	how to sort the trajectories in the post-processed csv file
$comparison\_criteria$	how to compare versions of the same trajectory from different iterations
$plots\_to\_make$	a list of plots to generate after each iteration

3.1. PEATSA Options

Before addressing the main tasks, it is useful to mention that the first operation executed is running a separate Python script to set options which will control the procedure of the given PEATSA run. PEATSA has more options than can be discussed here, but some of the most important are highlighted in Table 1. These options along with additional more specific ones will be discussed further in Sections **3.2.** - **3.5.**.

3.2. Task 1. Case Creation

The first main utility of PEATSA is to generate EMTG options scripts for sensitivity analysis cases. Even starting from a nominal trajectory, it can be a labor intensive task to create a similar, but slightly modified case for every new situation de-

Table 2. Highlighted PEATSA general trade study options

options_to_vary	a list of which options are to be varied
option_ranges	a list of lists to specify the values to consider for each option in *options_to_vary*
trade_study_type	flag controlling if PEATSA generates all unique combinations of options or only varies one option at a time

sired. For example, if a nominal trajectory launches on January 15th, then to develop a ± 15 day launch window, the same case must be run with the launch date constrained to January 1st, 2nd, 3rd, ..., 29th, and 30th. While this is not a particularly challenging task, PEATSA completes it faster, and with much less risk of error. Further, certain sensitivity analysis studies require thousands of cases, an impractical amount of work to complete manually.

3.2.1. General Trade Studies

The most common use of PEATSA is to perform trade studies, a subset of sensitivity analysis. In this sense, we are referring to varying any number of constraints and/or spacecraft system properties from their value in a nominal trajectory. With this definition, we include varying any of the following common trajectory details: the launch date of a trajectory (launch window), the launch vehicle, the solar array sizing, the spacecraft dry mass, the total flight time, the Sun-Earth-target angle at target arrival, etc. The only limit to the property which is varied is that it must be an existent constraint or setting in EMTG, and therefore EMTG's Python options class MissionOptions.py.

The additional options that PEATSA uses to create a trade study run are shown in Table 2. Note that general trade studies can be performed in two distinct ways, based on the *trade_study_type*. Because a given PEATSA run might involve a long list of options to vary, it is not clear how PEATSA will create the cases. That is, if *options_to_vary* has two properties, should PEATSA vary both of these items at once, or only one at a time? For example, assume *options_to_vary* = [MissionOptions.launch_date,MissionOptions.launch_vehicle] and *option_ranges* = [[Jan 1st, Jan 3rd],[Atlas V 401, Atlas V 551]] and the nominal trajectory launches on January 2nd on an Atlas V 431. Then, if both options are concurrently varied, the 4 cases created will be [Jan 1st, Atlas V 401], [Jan 1st, Atlas V 551], [Jan 3rd, Atlas V 401] and [Jan 3rd, Atlas V 551]. On the other hand, if only one option is varied at a time, then the 4 cases created will be [Jan 1st, Atlas V 431], [Jan 3rd, Atlas V 431], [Jan 2nd, Atlas V 401], and [Jan 2nd, Atlas V 551]. In this simple example, 4 cases are created in both circumstances. However, in general, concurrently varying options will result in far more cases. Concurrently varying cases will result in a number of cases equal to the multiplicative of the length of each list in *option_ranges*, whereas single property variation creates a number of cases equal to the summation of the length of each list in *option_ranges*.

3.2.2. Missed-thrust

A trajectory employing solar electric propulsion (EP) often requires firing thruster(s) continuously for months. However, if the spacecraft has a fault of any sort *en route*, it may need to go into safe-mode and temporarily turn off its thrusters. For this reason, EP trajectories are typically designed with additional margins, not imposed on chemical propulsion missions.[4] As a result, one crucial analysis task for EP missions is missed-thrust sensitivity. The general principle is to ensure that the spacecraft can still reach its destination and complete mission objectives if a fault occurs requiring an unexpected cessation of thruster firing.

Table 3. Highlighted PEATSA missed-thrust options

x	how often to insert a forced coast
objective_type	should PEATSA insert fixed length forced coasts or optimize on coast length

Missed-thrust analysis is performed by inserting forced-coasts into a nominal trajectory, in order to simulate a safe-mode event. Then, a new trajectory optimization is run following the forced-coast to ensure that a new feasible trajectory is still possible, despite having missed the initially planned thrusting opportunity. PEATSA does this in one of two ways:

1. Using a fixed forced-coast length, constraining delivered mass and running EMTG by optimizing on flight time.
2. Constraining delivered mass and flight time, and maximizing the forced-coast length as proposed by Sarli.[5]

The options specific to a missed-thrust PEATSA run are shown in Table 3. PEATSA iterates through the dates of the *nominal_trajectory* and creates a new options case file every *x* days following the spacecraft's launch. In each case file, the options will be modified such that the new initial boundary condition is a free point in space, corresponding to the spacecraft state along its planned optimal flight path. Then, PEATSA will either set the options up so that EMTG forces a fixed coast, or attempts to optimize how long the spacecraft coasts before resuming thrusting.

3.3. Task 2. Run All Cases

In each iteration, PEATSA will grab all cases that have been generated, and run them through the trajectory optimization solver. As shown in Table 1, one of the parameters set in the PEATSA options script is the number of cpu's available, *n*. This allows PEATSA to create a list of all active processes, and only run *n* instances of EMTG at once. Then, as each case is finished, PEATSA can start the next case in the queue.

3.4. Task 3. Post-process Results

After every iteration, PEATSA performs a simple parsing of all the finished cases that have been run through EMTG for all iterations. The output is an easy to digest csv file summarizing all of the cases in the PEATSA run. Each line in the csv file corresponds to one of the cases, and the data that populates the fields on that line is pulled from the most successful trajectory solution found for that case thus far. For all PEATSA run types, the csv file will have fixed entries: location of the trajectory file, the optimization objective used to generate that trajectory, the launch date, the final mass of the spacecraft, and the total flight time.

Beyond the default parameters, the user can specify additional data to populate the csv file using the PEATSA option, *analysis_formulas*. This list contains strings which are evaluated in Python using the "eval" function. For example, if *analysis_formulas* = ["Mission.launch_C3","MissionOptions.launch_vehicle"], then the csv file will also grab the launch energy and launch vehicle

type used for each trajectory. Note that these formulas must use EMTG's Python interface classes, explained in Section **2.**.

For every operation of the post-processing code, except for iteration 0, there will be multiple versions of each case that have been run through EMTG. Therefore, part of the responsibility of Task 3 is to compare the iteration 0 version of case xyz with the iteration *i* version of case xyz. The user must specify in the PEATSA options a criteria with which to compare, *comparison_criteria*. By performing the comparison, only one version of each case will end up in the csv file, hence one line per case.

Because PEATSA re-parses every file in its results folder between iterations, it is possible to manually insert cases into the cue as well. For example, assume that two identical PEATSA runs are taking place on servers A and B. If server B finds a particularly good version of case xyz that server A has not found yet, it is possible to manually insert it into the results folder on server A. Then, at the next iteration, server A's PEATSA run will find the improved result and include it in its own results and will use it as an initial guess for its other cases.

The final operation of Task 3 is to make any user requested plots. For example, if PEATSA is performing a launch window analysis, and optimizing each launch day's trajectory on final mass, then after every iteration, PEATSA can plot the highest mass trajectory as a function of time. PEATSA saves these plots as images. Then, after the program is completed, PEATSA generates a movie of all image files allowing an easy to see iteration history.

By generating these csv files and plots after every iteration, it is possible to monitor the progress of PEATSA as it goes. As each iteration progresses, the user can open the csv files and the images and ensure that the program is working correctly, or if modifications are needed.

3.5. Task 4. Re-create Cases with Improved Initial Guesses

The final operation is also the most novel and what makes PEATSA so remarkably powerful. EMTG can converge to a globally optimal solution without any initial guess whatsoever. However, as trajectory problems become more and more complicated with constraints, and the number of NLP decision variables increases dramatically, the likelihood of finding a globally optimal solution without an initial guess decreases greatly.

When performing sensitivity analysis, a natural method is to use the nominal trajectory as an initial guess for the modified cases. In many instances this is perfectly suitable, and an effective method. However, in many situations the nominal trajectory is not particularly useful. For example, if performing a missed-thrust analysis, the nominal trajectory often looks very different than the recovery trajectory after an unintended two week coast during a critical thrusting period. Or, if performing a launch window analysis over an entire year, the January 1st case is not a particularly useful initial guess for the December 1st case. However, the January 1st case is likely a good initial guess for the February 1st case which is a good initial guess for the March 1st case, and so on.

As will be shown later, for particularly difficult problems that require a good initial guess, PEATSA is capable of propagating initial guesses from one side of the sensitivity analysis cases to the other, where there initially were none. Again, using the example of the launch window analysis, the usable initial guesses propagated from the nominal launch of January 1st to February 1st to March 1st all the way through the calendar until the December 1st case had an effective initial guess.

This propagation of initial guesses even occurs when the nominal trajectory is not a useful initial guess, as in the case of missed-thrust. However, if 100 missed-thrust recovery cases are run each iteration, that increases the likelihood that a good trajectory is found by 100, as compared to a single case using the stochastic MBH. As soon as one case finds a good solution, the very next iteration, that trajectory can begin propagating out to all other cases.

Further, because of how MBH works, there is no difference in running an EMTG case once for 1 hour, or twice for thirty minutes or six times for ten minutes, as long as the best result from the previous execution is supplied as an initial guess for the next execution. However, because PEATSA can grab initial guesses from other "neighboring" cases, it is quite effective to decrease each EMTG run's execution time. The net computation time is therefore always decreased using PEATSA.

4. Case Studies

To demonstrate how PEATSA works, the design and sensitivity analysis of two sample missions is presented in this section.

4.1. Uranus

The first case study is a search for launch opportunities to Uranus. The constraints and mission parameters are summarized in Table 4. This study was not meant to be exhaustive and the best possible means of reaching Uranus were likely not found. Rather, this is meant to demonstrate the capability of PEATSA and how little hands-on engineering labor is required to complete tasks that were once very hands-on. It required roughly twenty minutes of set-up and then the entire run required roughly 36 hours of computation time on a 64 core server.

Reaching Uranus directly without any gravity assists *en route* is almost certainly infeasible. However, a mission designer likely would not know *a priori* what a usable set of flyby bodies is. PEATSA is able to solve this problem by randomly attempting different flyby combinations. In this case, PEATSA was not supplied with any specific flyby combinations, except that it was allowed up to 5 flybys of Venus, Earth, Mars, Jupiter, and Saturn in any combination. Then, during each iteration PEATSA would randomly insert additional cases with a different potential flyby sequence. Likely, in most situations, these cases fail to find an acceptable or improved result. However, as soon as one good combination is found, it percolates out to nearby cases. This ends up resulting in different families of solutions throughout the launch window.

The initial set of results without any gravity assists are presented in Fig. 1. It is already clear that there are families of solutions found, despite EMTG not having any initial guesses yet. In the second iteration results, shown in Fig. 2, two new flyby combinations were found to be better than the nominal case of a direct transfer to Uranus, as shown on the figure. Again, many

Table 4. Specifications for Uranus mission

Mission Parameters	
Propulsion model	impulsive
Maximum flight time	12 years
Maximum numbers of DSMs	1 per flyby
Launch Vehicle	Atlas V 551
Spacecraft Isp	220 seconds
Intercept velocity	< 7 km/s
EMTG objective	maximum mass
EMTG run-time per iteration	60 seconds
PEATSA Options	
run_type	launch window
sorting_criteria	launch date
comparison_criteria	maximum final mass
wait_for_guess	yes
modify_flybys	yes
maximum_flybys	5
flyby_bodies	Venus, Earth, Mars, Jupiter, Saturn
options_to_vary	launch date
option_ranges	July 2024 through June 2025

different randomly varied flyby sequences were attempted by EMTG, but only these two represented improvements. Now part of the solution set, these improved gravity assist trajectories can improve the solution of "nearby" cases. For example, the Venus-Earth flyby trajectory shown in Fig. 2 has a launch date of 3/29/2025. Using that as an initial guess, a very similar solution was found for a launch date of 3/30/2025 in the following iteration. By iteration 10, these solutions were able to propagate quite far over the design space as shown in Fig. 3. Because of the stochastic nature of both EMTG's solution algorithm and PEATSA's flyby randomizations, there is not a deterministic way to determine when the study is complete and globally optimal solutions have been found everywhere. Alternatively, at the cost of additional run-time per iteration, more than 1 new flyby combination could have been attempted per iteration. However, in this simple example, that was not done, and because no improvement was found from iteration 70 to 80, the run was terminated. The final results are shown in Fig. 4. In the final data set, there were 3 different gravity assist combinations present: Earth only, Venus-Earth, and Venus-Earth-Earth.

4.2. Low-Thrust Asteroid Sample Return

The second case study is similar to a low-thrust version of the Osiris REx mission which launched in 2016. Rather than redesign the mission to Bennu, a simple search of the JPL small body database[6] was done, in order to select an interesting target with eccentricity greater than .2 and inclination greater than 10 degrees. This was meant to create a challenging scenario that would require low-thrust in order to be feasible. 1949 TG, also known as Daphne, was chosen as the target.

Two PEATSA runs were performed in order to design this mission. First, a trade study was conducted in order to select mission parameters. This could have been done on any spacraft or mission system, but for this study, we selected launch vehicle selection, solar array sizing and propellant tank sizing. The full details of the first PEATSA run are presented in table 5.

PEATSA was an extremely efficient means of conducting this

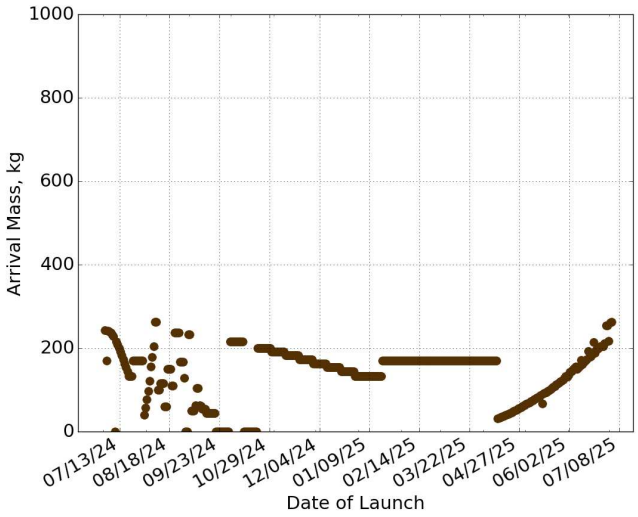

Fig. 1. First iteration of results for the Uranus launch window

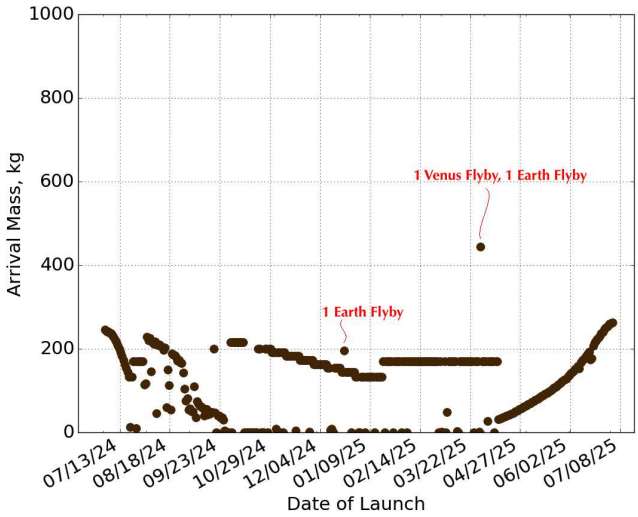

Fig. 2. Iteration 2 results for the Uranus launch window

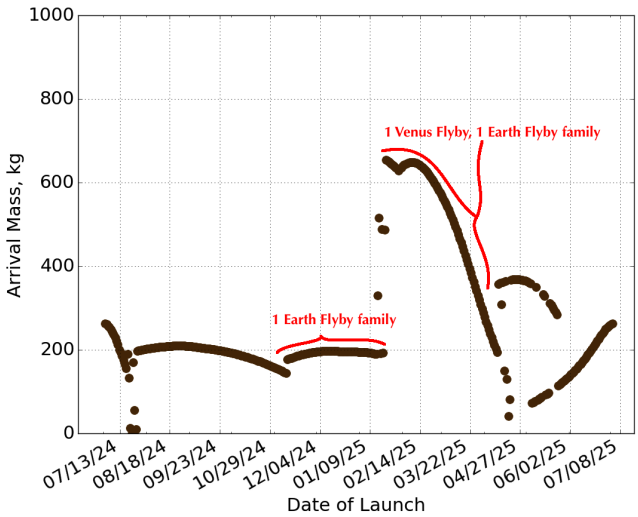

Fig. 3. Iteration 10 results for the Uranus launch window

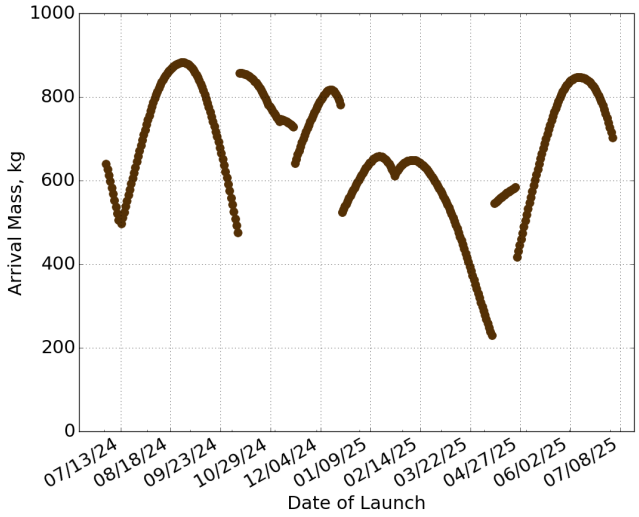

Fig. 4. Iteration 80 results for the Uranus launch window

Table 5. Specifications of system study for mission to Daphne

Mission Parameters	
Propulsion model	polynomial thrust, mass flow rate vs. power available
Propulsion system	2 NEXT engines[7)]
Maximum flight time	10 years
Earth return velocity	< 10 km/s
Duty cycle	90%
Propellant margin	10%
Power margin	15%
Bus power	1 kW
Stay time	> 500 days
EMTG run-time per iteration	20 seconds
Low-thrust transcription	Finite Burn[8)]
PEATSA Options	
run_type	trade study
comparison_criteria	maximum final mass
wait_for_guess	yes
flyby_bodies	none
options_to_vary	launch vehicle; solar array size; electric propellant load
option_ranges	Atlas V - 401 (0), 411 (1), 421 (2), 431 (3), 541 (9) or 551(10); 20 to 40 kW; 900 to 1500 kg
trade_study_type	vary each option separately

study. Only 19 total iterations were required before all cases had converged to a trendline that seemed likely to be globally optimal. Given the short run time in EMTG that was used, the full trade study was complete in roughly thirty minutes. Similar iteration histories could be shown as those in the previous section, however in the interest of space, they will not be reproduced and only the final results will be shown. The final trade study results are presented in Figs. 5 - 7.

A trajectory was arbitarily picked from the initial trade study to act as a baseline mission design. This trajectory is shown in Fig. 8. Then, PEATSA was used to perform missed-thrust

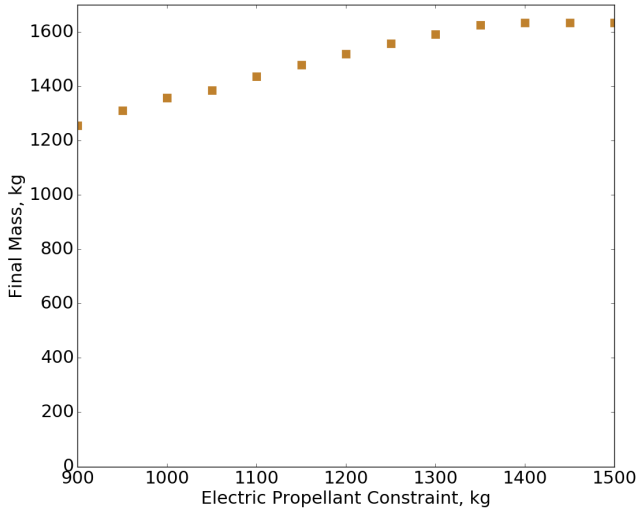

Fig. 5. Final trade study results for the Daphne mission.

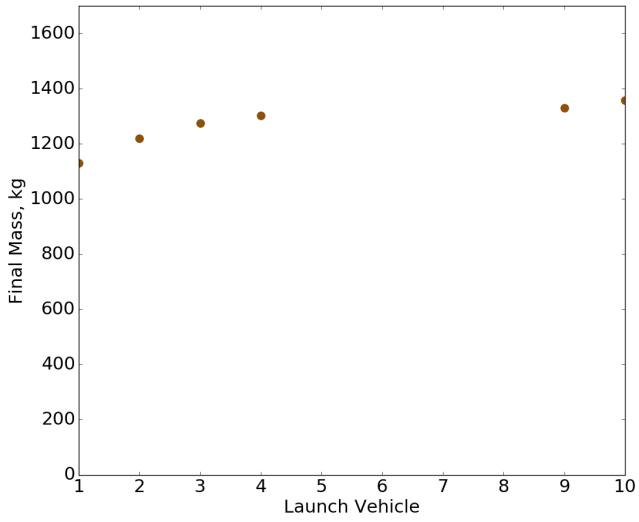

Fig. 6. Final trade study results for the Daphne mission. See table 5 for launch vehicle codes

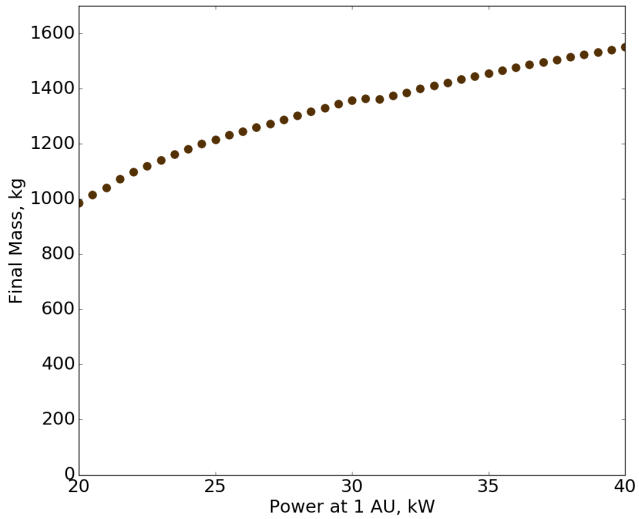

Fig. 7. Final trade study results for the Daphne mission.

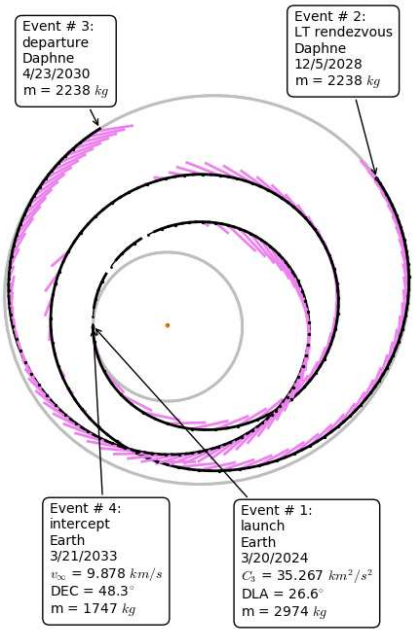

Fig. 8. Nominal Trajectory for the Daphne mission.

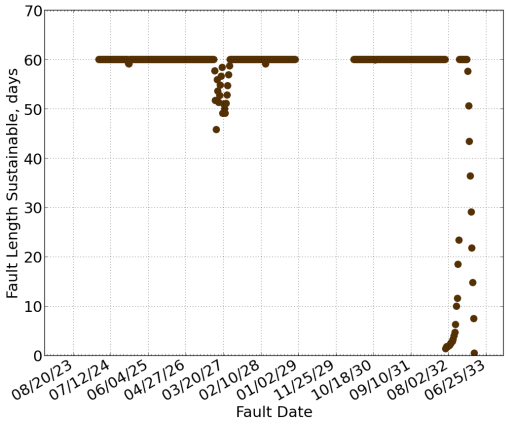

Fig. 9. missed-thrust results for the Daphne mission.

analysis on the baseline design. Recall that EP missions require higher margins in order recover from a safe-mode event preventing thrusting. As yet, there is no means of optimizing a trajectory for missed-thrust robustness, so designers must run cases to ensure that mission objectives and constraints can still be met despite faults. The specifications for the missed-thrust study are shown in Table 6. Requiring a boundary condition to be along a fixed nominal trajectory is a mathematically more difficult problem, so the run-time in EMTG was increased considerably for this study. The net effect is that the final results required almost a week to fully generate and over 60 iterations.

The final output of a missed-thrust study is typically the fault length sustainable vs. the fault date. This is presented for the Daphne mission in Fig. 9. Note there are critical periods where safe modes are more impactful than other periods of the trajectory. If a missed-thrust event occurs during one of these times, then the spacecraft cannot coast as long and still complete mission objectives. These critical events tend to correspond to important orbital extremes, such as aphelion, perihelion, and node crossings.

Table 6. Specifications of missed-thrust study for mission to Daphne

Mission Parameters	
Propulsion model	polynomial thrust, mass flow rate vs. power available
Propulsion system	2 NEXT engines[7]
Maximum flight time	10 years
Launch vehicle	Atlas V 551
Propellant load	1350 kg
Solar array size	30 kW (@ 1AU)
Earth return velocity	< 10 km/s
Duty cycle	90%
Propellant margin	10%
Power margin	15%
Bus power	1 kW
Stay time	> 500 days
EMTG run-time per iteration	1800 seconds
EMTG objective	maximum coast
Low-thrust transcription	Finite Burn[8]
PEATSA Options	
run_type	missed-thrust
comparison_criteria	maximum length initial coast
wait_for_guess	yes
flyby_bodies	none
options_to_vary	none
x	7 days
goal	60 day coast
objective_type	maximum coast

5. Conclusion

Performing sensitivity analysis for interplanetary missions is significantly easier to setup when using PEATSA because the cases can be auto-generated using only a simple Python options script and PEATSA's Task 1. They require literally zero human-effort to run through a trajectory optimization routine because of Task 2. They are easier to analyze because of the plots and data summary files made in Task 3. The wait time to the final results is less because Task 4 reduces total required computation time, typically by a very large margin. The ability to introduce random perturbations such as changing the flyby sequence can introduce trajectory modifications that often require a genetic algorithm to consider. And finally, the ability for cases to communicate with each other through initial guesses increases the computation efficiency of every optimization run.

References

1) J.A Englander and B. A. Conway : *An Automated Solution of the Low-Thrust Interplanetary Trajectory Problem* Journal of Guidance, Control, and Dynamics, Vol. 40, No. 1 (2017), pp. 15-27.
2) J. A. Sims and S. N. Flanagan : *Preliminary Design of Low-Thrust Interplanetary Missions* AAS/AIAA Astrodynamics Specialist Conference, Girdwood, Alaska, Paper AAS 99-338, August 1999.
3) M. Vavrina, J. Englander and D. Ellison : *Global Optimization of N-Maneuver, High-Thrust Trajectories Using Direct Multiple Shooting* 26th AAS/AIAA Space Flight Mechanics Meeting, Napa, CA, February 2016.
4) D. Oh, J.S. Snyder, R. Hofer, D. Laundau, and T. Randolph : *Solar Electric Propulsion for Discovery Class Missions* International Elec-

tric Propulsion Conference, Washington, DC, 2013.
5) B. Sarli, M. Ozimek, J. Atchison, J. Englander, and B. Barbee : *NASA Double Asteroid Redirection Test (DART) Trajectory Validation And Robustness* 27th AAS/AIAA Space Flight Mechanics Meeting, San Antonio, TX, 2017.
6) *JPL Small-Body Database Search Engine* `https://ssd.jpl.nasa.gov/sbdb_query.cgi`, Accessed March 2017.
7) NASAs Evolutionary Xenon Thruster (NEXT): Ion Propulsion GFE Component Information Summary for Discovery Missions, July 2014, `https://go.usa.gov/xXGwf`, July 2014, Accessed March 2017.
8) Englander, Jacob A., Jeremy M. Knittel, Ken Williams, Dale Stanbridge, and Donald H. Ellison. : *Validation of a Low-Thrust Mission Design Tool Using Operational Navigation Software* 27th AAS/AIAA Space Flight Mechanics Meeting, San Antonio, TX, 2017.
9) Vavrina, M.A. and Englander, J.A. and Ghosh, A.M. : *Coupled Low-Thrust Trajectory and Systems Optimization Via Multi-objective Hybrid Optimal Control* 25th AAS/AIAA Space Flight Mechanics Meeting, Williamsburg, VA, 2015.