CEAS Space Journal (2018) 10:79–86
https://doi.org/10.1007/s12567-017-0171-7

ORIGINAL PAPER

MONTE: the next generation of mission design and navigation software

Scott Evans[1] · **William Taber**[1] · **Theodore Drain**[1] · **Jonathon Smith**[1] · **Hsi-Cheng Wu**[1] · **Michelle Guevara**[1] · **Richard Sunseri**[1] · **James Evans**[1]

Received: 1 May 2017 / Accepted: 13 September 2017 / Published online: 6 January 2018

Abstract
The Mission analysis, Operations and Navigation Toolkit Environment (MONTE) (Sunseri et al. in NASA Tech Briefs 36(9), 2012) is an astrodynamic toolkit produced by the Mission Design and Navigation Software Group at the Jet Propulsion Laboratory. It provides a single integrated environment for all phases of deep space and Earth orbiting missions. Capabilities include: trajectory optimization and analysis, operational orbit determination, flight path control, and 2D/3D visualization. MONTE is presented to the user as an importable Python language module. This allows a simple but powerful user interface via CLUI or script. In addition, the Python interface allows MONTE to be used seamlessly with other canonical scientific programming tools such as SciPy, NumPy, and Matplotlib. MONTE is the prime operational orbit determination software for all JPL navigated missions.

Keywords Navigation · Optimization · Orbit determination · Trajectory · Software

1 Introduction

Prior to the creation of MONTE [1], navigation at JPL was performed using the software suite known as DPTRAJ/ODP [2, 3]. DPTRAJ/ODP is a FORTRAN-based software-set that originated in the late 1960s and expanded over the next three decades. During this same period, mission analysis and design tools were developed independently from DPTRAJ/ODP in a suite of tools known as the Mission analysis Software Library. Moreover, the JPL mission design analysts who used these design tools were an entirely distinct group of individuals from JPL navigation analysts. Transporting mission design models into the navigation world became a small cottage industry unto itself. Meanwhile, computing technology and software development practices in the outside world evolved rapidly. New technology such as the rise of object-oriented (OO) programming rendered many of the computing constraints imposed by early 1970s technology irrelevant. New development practices exposed weaknesses in the established ways of development. Finally, in 1998 the strategic decision was made to begin again and develop a new set of software tools that would exploit the gains in computing and understanding of how to implement code. The new system would start from a blank sheet of paper but rely upon the legacy tools to define correct behavior and define the mathematical specification of essential algorithms. The result of this decision is MONTE.

2 Development considerations

The development of a replacement for DPTRAJ/ODP required consideration of a number of programmatic and infrastructure topics. At the programmatic level these included implementation process, stakeholder concerns, efficiency of implementation, the inevitable growth in requirements, and the long-term sustainability and correctness of the new software.

This paper is based on a presentation at the 6th International Conference on Astrodynamics Tools and Techniques, March 14–17, 2016, Darmstadt, Germany.

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

✉ Scott Evans
scott.evans@jpl.nasa.gov

1 Jet Propulsion Laboratory, California Institute of Technology, Pasadena, USA

2.1 Development infrastructure

When DPTRAJ/ODP was developed, the open-source movement was in its infancy. However, by the late 1990s open-source software had become the foundation of almost all computing. Open source would provide the computing infrastructure for MONTE and would be a major design consideration during development. All third-party dependencies would include source code. The open-source Linux operating system was chosen as the development and operational environment for MONTE.

From the outset it was determined that MONTE would be developed in an object-oriented language. While there was consideration of JAVA, C++ was adopted as the backbone of the implementation. Moreover, it was determined that the new system would allow users access to all levels of the implementation and that the system should be presented as a "software toolbox" similar to the successful SPICE system [4] that had already achieved wide adoption in the planetary science community. It was also recognized that compiled languages do not present users with a flexible, rapid-analysis system. For rapid analysis a scripting language is ideal. A number of options were considered, but the Python language with its OO interface, wide cross-platform support, extensive suite of built in and third-party languages, and ability to "bind" to compiled libraries made it a natural choice. Indeed, Python turned out to be far better choice than initial expectations.

2.2 Development practices

The implementation of MONTE was coincident with JPL's commitment to software process improvement. The development team adopted a number of new practices. Senior management recognized the importance of sound software development processes. The CMMI process model for software engineering was selected as the standard against which JPL's processes would be measured. The goal of achieving a CMMI maturity level 3 certification was established and the MONTE development team was selected as the pathfinder in achieving this goal. This selection was partly due to the strong process focus already adopted by the development team and partly as a forcing function to shore up those areas where weaknesses would be identified. The MONTE development project was the first to be appraised at JPL in 2004 and achieved a CMMI maturity level 2 rating. In 2007 MONTE was among those projects in JPL's first CMMI maturity level 3 rating.

On the road to the maturity level 3 rating, the team adopted or developed a number of practices:

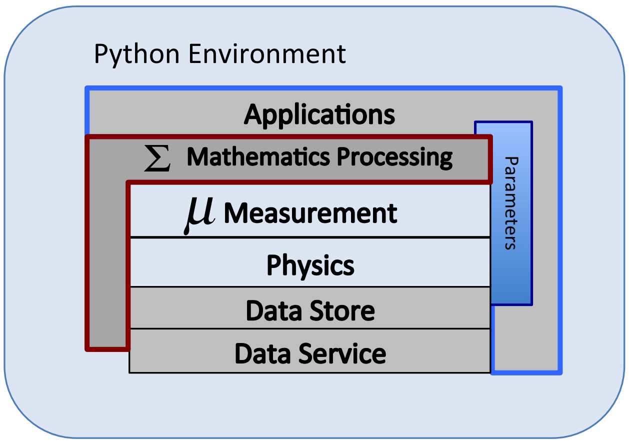

Fig. 1 MONTE architecture

- All new code requires configuration-managed unit tests and must meet code coverage requirements.
- Documentation for the online documentation is included in the internal comments of the source code. The reference documentation is built from these comments.
- Source code is required to pass a style checker.
- A "clean" build of the entire system is performed nightly in which style rules are checked, all unit and system level tests are run, the documentation is built, code examples are executed and checked for correctness, and a static memory check is performed.
- Defects are reported and managed via the open-source bug tracker *bugzilla*.
- The scope of each delivery is determined through a collaboration with the users. The cost of new features is estimated and the entire list of "desirements" presented to the users. Through this process the varied desires of the user community are balanced against the resources available for their development. As a result, the development team routinely delivers features on schedule and on budget.
- Stakeholder communications are managed through a number of regular status meetings as well as an online forum where users and developers can post questions, problems, and solutions.

3 Architecture and design patterns

At the highest level view, MONTE is a large, importable Python module. Importing MONTE into Python expands the Python ecosystem adding the unique astrodynamic features of MONTE. As such it can be combined with the vast set of built-in Python capabilities as well as the rich set of third-party tools such as NumPy, SciPy and Matplotlib.

The MONTE module is a modification of a layered architecture as shown in Fig. 1. The layers in this architecture are:

- the file system layer,
- the data store layer,
- the physics layer,
- the measurement layer,
- the mathematical processing layer,
- the application layer,
- the parameters layer.

The layers of the MONTE architecture are not arranged like the layers of a cake from top to bottom. Instead, as the figure illustrates, some layers cross other layers. For example, the user interface layer has access to all layers of the architecture as well as the surrounding Python environment. Where the layers touch each other in the figure show possible data communications via the Binary Object Archive, or BOA, database system. See Sect. 3.2.

3.1 The data service layer

At the base level is the data service layer. It is at this level that externally produced data products (radiometric observations, planetary ephemerides, earth orientation parameters, etc.) come into the MONTE environment. Although these data products are typically files, this layer also includes telemetry streams and connections to other data providers. Each data product has its own unique format custom software to read the data and form the objects that MONTE uses to carry out its computations.

3.2 The data store layer

MONTE is an object-oriented system. Each object requires a construction step. Once constructed, the methods of that object can be employed to carry out the various computations required of MONTE. Storage and retrieval of these objects is necessary for many large-scale computations. Rather than reconstructing these objects from the original unstructured data, MONTE provides an object storage layer called the binary object archive (BOA). Built upon XDR, BOA provides a machine-independent mechanism for instantiating, storing, and transporting the objects that provide the MONTE computational capabilities.

3.3 The physics layer

The layer above the data storage layer is the physics layer of MONTE. This layer contains the models for the astrodynamics of the solar system. The physics layer provides:

- coordinate systems for specifying the position and velocity of objects,
- the time system used to specify the moments of events, the span of time between events, time systems and the transformations between them (e.g., UTC, TT, TAI, and the detection of events determined by dynamics),
- the trajectory system giving time dependent position, velocity and acceleration of spacecraft and natural bodies, the coordinate frame system that specifies the time-dependent orientation of bodies and celestial frames,
- the force system for modeling the physical forces acting upon objects,
- spacecraft modeling of shape, reflectance, drag, propulsion, etc.,
- solar flux,
- planetary atmospheres, albedos.

3.4 The measurement layer

Our instruments (deep space antennae, cameras, etc.) do not observe the parameters that define our models in the real world. Rather, they sample some function that is dependent upon those parameters. The measurement layer implements the models for those observations as a function of physical parameters as well as their sensitivity to those parameters. In addition, it performs residual calculations associated with actual measurements. Measurement models are supplied for:

- 1-way, 2-way, and 3-way radio Doppler measurements,
- 1-way, 2-way, and 3-way radio range measurements,
- very long baseline interferometry (VLBI),
- DSN station angles,
- optical (line/pixel),
- GPS phase and pseudo range,
- accelerometer,
- spacecraft torque.

Users may construct their own measurement models via Python. In addition, a number of mathematical measurement types are provided such as instantaneous range, rate, and acceleration.

3.5 The mathematical processing layer

The mathematical processing layer includes propagators that perform the numerical integration of the differential equations that describe various astrodynamic models, filters for performing orbit determination, optimizers, Monte Carlo tools, etc. These mathematical tools interact with the measurement and physics layers. It manipulates those layers through the parameters layer. Parameters are modeled as objects in the BOA system and can thus have their data read from and written to by underlying layers. See Sect. 3.4. Note that the governing mathematical models are the same as those found in the NAIF/SPICE software. SPICE kernels can be read natively into MONTE.

3.6 The parameters layer

The models implemented in the physics and measurement layers of MONTE are functions of a large set of parameters. Depending upon user needs, some parameters are treated as variables with associated sensitivities and uncertainty. The parameters layer provides the naming, grouping and selection of sets of parameters used in higher-level computations. The mathematics, physics and measurement layers interact through the use of the parameters layer.

3.7 The application layer

While access to the various layers of the MONTE architecture through the Python language provides maximum flexibility for users, it is not practical to require users to build their operational environment from these lower level objects. Instead, the development team has worked with the user community to identify common workflow patterns. These patterns have been encapsulated as Python callable applications that harness the capabilities in the lower layers of the architecture.

3.7.1 UI system

MONTE's UI system provides a set of data setup and run commands that support the "lockfile-update-run" style of navigation operations. This interface is used extensively by flight projects for operational orbit determination and flight path control. The UI system is implemented as set of command-line Python objects that simplify complex data construction and editing.

3.7.2 COSMIC multi-leg trajectory optimization

The MONTE application COSMIC supports trajectory optimization via a control-point/break-point problem structure. Through COSMIC, users specify a cost function to minimize (usually total delta-V for a mission) over a series of independently propagated trajectory segments. The supported optimizers include SNOPT and DBLSE (JPL internal legacy optimizer).

3.7.3 Trajectory differential corrector

MONTE provides a differential corrector [5] for optimizing trajectories using a patch-point framework. The differential corrector has two main use cases.

The first is for connecting together and optimizing a series of analytically derived states defining a trajectory. For instance, MONTE's HaloRich and LissajousRichCary classes can be used to produce a set of states analytically approximating Halo and Lissajous orbits, respectively. These states can then be processed by the differential corrector to produce a smooth, fully integrated trajectory.

The second main use case is to alter a pre-existing trajectory so that it conforms to a different set of constraints. For instance, one may want to alter a particular leg of a reference trajectory to target a new flyby aimpoint.

3.7.4 Horizons small body ephemeris interface

The JPL Horizons system [6] is an online ephemeris system for natural bodies in the solar system. MONTE's Horizons interface is designed for fast access to the small bodies (comets and asteroids) in the Horizons system.

3.7.5 Mean element integration with morbiter

Morbiter is a utility that allows the integration of the equations of motion for the elements of one or more spacecraft in closed orbits around a central body. In addition to the standard point mass gravity from the central body perturbations by non-spherical gravity, non-central point mass bodies, drag, and solar radiation pressure are supported. Finally, conversions between osculating and mean elements [7] are provided.

3.7.6 Residual viewing and editing

MONTE's residual viewer application supports both the real-time visualization of streaming tracking data (the real-time residual viewer, or RTRV) and the batch-mode visualization and editing of accumulated tracking data (residual viewing and editing tool, or RVET) (Fig. 2).

3.7.7 Landing sites analysis tool

The landing sites analysis tool is a Python GUI application used to display a region on a planetary body that may contain one or more potential landing sites. From an initial, probabilistic landing dispersion, the application identifies landing targets that afford a minimum expectation of landing on a hazard. The tool is designed to display the location of various hazards, probabilities of landing on a hazard, and other types of information (Fig. 3).

3.7.8 Timeline and scheduling tool

The timeline and scheduling tool is a multi-purpose utility for visualizing meetings, events, tracking passes, personnel,

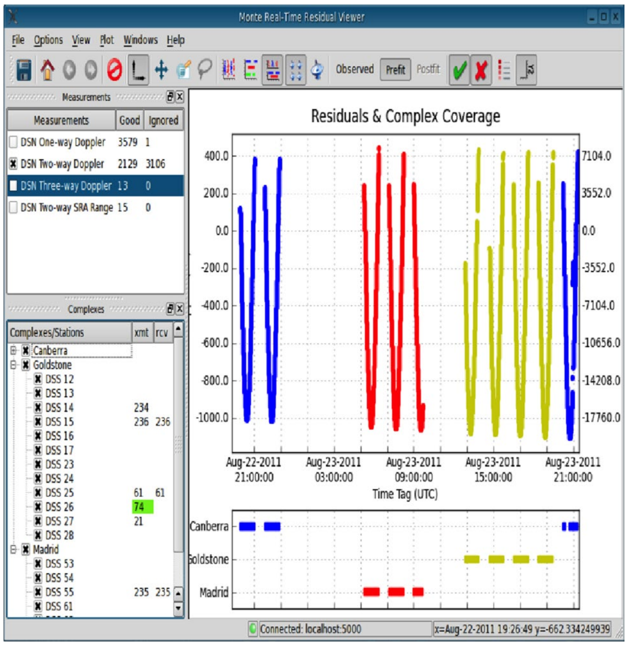
Monte Real-Time Residual Viewer
File Options View Plot Windows Help
Observed Prefit Postfit
Measurements
Measurements Good Ignored
DSN One-way Doppler 3579 1
DSN Two-way Doppler 2129 3106
DSN Three-way Doppler 13 0
DSN Two-way SRA Range 15 0
Complexes
Complexes/Stations xmt rcv
Canberra
Goldstone
DSS 14 234
DSS 15 236 236
DSS 25 61 61
DSS 26 74
DSS 27 21
Madrid
DSS 55 235 235
Residuals & Complex Coverage
Time Tag (UTC)
Canberra
Goldstone
Madrid
Connected: localhost:5000

Fig. 2 Residual viewing and editing

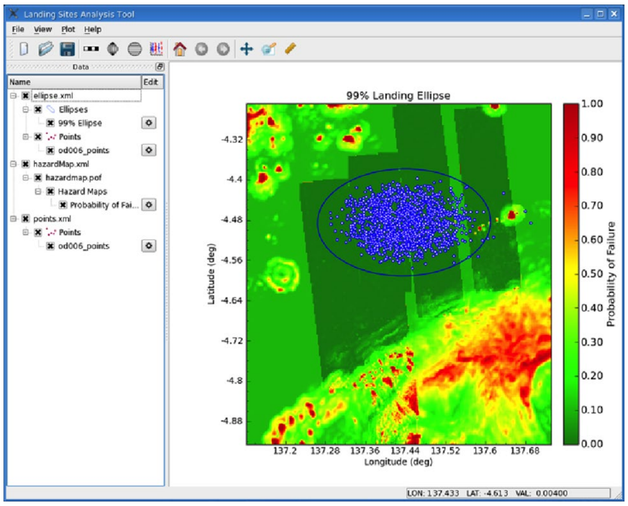

Fig. 3 Landing sites analysis

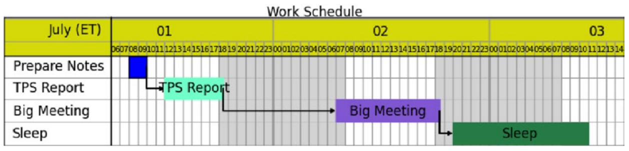

Fig. 4 Timeline and scheduling tools

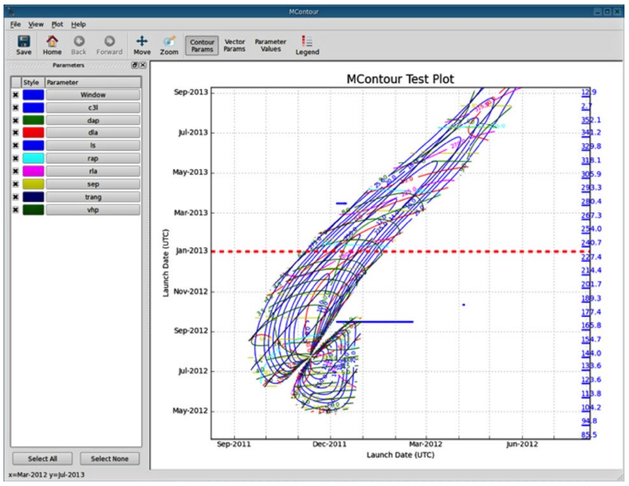

Fig. 5 Launch contour plotting

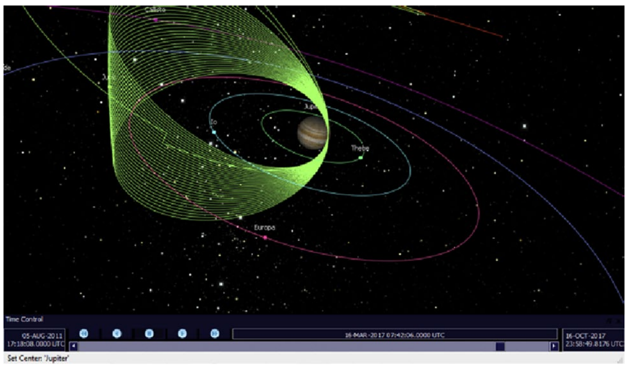

Fig. 6 3-D rendering and exploration of trajectories

etc. on a Gantt-like chart form. Using the Python interface, operations teams can link up-to-date mission dynamics together with mission schedules in an on-demand basis greatly simplifying operations scheduling (Fig. 4).

3.7.9 Launch contour analysis tool

MONTE's launch contour analysis tool (MContour) is used for generating launch contour plots ("porkchop" plots) corresponding to various launch/arrival date parameters (Fig. 5).

3.7.10 Measurement simulation toolbox

The measurement simulation toolbox provides utilities for creating simulated tracking data for orbit determination covariance analysis.

3.7.11 3D trajectory viewer

MONTE's 3D trajectory viewer is a lightweight trajectory visualization application that renders MONTE trajectories in a three-dimensional astrometric environment (Fig. 6).

3.8 Additions to the Python environment

The MONTE system lives within a larger Python environment. This environment has been extended to include unit safety when specifying physical constants, a plotting style system that interacts with the third-party plotting library matplotlib, a framework for performing parallel processing across a cluster of computers, modules that allow users to define their own measurements and forces, and a default set of data for generic studies.

4 The MONTE ecosystem

Beyond the Python environment there is a substantial infrastructure that supports MONTE development and the user community.

4.1 Documentation

MONTE comes with an extensive set of documentation presented as a cross-linked website comprising well over 1000 pages. The documentation provides:

- an overview of the key concepts of the system,
- tutorial examples to help users learn basics of the system,
- detailed descriptions of algorithms implemented in MONTE,
- low level class descriptions, relevant equations, and examples of usage,
- links to release information,
- links to training videos,
- links for downloading PDF User Guides,
- links to the NAIF and Horizons homepages,
- links to third-party documentation.

The documentation website is built from a configuration-managed collection of source pages. Crosslinks are automatically generated and equations are built from LaTex imbedded in the comments of source code as well as text-based source pages. The MONTE software and its documentation are built at the same time within the MONTE build framework. This ensures consistency between the product and the documentation that describes it.

4.2 Daily build and test

MONTE has adopted a lifecycle that mixes the agile development model with a more traditional iterative development lifecycle. Central to this is the daily “night build.” Each night MONTE and its documentation are automatically built from source on a “clean” platform—an environment that has no predefined environment variables or custom libraries. Source code and documentation source are checked for style violations. MONTE is compiled with all compiler warnings enabled. All unit tests are executed and checked for failures. Unit tests are checked to determine that they adequately cover all functions and execute all source lines. Examples in documentation are executed and results checked for correctness. All system level tests are performed and checked for correctness. The results of the build are published to a webpage for examination on the following morning. Any problems identified are fixed by the development team before continuing with new development.

5 Verification and validation

MONTE was designed to replace the legacy orbit determination software at JPL. The legacy software had an established record of correct performance. The software had been used to navigate every JPL mission since 1971, more than 17 missions. MONTE needed to be as reliable as the software it replaced. This requirement influenced the development of MONTE from the beginning of its implementation.

5.1 The MONTE test system

The MONTE system has approximately 2.2 million logical lines of source code. The deliverable portion of the MONTE system (the portion the user sees) accounts for only about 30% of this number. The other 70% is code used to test the functionality of the system. A small portion of this is the code developed for the test harnesses that enable nightly automated testing. The rest is the code for unit and system testing of MONTE.

5.1.1 Unit testing

The MONTE development process requires that every function in the underlying C++ and Python libraries have a test case that can be run and render a pass/fail determination on the function being tested. In addition, tests are required to exercise 90% of the source lines of any function or gain a waiver from the project architect (typically 100% code coverage is achieved). The unit test system has over 3500 such tests. Tests of low level functions and classes can often be developed to confirm a computation “offline” by simple derivation. However, more complicated tests such as the value of a computed measurement or numerical integration are developed by obtaining a “correct” result as determined by the legacy software. This “correct” result is then compared to the result obtained by the MONTE implementation. The comparisons are expected to agree to numerical round-off or be accompanied by an explanation for differences.

5.1.2 System testing

While unit testing catches a large number of errors, they do not detect errors arising from the interaction of system components such as the quality of fit of a navigation solution or the trajectory determined by an optimization run. For these tests, the development team captures operational results generated by navigation and mission design users. These results are packaged to be compatible with the MONTE system test harness and placed under configuration management. These end-to-end cases are then run as part of the nightly regression tests and as part of every delivery of the system to the user community.

5.1.3 Defects

In spite of the efforts to deliver an error-free product to users, bugs are present in the delivered product. When users discover an apparent bug, the bug is reported using the open-source, web-based bug tracking tool *bugzilla*. The next step in the life of a reported bug is the development of a test by the development team that would pass if the code functioned as intended, but fails due to the presence of the bug in the system. This new test becomes part of the test system. The development team then implements a repair to the software so that the system passes the bug test. All other regression tests are also performed to ensure that the repair did not break some other aspect of the software. The repaired software is made available to the user who first reported the problem. That user then confirms that the repaired software functions as expected. As a result of this process, the software converges relatively quickly to a system that meets its intended capabilities.

5.2 Delivery

MONTE is under continual development. New features are requested by the user community on a regular basis. New development comes with a new set of tests that ensure that the software is largely correct once the new features are deemed ready for use by the user community. However, before the software is placed into operations, a testable version of the software is made available and a subset of the user community agrees to use the software in the planned operational sense. This form of testing goes beyond the imagination of the development team in the use of the software. Only after a period sufficient for the user representatives to exercise the software (and have any discovered bugs repaired) is the software delivered into operational service.

6 Operations, adoption, and the future of MONTE

MONTE was first used for operations in 2007, for the Phoenix Mars Lander. Since that time it has gradually replaced the legacy orbit determination software. The last project to transition from the legacy software to MONTE was the Cassini mission in 2012.

The transition from legacy mission design tools to MONTE began informally around 2004 but many of the features needed by the mission design community were not implemented until the completion of the navigation transition. Today, majority of the mission design community has moved away from the legacy design tools and have adopted MONTE as the tool of choice when designing mission trajectories.

Monte is in active development. It will continue to address flight project needs as well as the needs of the mission design community in supporting research, proposals, and pre-flight analysis. The project is currently working on a 3D plotting capability, a SRIF (square root information) filtering capability, and refining its parallel processing capability. The system is fully supported for the Red Hat Enterprise 7, 64-bit Linux OS and beta versions exist for Windows 10 and Mac OS X 10.11 and newer. These will become fully supported versions in the near future.

MONTE is ITAR-free and a version with full capability apart from the ability to generate navigation solutions in operations is EAR 99. This means that the software can be distributed nearly worldwide pending a straightforward licensing process. Direct inquiries about obtaining the software to: mdn_software@jpl.nasa.gov.

Acknowledgements Funding was provided by Jet Propulsion Laboratory.

References

1. Sunseri, R., Wu, H.-C., Evans, S., Evans, J., Drain, T., Guevara, M.: Mission analysis, Operations, and Navigation Toolkit Environment (MONTE) version 040. NASA Tech Briefs **36**(9), 2012
2. Moyer, T.: Formulation for Observed and Computed Values of Deep Space Network Data Types for Navigation. John-Wiley & Sons, Inc., Hoboken (2003)
3. Moyer, T.: Mathematical Formulation of the Double-Precision Orbit Determination Program (DPODP), TR 32-1527 Jet Propulsion Laboratory, Pasadena (1971)
4. Acton, C.H.: Ancillary data services of NASA's navigation and ancillary information facility. Planet. Space Sci. **44**(1), 65–70 (1996)
5. Marchand, B.G., Howell, K.C., Wilson, R.S.: An improved corrections process for constrained trajectory design in the n-body problem. J. Spacecr. Rocket. **44**(4), 884–897 (2007)

6. Giorgini, J.D., Chodas, P.W., Yeomans, D.K.: Orbit uncertainty and close-approach analysis capabilities of the Horizons on-line ephemeris system. In: The 33rd Meeting of the AAS Division for Planetary Sciences, Bulletin of the American Astronomical Society, vol. 33, p. 1562. New Orleans, LA (2001)
7. Ely, T.A.: Mean element propagations using numerical averaging. J. Astronaut. Sci **61**(3), 275–304 (2014)