NASA/TM–20210014622
NESC-RP-18-01309

Improvements to the Flight Analysis and Simulation Tool (FAST) and Initial Development of the Genesis Flight Mechanics Simulation for Ascent, Aerocapture, Entry, Descent, and Landing (A2EDL) Trajectory Design

Daniel G. Murri/NESC
Langley Research Center, Hampton, Virginia

Daniel A. Matz and David A. Hoffman
Johnson Space Center, Houston, Texas

Jon S. Berndt abd Susan C. Brown
Jacobs Technology, Inc., Houston, Texas

Lorraine E. Prokop
Johnson Space Center, Houston, Texas

April 2021

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- Help desk contact information:

https://www.sti.nasa.gov/sti-contact-form/
and select the "General" help request type.

NASA/TM–20210014622
NESC-RP-18-01309

Improvements to the Flight Analysis and Simulation Tool (FAST) and Initial Development of the Genesis Flight Mechanics Simulation for Ascent, Aerocapture, Entry, Descent, and Landing (A2EDL) Trajectory Design

Daniel G. Murri/NESC
Langley Research Center, Hampton, Virginia

Daniel A. Matz and David A. Hoffman
Johnson Space Center, Houston, Texas

Jon S. Berndt abd Susan C. Brown
Jacobs Technology, Inc., Houston, Texas

Lorraine E. Prokop
Johnson Space Center, Houston, Texas

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

April 2021

The use of trademarks or names of manufacturers in the report is for accurate reporting and does not constitute an official endorsement, either expressed or implied, of such products or manufacturers by the National Aeronautics and Space Administration.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

NASA Engineering and Safety Center
Technical Assessment Report

Improvements to the Flight Analysis and Simulation Tool (FAST) and Initial Development of the Genesis Flight Mechanics Simulation for Ascent, Aerocapture, Entry, Descent, and Landing (A^2EDL) Trajectory Design

April 8, 2021

Report Approval and Revision History

NOTE: This document was approved at the April 8, 2021, NRB. This document was submitted to the NESC Director on April 12, 2021, for configuration control.

Approved:	*Original Signature on File*	4/12/21
	NESC Director	Date

Version	Description of Revision	Office of Primary Responsibility	Effective Date
1.0	Initial Release	Daniel G. Murri, NASA Technical Fellow for Flight Mechanics, LaRC	4/8/21

Table of Contents

Technical Assessment Report

List of Figures

List of Tables

Technical Assessment Report

1.0 Notification and Authorization

The NASA Engineering and Safety Center (NESC) was requested to provide significant enhancements to the Flight Analysis and Simulation Tool (FAST), a generic, variable-degree-of-freedom (DOF), multi-body ascent, aerocapture, entry, descent, and landing (A^2EDL) flight simulation code and a key Agency analysis tool. The primary stakeholders for this assessment are the Commercial Crew Program (CCP), the Orion Multi-Purpose Crew Vehicle (MPCV) Program, the Orion Capsule Parachute Assembly System (CPAS) project, the Human Landing System Program, and future missions to the Moon and Mars.

NESC Lead	Daniel Murri, NASA Technical Fellow for Flight Mechanics
Technical Co-Lead	Daniel Matz, Aerospace Engineer
Technical Co-Lead	Frank Monahan, Johnson Space Center (JSC) Flight Mechanics and Trajectory Design Branch Chief
Approval to Proceed	March 15, 2018
Assessment Plan	May 3, 2018
Final Report	April 8, 2021

2.0 Signature Page

Submitted by:

Team Signature Page on File – 4/21/21

Mr. Daniel G. Murri Date

Significant Contributors:

Mr. Daniel A. Matz Date	Mr. David A. Hoffman Date
Mr. Jon S. Berndt Date	Ms. Susan C. Brown Date
Ms. Lorraine E. Prokop Date	

Signatories declare the findings, observations, and NESC recommendations compiled in the report are factually based from data extracted from program/project documents, contractor reports, and open literature, and/or generated from independently conducted tests, analyses, and inspections.

3.0 Team List

Name	Discipline/Function	Organization
Core Team		
Dan Murri	NESC Lead	LaRC
Daniel Matz	Technical Co-lead	JSC
Frank Monahan	Technical Co-lead	JSC
David Hoffman	Technical Support	JSC
Jon Berndt	Development Support	JSC Engineering, Technology, and Science (JETS)
David Grismore	Development Support	JETS
Susan Brown	Development Support	JETS
Consultants		
Michael Aguilar	Former NASA Technical Fellow for Software	LaRC (retired)
John Carson	Interface to Flight Software/Hardware in the Loop	JSC
Neil Dennehy	NASA Technical Fellow for Guidance, Navigation, and Control (GN&C)	GSFC
Joe Guinn	Interface to JPL Analysis Tools	JPL
Steve Hughes	Interface to GSFC Analysis Tools	GSFC
Mark Ivanov	Descent Guidance	JPL
Breanna Johnson	Interface to Modular Robotic Vehicle	JSC
Chris Madsen	Interface to CPAS and Orion Flight Dynamics	JSC
Gavin Mendeck	Interface to CCP Flight Dynamics	JSC
Jeremy Rea	Interface to Orion Entry	JSC
Business Management		
Linda Moore	Program Analyst	LaRC/MTSO
Assessment Support		
Linda Burgess	Planning and Control Analyst	LaRC/AMA
Jonay Campbell	Technical Editor	LaRC/KBR
Melinda Meredith	Project Coordinator	LaRC/AMA

4.0 Executive Summary

NASA's Flight Analysis and Simulation Tool (FAST) is a generic, multi-vehicle, variable-degree-of-freedom (DOF) flight mechanics simulation. A key Agency objective is to obtain desired flight characteristics for vehicles used in accomplishing Agency missions and to ensure performance requirements are met. FAST is capable of simulating ascent, aerocapture, entry, descent, and landing (A^2EDL) trajectories and provides the functionality of several legacy simulation tools. It is built using the Trick Simulation Environment, a framework for building simulations that generates the executive and provides various utilities like input processing and data recording, and it leverages the Johnson Space Center (JSC) Engineering Orbital Dynamics (JEOD) library, a collection of commonly used dynamics and environment models. FAST is used by the Orion Multi-Purpose Crew Vehicle (MPCV) Capsule Parachute Assembly System (CPAS) project, the Orion MPCV aerosciences team, the Commercial Crew Program (CCP) parachute team, and for other projects and academic collaborations within the JSC Flight Mechanics and Trajectory Design Branch.

The near-term goals of this assessment were to address a backlog of needed FAST upgrades, including updating to the latest versions of Trick and JEOD and migrating to a new computing cluster, expanding the FAST user documentation, and certifying the CPAS project version of FAST as a NASA Procedural Requirement (NPR) 7120.2C Class C software. This assessment was also intended to contribute toward JSC's long-term goal to fully consolidate and replace several legacy flight mechanics simulations, chiefly by developing an optimization capability.

This report details the upgrades to FAST that were completed, describes a change to the assessment's scope that allowed its goals to be exceeded with a new and innovative approach, and discusses the results of the modified approach. The rescoped assessment resulted in the initial development of Genesis, a flight mechanics simulation and trajectory design tool, which overcomes the limitations of FAST.

The assessment was rescoped when it became evident that the architecture of FAST, specifically its use of Trick and JEOD, precluded completion of the assessment's goals. In particular, the goal of creating a generic optimization capability could not be met without substantial changes to Trick. The assessment was redirected to create a successor to FAST. The rescope provided the opportunity to take advantage of the programming language Julia, a new language designed for technical computing that combines the ease of use of scripting languages with the performance of compiled languages.

During the first year of the assessment prior to the rescope, the near-term goals were addressed, including upgrading FAST to the latest versions of Trick and JEOD, transitioning FAST and its users to the JSC Flight Sciences Laboratory (FSL) computing cluster, implementing several new guidance algorithms, certifying FAST as NPR 7120.2C Class C software to support the CPAS project's requirements verification, improving the onboarding of new users by expanding the FAST user guide and creating several new example cases, and validating FAST against the NASA Engineering and Safety Center (NESC)-published simulation verification cases [refs. 1–3]. With these tasks completed, FAST was prepared to meet the needs of the CPAS project and the CCP parachute team.

After the rescope, the assessment focused on the creation of Genesis, a multi-vehicle, variable-DOF flight mechanics simulation and trajectory design tool written in Julia. The architecture of Genesis makes it more flexible, capable, and performant than FAST and allows a straightforward implementation of trajectory optimization. Some existing FAST models were reused by creating Julia wrappers that call the original C, C++, or Fortran code. For others, the formulation of the FAST model was reused, but a Julia implementation was made. Onboarding of new users was simplified, as Julia is an easy-to-learn, high-level programming language. Extensive documentation, tutorials, unit tests, and example cases were created with Julia’s built-in tools. Genesis was verified against existing FAST and SORT (Simulation for the Optimization of Rocket Trajectories) cases and NESC-published simulation verification cases. As a result of this initial development effort, Genesis is ready to support upcoming programs and projects with A^2EDL flight mechanics and trajectory design products. The ongoing development of Genesis will expand its capabilities and complete the consolidation of legacy flight mechanics simulations.

In summary, the upgrades to FAST and the subsequent initial development of Genesis produced significant improvements to key areas of flight mechanics simulation and mission trajectory design and analysis. FAST is prepared to successfully meet the needs of the CPAS project and the CCP parachute team. The assessment was redirected when the NESC team determined that FAST was not suitable for A^2EDL trajectory design. Genesis improves upon FAST by being more amenable to trajectory optimization and modern high-performance computing (HPC) environments. It leverages Julia to improve ease of use and run-time performance. Genesis can be incorporated into Copernicus, an exo-atmospheric and interplanetary trajectory design and optimization tool, for end-to-end mission planning. Genesis is more capable, faster, and easier to maintain than FAST.

5.0 Assessment Plan

The NESC was requested to provide critical enhancements to FAST to ensure its capabilities in providing flight performance analysis, algorithm development, and flight software contributions for NASA programs and projects. The technical activities and objectives of this NESC assessment were to:

1. Mature the models, algorithms, and analysis techniques required to support the Human Exploration and Operations Mission Directorate (HEOMD) for CCP, the HEOMD and Science Mission Directorate for lunar, Mars, and other planetary exploration, and the Space Technology Mission Directorate (STMD) for technology and flight test activities.
   a. Upgrade to the latest version of Trick and JEOD and transition to JSC's FSL computing cluster.
   b. Improve 3- and 6-DOF powered ascent capabilities for the Moon and Mars.
   c. Create a generic optimization capability.
   d. Incorporate new vehicle, environment, and aerodynamic models for CCP and the Moon and Mars.
   e. Implement heritage and new flight guidance algorithms, control systems, and navigation models.
2. Certify FAST as NPR 7120.2C Class C software to support the CPAS project requirements verification.
   a. Expand unit testing.
   b. Improve model documentation.
   c. Clean up existing code as necessary.
3. Improve onboarding of new users.
   a. Create a detailed user's guide.
   b. Develop a suite of example cases based on historical vehicles.
4. Compare FAST to other simulations.
   a. Compare to published verification cases [refs. 1–3].
   b. Compare to Program to Optimize Simulated Trajectories II (POST2) cases.

After the first year of the assessment, the high-priority, near-term work on FAST was complete, and FAST was ready to meet the needs of its current customers, as described in Section 6.3. However, some of the upgrades, onboarding of new developers, and implementation of features (e.g., optimization) were hampered by difficulties working with the Trick framework and the JEOD library. These issues, discussed in Section 6.4, revealed that the assessment's original plan for the long-term development of FAST was not feasible.

As a result, the remainder of the assessment was rescoped to pursue the following technical activities and objectives:

1. Migrate environment models, vehicle models, and guidance algorithms from FAST to Julia.
2. Create a prototype flight mechanics simulation and trajectory design tool in Julia.
   a. Demonstrate a basic 3-DOF capability.
   b. Demonstrate optimization and other features missing from FAST.

3. Expand the prototype.
    a. Add high-fidelity lunar orientation.
    b. Add high-fidelity gravity.
    c. Add 6-DOF dynamics.
    d. Improve onboarding of new users.
    e. Create a detailed user’s guide.
    f. Develop a suite of example cases.
4. Compare the new flight mechanics simulation and trajectory design tool to other simulations.
    a. Compare to published verification cases [refs. 1–3].

6.0 Flight Mechanics and Trajectory Design Tool Enhancements

6.1 Legacy JSC Flight Mechanics Simulations

The JSC flight mechanics simulations and trajectory design tools are typically used for a broad range of studies. These studies include developing Design Reference Mission (DRM) trajectories for conceptual vehicle designs, evaluating or prototyping guidance algorithms, designing and reconstructing test flights, evaluating vehicle performance, and designing trajectories for each mission.

Historically, JSC flight mechanics personnel have relied on legacy tools from the Space Shuttle Program era or earlier because they were small and simple enough to be modified by users as needed. This allowed a single engineer to quickly respond to requests without requiring support from a separate team of developers. However, these tools were designed for other purposes, which limits their utility. New engineers have difficulty understanding their implementation, and maintaining the tools was becoming increasingly difficult. The case for a better option was compelling and led to the original development of FAST in 2011.

6.2 Description of FAST

The original goal of FAST was to consolidate and replace several legacy programs:

1. Simulation for the Optimization of Rocket Trajectories (SORT), a 3-DOF flight mechanics simulation and trajectory optimization tool.
2. Decelerator System Simulation (DSS), a 6-DOF, multi-body flight mechanics simulation used primarily for its parachute modeling capabilities.
3. General Electric Missile and Satellite Simulation (GEMASS), a 6-DOF atmospheric flight simulation used primarily for aerodynamic stability analysis of entry vehicles.

To meet this objective, FAST needed to generate trajectories for both powered and atmospheric flight. It needed a configurable number of degrees of freedom and needed to provide optimization and Monte Carlo capabilities.

FAST was built on top of the Trick Simulation Environment and the JSC Engineering Orbital Dynamics (JEOD) library. Trick is a framework for building simulations that generates the executive and provides various utilities like input processing and data recording. JEOD is a collection of commonly used dynamics and environment models. The decision to use Trick and JEOD was based on two perceived benefits. First, because JEOD provided all the models necessary for simulating 6-DOF vehicles in orbit, it was hoped that the only additional development required would be the creation of models for atmospheric flight. Second, since programs at JSC generally use Trick and JEOD for their requirements verification simulations, it was thought that using these tools in FAST would provide a natural pathway for sharing project-specific code.

After several years of development, FAST had become a capable and flexible tool: a generic, multi-body, variable-DOF, flight mechanics simulation for atmospheric flight largely implemented in C++.

FAST continues to support a wide variety of NASA programs and projects. The CPAS project used FAST for preflight analysis, drop test design, drop test reconstruction, and verification of several requirements by analysis. The MPCV aerosciences team uses FAST to perform aerodynamic reconstruction of drop tests and flight tests. The CCP parachute team analysis team uses FAST to perform post-test reconstructions of partner drop tests and Monte Carlo assessments to assess the performance of the partner parachute systems. The Pterodactyl project at Ames Research Center used FAST for stability analysis, performance analysis, and guidance, navigation, and control (GN&C) development. The Mid Lift/Drag (L/D) Rigid Vehicle project used FAST for entry and aerocapture performance analysis and GN&C development. JSC has used FAST as a testbed for new guidance algorithms developed through academic collaborations.

6.3 Updates to FAST

By 2018, it became apparent that FAST's small, part-time development team needed additional support because the completion of long-term tool consolidation goals was being deferred in favor of the near-term needs of current vehicle development programs, specifically the CPAS project.

The CPAS project's decision to use FAST for their analysis brought in new users, which led to a need to enhance the FAST documentation. When the CPAS project realized that one of its requirements for high-altitude parachute operations could not be verified by test and would need to be verified by analysis, it became necessary to certify FAST as NPR 7120.2C Class C software. In addition, the steady stream of Trick and JEOD updates, combined with the decision to consolidate computing resources in a centralized HPC facility, required many FAST modifications.

The maintenance needs of FAST constrained the time available to work toward the original long-term goal of incorporating the capabilities of legacy tools into FAST. Modelling atmospheric flight with FAST was relatively straightforward, but the same could not be said about powered flight. During powered-flight ascent, a vehicle's mass and inertia tensor change considerably. The dynamics models provided by JEOD assumed constant mass properties and needed to be modified. It became clear that there was no path forward for a small development team and a more focused effort was required.

Based on these needs, the NESC team was assembled to provide a more focused effort to update FAST. In the first year of the assessment, the team accomplished the near-term work that was proposed, including:

- Upgraded FAST to the latest versions of Trick and JEOD.
- Migrated users to the FSL.
- Migrated version control from Mercurial to Git for commonality with other JSC projects.

- Migrated issue tracking to GitLab for commonality with other JSC projects.
- Configured continuous integration (CI) to automatically run tests and deploy documentation.
- Compared FAST against verification cases published by the NESC.
- Integrated new guidance algorithms.
- Created new example cases, including lunar descent and ascent.
- Expanded the FAST user guide.
- Certified the CPAS project's version of FAST as NPR 7120.2C Class C software.

6.3.1 Comparison to the NESC Verification Cases

In 2015, the NESC led an effort to develop a suite of 6-DOF simulation verification cases [refs. 1, 2]. The final report [ref. 3] contains, for each case, a family of solutions that were generated by seven different flight simulations. To verify the FAST equations of motion and illustrate FAST usage, the first ten atmospheric verification cases, summarized in Table 6.3-1, were implemented in FAST.

Table 6.3-1. Atmospheric Verification Cases

Number	Description	Verifies
1	Dropped sphere with no drag	Gravitation, translational equations of motion
2	Tumbling brick with no damping, no drag	Rotational equations of motion
3	Tumbling brick with dynamic damping, no drag	Inertial coupling
4	Dropped sphere with constant drag coefficient, no wind, nonrotating planet	Gravitation, integration
5	Dropped sphere with constant drag coefficient, no wind, rotating planet	Earth rotation
6	Dropped sphere with constant drag coefficient, no wind, zonal-harmonic gravity	Ellipsoidal Earth
7	Dropped sphere with constant drag coefficient, steady wind	Wind effects
8	Dropped sphere with constant drag coefficient, variable wind with shear	Two-dimensional wind
9	Sphere launched eastward along equator	Translational equations of motion
10	Sphere launched northward along prime meridian	Coriolis

The results from FAST were compared with the results generated by JSBSim [ref. 4], a flight simulation used in the original NESC study. For each case, the time histories of position, velocity, attitude, and angular velocity of the spacecraft were compared. The results are summarized in Table 6.3-2. Note that for Case 4 there is a known issue in FAST that prevents the nonrotating planet from being properly configured in FAST, which results in an incorrect position vector time history.

Table 6.3-2. Maximum Absolute Difference Over Time between JSBSim and FAST Verification Case Results

		Position		**Relative Velocity**		**North East Down (NED) to Body Euler Angles**		
Case	**Altitude**	**X**	**Y**	**Y**	**Z**	**Roll**	**Pitch**	**Yaw**
	[ft]	**[ft]**	**[ft]**	**[ft/s]**	**[ft/s]**	**[°]**	**[°]**	**[°]**
1	0.2	0.17	0.08	0.005	0.014	2.10E-07	1.67E-09	1.33E-07
2	0.2	0.17	0.08	0.005	0.014	0.01	0.005	0.002
3	0.2	0.17	0.08	0.005	0.014	0.12	0.25	0.055
4	0.75			6.00E-09	0.08	1.25E-09	8.00E-08	1.00E-09
5	0.75	0.75	3.00E-03	2.25E-04	0.08	1.50E-08	7.00E-08	9.00E-09
6	0.92	0.9	7.60E-02	4.50E-03	0.09	2.00E-07	1.65E-09	1.10E-09
7	0.92	0.9	1.10E-01	7.00E-03	0.09	3.00E-07	8.00E-08	1.10E-09
8	0.92	0.9	1.80E-01	1.20E-02	0.09	5.00E-07	8.00E-08	1.10E-09
9	0.49	0.52	8.70E-01	4.50E-02	0.018	1.10E-09	2.40E-06	8.00E-08
10	0.56	0.52	7.40E-02	5.00E-03	0.018	1.30E-07	4.20E-04	8.00E-08

6.4 Motivation for the Development of Genesis

Following development of the updates described in Section 6.3, FAST was in a better position to meet the needs of its current customers, including the CPAS project and the CCP parachute analysis team. However, the NESC team discovered that further development of FAST faced a number of interconnected challenges.

First, the implementation of several desired features was hindered by the fundamental architecture and inherent assumptions of Trick and JEOD, which were developed primarily for real-time flight simulators and project-specific verification simulations. The team originally chose to leverage the Trick framework and JEOD library with the expectation that this would reduce the effort required to develop FAST. However, changes to JEOD to incorporate the models necessary for atmospheric flight and to improve its mass properties model so that it behaved appropriately during powered flight required a significant effort. The team determined that even more substantial changes to Trick would be necessary to integrate an optimizer into FAST. As the level of effort required to modify Trick and JEOD grew, the case for using them was reduced.

Second, FAST is difficult to maintain. New versions of Trick and JEOD continue to be released, making it difficult for the FAST development team to keep up with the changes. In addition, Trick and JEOD updates are not always compatible with previous modifications. While FAST is primarily used in a Linux environment, support for macOS was desired. However, FAST support for macOS proved to be unreliable. In addition to the maintenance burden, these constant updates and the sensitivity to the versions of library dependencies made it difficult to revisit old analyses.

Third, it is difficult to onboard new engineers who use and develop FAST. Flight mechanics and trajectory design at JSC is best served by engineers who are able to both modify the source code (e.g., add a new aero model or a new guidance) and perform the associated analysis. However, engineers with a flight mechanics background and C++ experience are rare, so new engineers often need to learn C++ as they learn FAST. Trick and JEOD have complex data structures with complex interdependencies, and the more FAST requires modifications to those libraries, the more new engineers have to learn about their complexities.

These challenges prompted the NESC team to pursue a different approach rather than make further upgrades to FAST. While the assessment's near-term goals were achieved and FAST was prepared to meet the needs of its current customers, the ability to efficiently pursue the long-term goals was hindered by these limitations.

The NESC team decided to create a FAST successor, called Genesis. This allowed the team to design a new architecture that enables the desired features. It also freed the team to select a language more appropriate for technical computing and more approachable to engineers without a formal computer science background. Lastly, it allowed the team to apply lessons learned from the development and use of FAST, Copernicus (an exo-atmospheric and interplanetary trajectory design and optimization tool) [ref. 5], and Damocles (a wrapper around Copernicus for performing large scans in parallel) [ref. 6] to better take advantage of modern HPC environments.

6.5 Description of Julia

Julia is a programming language designed for scientific computing [refs. 7 and 8]. It was developed at the Massachusetts Institute of Technology by a group of former MATLAB users. It fills a niche between statically compiled and interpreted scripting languages. Its just-in-time compiler considers execution context when generating code, allowing it to automatically apply a variety of optimizations. The run-time performance of Julia approaches that of C or Fortran. At the same time, Julia is a high-level language with a simple syntax, similar to MATLAB and Python. It has a powerful type system that can be extended with user-defined types. Julia comes with built-in support for vector and matrix operations and other utilities useful for scientific computing. Users proficient with MATLAB and Python can quickly become productive with Julia.

Like most modern languages, Julia is open source. Because it was a distributed, open-source project, it comes with built-in support for the tooling that a modern software development team needs: unit-testing facilities, documentation tools, a package manager with reproducible package environments, and support for a variety of operating systems and central processing unit (CPU) architectures. And even though it is relatively new, it has a rich system of third-party libraries for numerical computation. Julia also provides a variety of parallel computing tools, including multi-threading, multi-processing, and graphics processing unit (GPU) computing.

In scientific computing, it is especially important to be able to leverage existing codes. Julia provides a built-in capability to call C and Fortran code. Packages exist to allow Julia to call code written in Python, MATLAB, Java, R, and other languages.

Julia's features allowed the team to focus on the domain logic rather than low-level computing details.

6.6 Development of Genesis

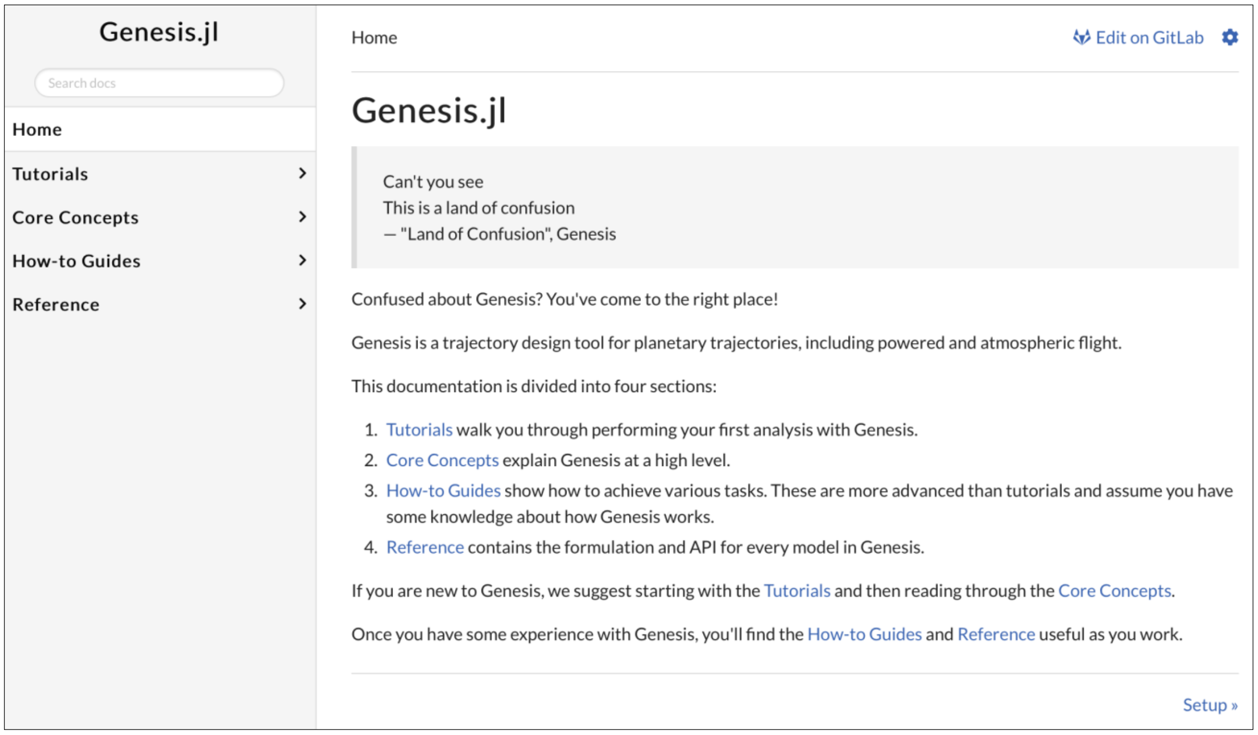
Genesis is the successor to FAST and is implemented in Julia. It is a generic, multi-body, variable-DOF flight mechanics simulation and trajectory design tool for A^2EDL trajectories. Genesis does not rely on Trick and JEOD, and its architecture was designed to better fit NASA's needs, including optimization. Appropriate equations of motion were derived and implemented to provide full support for powered flight and movable mass control systems. Genesis will be NPR 7120.2C Class D software.

Some FAST models were reused by creating Julia wrappers around C, C++, or Fortran code. For other models, it was deemed easier to translate the existing model into Julia. Each model is hosted in its own GitLab repository and has a suite of standalone unit tests, which are automatically run by GitLab's continuous integration system whenever the code is changed. The models available in Genesis include:

- Fixed-step and adaptive-step explicit Runge-Kutta integrators
- Gravity
  - Point mass
  - Spherical harmonic
- Planet orientation
  - Generic
  - High-fidelity lunar orientation
- Atmosphere
  - Earth and Mars Global Reference Atmospheric Models (GRAM)
  - United States Committee on Extension to the Standard Atmosphere (COESA) – U.S. Standard Atmosphere 1976
- Aerodynamics
  - Generic models for symmetric and non-symmetric vehicles
  - Mars Science Laboratory aerodynamics interface
  - Apollo Command Module aerodynamics
- Guidance algorithms
  - Fully Numeric Predictor-corrector Entry Guidance (FNPEG), including bank control and alpha/beta control versions
  - Fully Numeric Predictor-corrector Aerocapture Guidance (FNPAG), including bank control and alpha/beta control versions
  - Dual Quaternion Powered Descent Guidance (DQPDG)
  - Space Shuttle Ascent Powered Explicit Guidance (PEG)
  - Apollo powered descent
  - Tunable Apollo powered descent
  - Fractional polynomial powered descent
  - Quadratic powered descent
  - E-guidance powered descent

- Minimum acceleration powered descent
- Gravity turn descent

The NESC team created an online user guide for Genesis (see Figure 6.6-1), including tutorials and reference material. The online documentation is searchable and provides links to Genesis' application programming interface documentation and GitLab-hosted source code.



Genesis.jl

> Can't you see
> This is a land of confusion
> — "Land of Confusion", Genesis

Confused about Genesis? You've come to the right place!

Genesis is a trajectory design tool for planetary trajectories, including powered and atmospheric flight.

This documentation is divided into four sections:

1. Tutorials walk you through performing your first analysis with Genesis.
2. Core Concepts explain Genesis at a high level.
3. How-to Guides show how to achieve various tasks. These are more advanced than tutorials and assume you have some knowledge about how Genesis works.
4. Reference contains the formulation and API for every model in Genesis.

If you are new to Genesis, we suggest starting with the Tutorials and then reading through the Core Concepts.

Once you have some experience with Genesis, you'll find the How-to Guides and Reference useful as you work.

Setup »

Figure 6.6-1. Screenshot of Genesis Documentation

6.6.1 Comparison to the NESC Verification Cases

As was done with FAST, the first ten NESC atmospheric check cases were run with Genesis, and the time history responses were compared with the results from JSBSim. The full comparison included:

- Position, resolved in the planet-centered inertial frame
- Position, resolved in the planet-centered, planet-fixed frame
- Inertial velocity, resolved in the planet-centered inertial frame
- Planet-relative velocity, resolved in the local north-east-down frame
- Altitude
- Latitude
- Longitude
- Gravitational acceleration
- Euler angles from the local north-east-down frame to the body frame

- Angular velocity of the body frame
- Local atmospheric density
- Local atmospheric pressure
- Local atmospheric temperature
- Aerodynamic force, resolved in the body frame
- Aerodynamic moment, resolved in the body frame

The comparison was excellent for all variables and all cases. A summary of the maximum absolute errors seen for several variables is shown in Table 6.6-1. The check cases were integrated into the GitLab CI system for use as regression test cases. The exercise additionally proved useful in that it showed where improvement could be made in the implementation of one of Genesis' integrators.

Table 6.6-1. Maximum Absolute Difference Over Time between JSBSim and Genesis Verification Case Results

		Position		**Inertial Velocity**		**NED to Body Euler Angles**		
Case	**Altitude [m]**	**X [m]**	**Y [m]**	**Y [m/s]**	**Z [m/s]**	**Roll [rad]**	**Pitch [rad]**	**Yaw [rad]**
1	2.50E-05	2.50E-05	9.00E-08	9.37E-06	0.00E+00	4.30E-11	2.30E-16	1.40E-19
2	2.50E-05	2.50E-05	9.00E-08	9.37E-06	0.00E+00	1.70E-04	9.00E-05	4.20E-05
3	2.50E-05	2.50E-05	9.00E-08	9.37E-06	0.00E+00	9.00E-04	1.50E-03	3.80E-04
4	3.20E-03	3.30E-03	1.70E-11	6.00E-13	2.40E-10	6.70E-06	4.70E-06	1.10E-05
5	3.20E-03	3.30E-03	7.00E-06	9.37E-06	6.00E-17	6.70E-06	4.70E-06	1.10E-05
6	3.20E-03	3.30E-03	7.00E-06	9.37E-06	1.80E-15	4.30E-11	2.30E-16	1.40E-19
7	3.20E-03	3.30E-03	1.15E-04	1.00E-05	1.80E-15	2.60E-11	2.30E-16	1.40E-19
8	3.20E-03	3.30E-03	8.30E-04	1.00E-04	1.90E-15	8.80E-11	2.30E-16	1.40E-19
9	1.00E-01	1.00E-01	1.00E-01	5.00E-03	2.90E-14	2.75E-16	2.00E-08	1.75E-15
10	1.28E-01	2.00E-04	1.20E-01	1.06E-05	2.70E-03	4.50E-09	7.00E-06	1.20E-18

6.6.2 Optimization

The architecture of Genesis was designed to enable trajectory optimization. To verify this capability, a Genesis case was created to match a SORT case that flies a guided lunar ascent and uses an optimizer to select the single-axis rotation and PEG targets to maximize the final mass while being subject to several constraints on the final state. Genesis was used in conjunction with the Sparse Nonlinear Optimizer (SNOPT), although other optimizers could be used. The resulting optimized Genesis trajectory is very similar to the SORT case. Figure 6.6-2 shows the time histories of the altitude and boost reference to body pitch angle and the differences between the two cases over time. The full comparison included altitude, latitude, longitude, mass, and boost reference to body yaw, pitch, and roll angles. The final mass of the Genesis case was within 0.1 kg of the final mass in SORT.

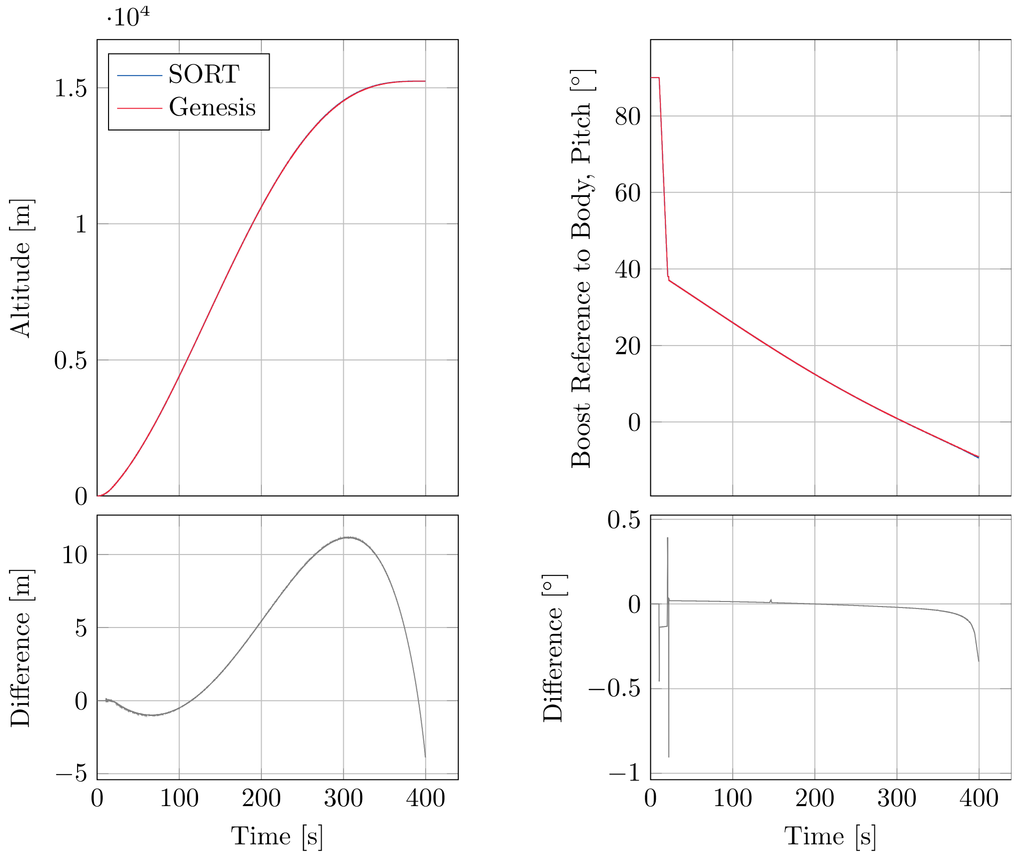


Figure 6.6-2. Time Histories of Altitude and Boost Reference to Body Pitch Angle show that Optimized Genesis Case Matches SORT Case Well

6.6.3 Powered Flight Capability

The equations of motion for Genesis were developed without assuming constant mass properties. A test case was developed to verify the implementation of the equations of motion. The case consists of a spacecraft that produces a time-varying thrust equal to its current weight, so that it hovers. The spacecraft is composed of two point masses, oriented in a vertical configuration. The upper point mass is the dry mass, and the lower point mass is the propellant mass. As the propellant mass is consumed, the center of mass migrates toward the location of the upper point mass. A common simplification in flight simulations with fixed mass properties is to reference the integrated state to the center of mass. If this simplification were done here, then the altitude of the center of mass would remain constant but the locations of the point masses (and any other structural points) would slowly decrease. When mass properties can change, the correct approach is to use a reference point that is fixed with respect to the structure of the spacecraft. Applying this approach, the altitude of the two point masses (or any other structural points) will remain constant, while the altitude of the center of mass slowly increases. The results for the Genesis case are shown in Figure 6.6-3.

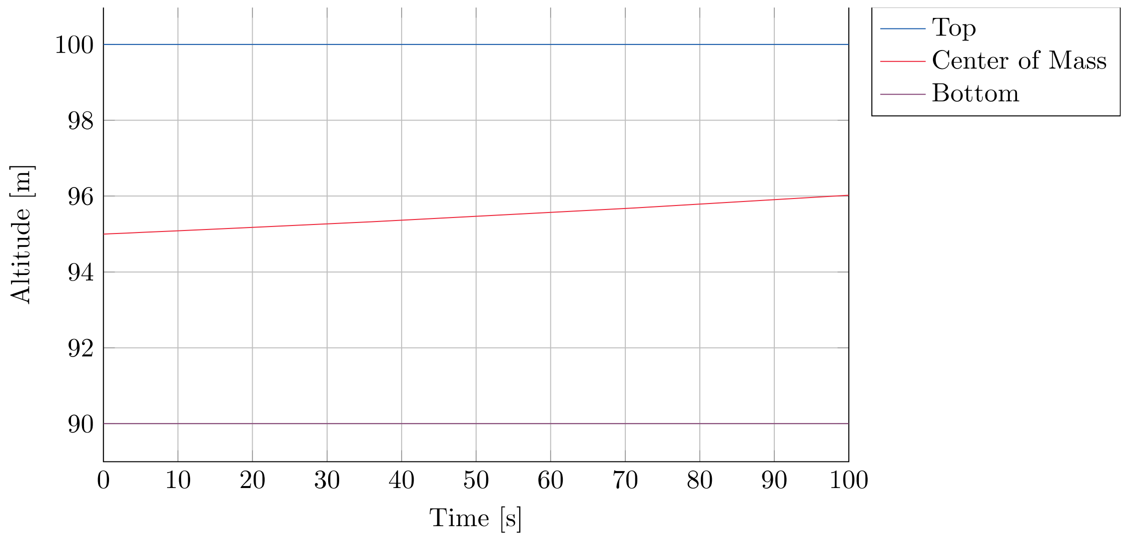


Figure 6.6-3. Test Case of Hovering Spacecraft shows that State in Genesis is Properly Referenced to a Structural Point

6.6.4 Flight Simulation Comparisons

In addition to the NESC check cases and the comparison to an optimized SORT case discussed above, the NESC team compared Genesis with outputs obtained using other flight mechanics simulations for several recent JSC tasks:

- An open-loop aerocapture case at Earth in FAST.
- A Mars entry guidance case using FNPEG in FAST.
- Several lunar powered descent cases using a simple MATLAB trajectory development tool.

In all cases, the time history output from Genesis matched the results of the legacy flight mechanics simulations well. As an example, several outputs of the comparison between Genesis and FAST for an open-loop aerocapture case are shown in Figure 6.6-4. The full comparison considered the time histories of radius, altitude, planet-relative velocity magnitude, dynamic pressure, altitude of apoapsis, and specific energy.

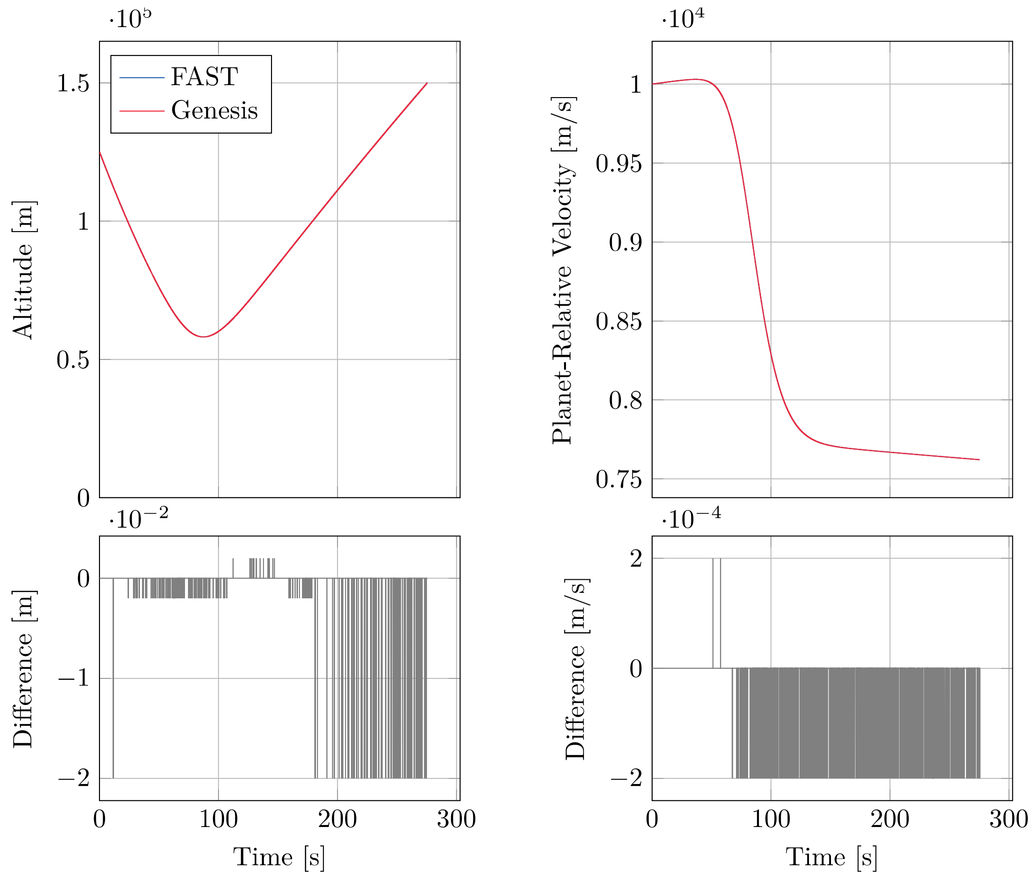
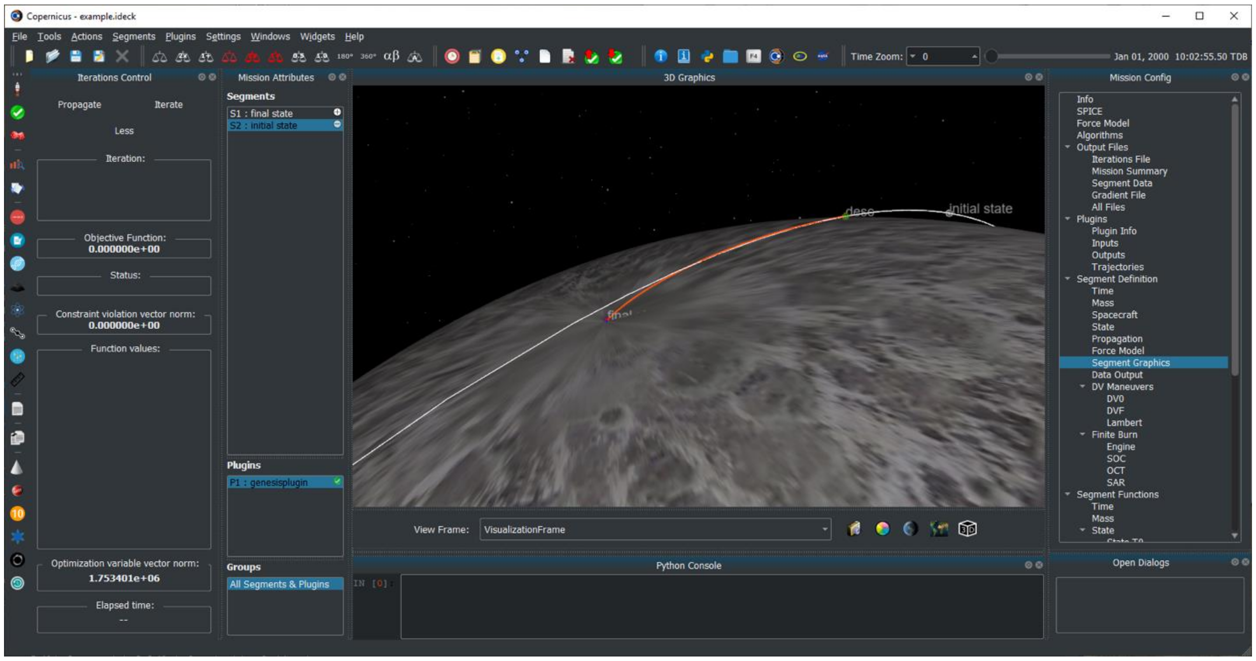


Figure 6.6-4. Comparison of Open-Loop Aerocapture Case in Genesis and FAST

6.6.5 Copernicus Plug-in

Copernicus is widely used by NASA, industry, and academia to study, design, and execute spacecraft missions. A plug-in was developed so that Copernicus is able to call Julia code. This allows Copernicus to, among other things, call Genesis. Copernicus can pass optimization variables to Genesis, and Genesis can return a time history to Copernicus, so that a segment of the overall trajectory is provided by Genesis. By modeling the exo-atmospheric segments of a trajectory in Copernicus and calling Genesis for atmospheric, powered descent, or powered ascent segments, Copernicus is able to perform end-to-end mission optimization.

An example lunar descent case was developed in which Copernicus propagates to the powered descent initiation (PDI) point and then calls Genesis to simulate the trajectory to the ground. Copernicus can adjust the PDI point where the Genesis segment begins in order to perform an end-to-end trajectory optimization. Figure 6.6-5 shows a screenshot of Copernicus, where the red trajectory segment is provided by Genesis.


Copernicus - example.ideck

Objective Function: 0.000000e+00

Constraint violation vector norm: 0.000000e+00

Optimization variable vector norm: 1.753401e+06

Elapsed time: --

Time Zoom: 0

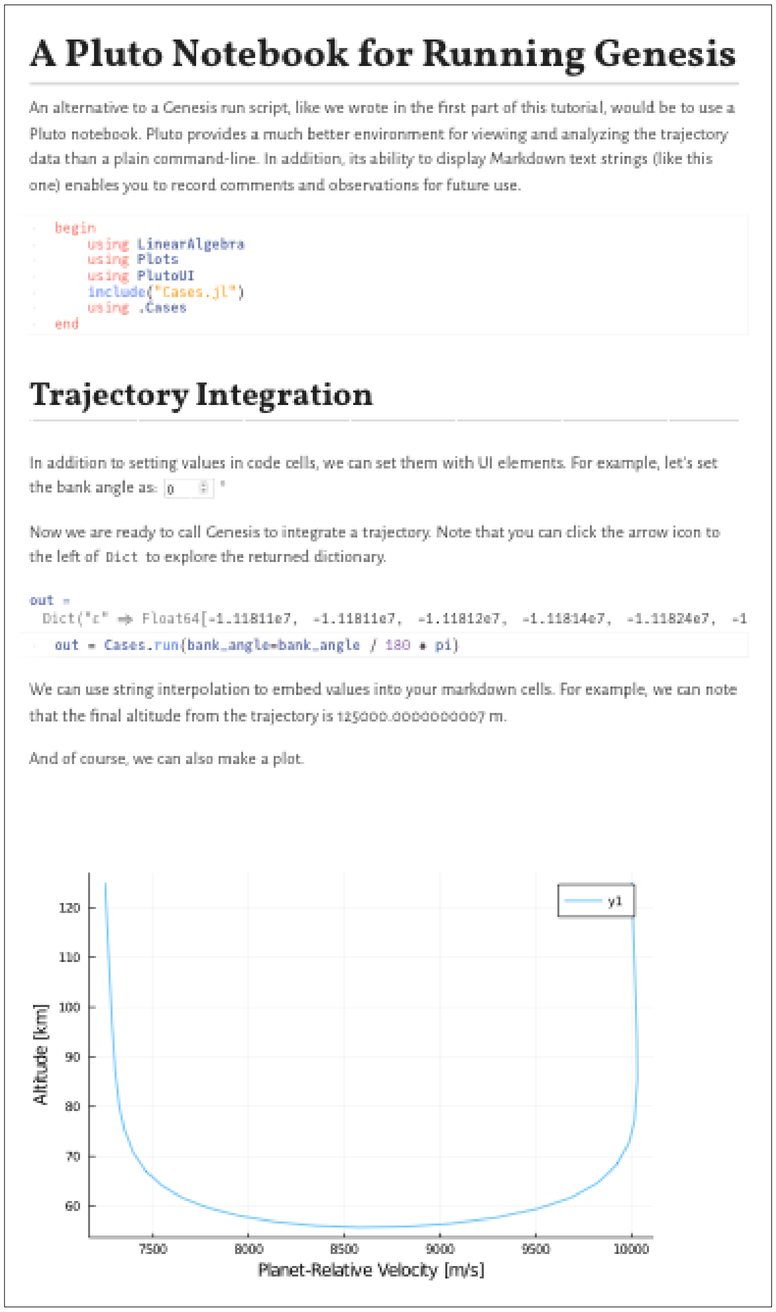
Jan 01, 2000 10:02:55.50 TDB

View Frame: VisualizationFrame

Figure 6.6-5. Example of Copernicus calling Genesis to compute a Segment of Overall Trajectory (red trajectory segment provided by Genesis)

6.6.6 Alternative User Interfaces

While many users choose to call Genesis from a Julia script, other user interfaces (e.g., Jupyter [ref. 9] and Pluto [ref. 10] notebooks) can be leveraged. Figure 6.6-6 shows an example Genesis analysis within a Pluto notebook. With these tools, the user is able to mix expository text, code, and code outputs to create an interactive document. The user may create sliders, text input boxes, and other widgets to create a custom graphical user interface. Whenever code is changed or a widget is adjusted, the dependent computations are repeated so that the notebook is in a consistent state. This type of environment provides a useful way for users to explore the behavior and sensitivities of a trajectory.



A Pluto Notebook for Running Genesis

An alternative to a Genesis run script, like we wrote in the first part of this tutorial, would be to use a Pluto notebook. Pluto provides a much better environment for viewing and analyzing the trajectory data than a plain command-line. In addition, its ability to display Markdown text strings (like this one) enables you to record comments and observations for future use.

```
begin
    using LinearAlgebra
    using Plots
    using PlutoUI
    include("Cases.jl")
    using .Cases
end
```

Trajectory Integration

In addition to setting values in code cells, we can set them with UI elements. For example, let's set the bank angle as: 0 °

Now we are ready to call Genesis to integrate a trajectory. Note that you can click the arrow icon to the left of `Dict` to explore the returned dictionary.

```
out =
 Dict("r" ⇒ Float64[-1.11811e7, -1.11811e7, -1.11812e7, -1.11814e7, -1.11824e7, -1
out = Cases.run(bank_angle=bank_angle / 180 * pi)
```

We can use string interpolation to embed values into your markdown cells. For example, we can note that the final altitude from the trajectory is 125000.000000007 m.

And of course, we can also make a plot.

Figure 6.6-6. Example Pluto Notebook for running Genesis from Genesis Tutorial

6.7 FAST and Genesis Design Comparison

Certain patterns and practices are common in NASA's legacy flight simulations. Sometimes these patterns are beneficial. For example, SORT has an event-driven design that is useful for defining trajectories. However, the patterns sometimes become outdated (e.g., common blocks that are found in many legacy tools). The architecture of FAST was dictated by Trick; however, with Genesis the NESC team had the opportunity to reevaluate the inherited patterns and practices and choose which ones to preserve. This section compares the design of Genesis with FAST and other legacy flight simulations.

6.7.1 Executable versus Library

FAST and other JSC legacy flight simulations are designed as executables. To run a flight simulation, the user runs the executable and specifies an input deck to load. The program writes one or more files containing the results of the simulation. Running a simulation requires spawning an operating system process, and inputs and outputs are communicated through the file system.

However, what happens when a user wants to run a Monte Carlo analysis or a parametric scan? Users often create a custom script that automates the process of manipulating the input deck, spawning the executable, and loading the outputs. In essence, the user is trying to move toward an architecture where running a simulation is a function call.

Genesis is designed as a library. The user writes a Julia script that imports Genesis, configures the case, and then calls a function provided by Genesis to run the simulation. The results of the simulation are returned from the function. This is done in memory, and by default Genesis does not interact with the file system.

This design has a number of benefits. First, as discussed, it tends to be more useful for users because running a parametric sweep is as simple as writing a loop. Second, it composes with other Julia packages (e.g., optimizers). Third, it has better performance in modern HPC environments, where file system performance can become a bottleneck.

6.7.2 Executive

The Trick framework allows the simulation developer to specify functions to be called and fixed rates at which to call them. From this information, Trick generates the simulation executive. By using Trick, FAST was forced to use this style of executive, which limited its capabilities.

In contrast, Genesis is architected more like an ordinary differential equation solver. While it can call functions at fixed rates, that application is the exception rather than the rule. By having a more flexible executive, Genesis is able to support adaptive-time-step integrators, forward and backwards integration, and events that converge on state-based triggers.

6.7.3 Flow of Information

FAST and other JSC legacy flight simulations use an imperative programming paradigm. Computations are viewed as a series of steps (i.e., the executive calls model A before model B because model B is known to depend on some value computed by model A). This is simple to understand but has several disadvantages. First, the dependency between the two models is not reflected locally in the models but lives at a higher level. If the developer mistakenly calls model B before model A, then model B would use stale values. If model B needs the value from model A infrequently, then model A would perform unnecessary computations. Second, this design often relies extensively on mutable data structures (i.e., model A is represented by some structure, and when model A runs, it replaces some fields in its structure with new values). It is difficult to implement multi-threading with this design.

With Genesis, the NESC team has shifted toward a functional programming paradigm, which emphasizes the use of immutable data and pure functions. In Genesis, if model B relies on information computed by model A, then the code for model B will call a function provided by model A. The dependency is reflected locally in model B. The value will not be stale because it is computed on demand. If model B needs the value infrequently, then it only calls the function when needed. When possible, the function is pure, such that model A is not mutated. This approach is easier for the programmer to understand, is more amenable to multi-threading, and is more performant.

6.7.4 Repository

The source code for FAST is stored in a single repository. If another project wants to use a particular model from FAST, then they must make a separate copy of the model. However, this limits collaboration, as neither project will see future improvements made by the other team unless the two versions are synchronized.

In contrast, Genesis promotes collaboration. It is built from a suite of independent Julia packages, with each implementing a particular model. Each package is tracked in its own repository. Another project can reuse one or more of the Julia packages, and multiple users can collaborate on future development of that package. Whereas manually orchestrating dozens of repositories would be difficult, the Julia package manager automatically resolves dependencies to install the appropriate packages.

6.7.5 Run Time

To compare the performance of Genesis and FAST, the NESC team configured both tools for a 3-DOF, open-loop aerocapture trajectory at Earth. For the first comparison, Genesis was configured as similarly to FAST as possible. Both tools used a Runge-Kutta 4 (RK4) integrator, which uses a fixed time step. Both were configured to log time history data at a fixed rate.

The execution time as a function of the fixed time step is shown in Figure 6.7-1. As the time step is increased, each tool takes less time to execute. This relationship is expected to be linear. The dashed lines in Figure 6.7-1 show the linear trend for both tools. Genesis follows the linear trend. However, FAST has a different trend, indicating that there is overhead that becomes a larger portion of the total execution time as the time step is increased. For small time steps, Genesis is about 10 times faster than FAST, but that difference grows to 1000 times faster for larger time steps.

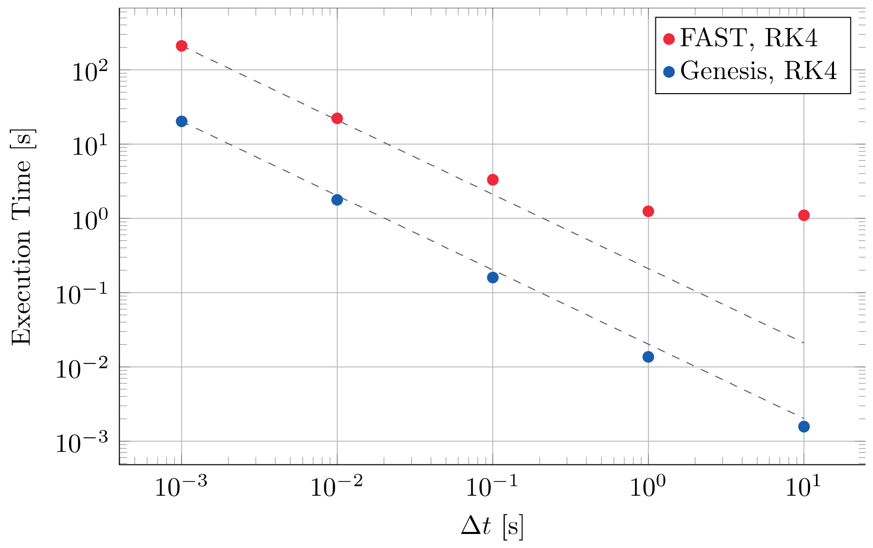


Figure 6.7-1. Comparison of FAST and Genesis, with both Tools configured to use RK4 Integrators

However, Genesis is not restricted to the use of fixed-time-step integrators. Figure 6.7-2 compares Genesis using RK4 and Genesis using the Dormand-Prince 5 (DP5) integrator, which is an adaptive-time-step integrator. The DP5 integrator can often outperform RK4. This depends on the particular trajectory, but Genesis provides the flexibility to use adaptive-time-step integrators when desired.

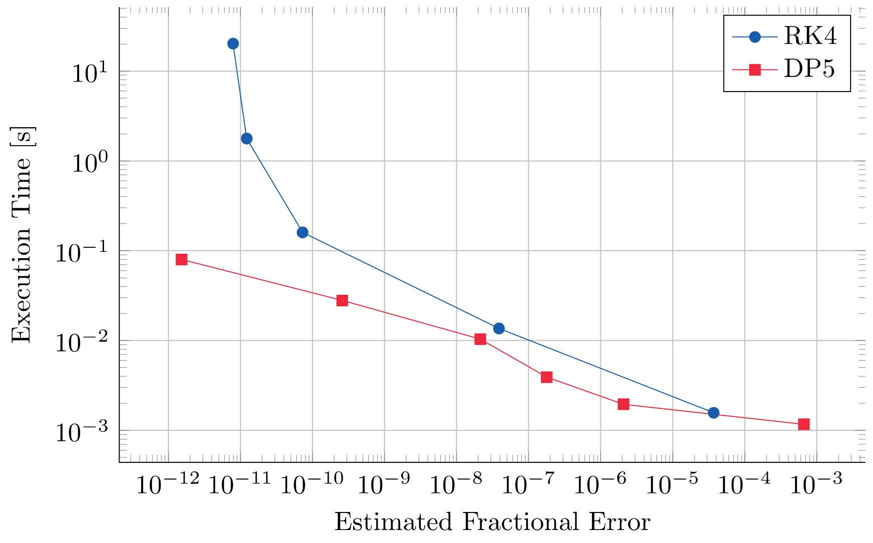


Figure 6.7-2. Comparison of Genesis Trajectories generated with RK4 and DP5 Integrators

6.7.6 Lines of Code

The NESC team used the tokei utility [ref. 11] to count the lines of code in FAST and Genesis. The results for FAST, totaling nearly half a million lines, are shown in Table 6.7-1. The total lines of code for the various packages that make up Genesis is 11,000. While this kind of comparison is crude, the results are striking. Genesis is more capable than FAST but requires only one line of code for every 50 in FAST. This is partly the result of using Julia, which has built-in support for linear algebra that allows mathematical models to be implemented compactly, often in a manner nearly identical to the way they are written by hand.

Furthermore, since FAST's model code is implemented in C++ and its input deck uses Python, a new developer must become proficient in both languages. With Genesis, a developer need only become competent in Julia.

Table 6.7-1. FAST Lines of Code (shown in thousands)

	C++	**C**	**Python**	**Total**
Trick	90	16	11	117
JEOD	249		78	327
FAST	22		7	29
Total	361	16	96	473

6.7.7 Development Effort

While detailed time tracking was not performed, a rough estimate of the development effort can be made. The development of Genesis began in June of 2019, and the milestones discussed above were achieved by the end of calendar year 2020. Contributors to the development of Genesis include half-time support from two civil servants and full-time support from two contractors. The contractors did not have prior experience with Julia, so the first 6 months of development also included learning the language. A rough estimate of the overall development effort is three developers over a 1-year period.

7.0 Summary

The first goal of this assessment was to address near-term updates for FAST in support of the CPAS project and CCP. This was achieved in the first year of this assessment, and FAST continues to be used by the CPAS project and the CCP parachute team, primarily for parachute system reconstruction and analysis. The second goal of the assessment was to consolidate several of JSC's legacy flight simulations tools into FAST by developing several missing features, in particular, optimization. The NESC team discovered that these features were difficult to implement in FAST because of limitations imposed by Trick and JEOD. To overcome these limitations, Genesis was developed in the Julia language as a successor to FAST. The architecture of Genesis was designed to be more amenable to optimization and modern HPC environments. Comparisons between FAST and Genesis show that Genesis is between 10 and 1000 times faster than FAST, with 50 times fewer lines of code. Development of Genesis is ongoing, but it is planned that Genesis will be available from the NASA Software page [ref. 12].

8.0 Findings and Observations

8.1 Findings

The following findings were identified:

F-1. Genesis provides upgraded flight mechanics simulation capabilities for ascent, aerocapture, entry, descent, and landing (A^2EDL) trajectories. It has fewer lines of code and a faster execution speed, and it is easier to maintain than FAST.

F-2. A Genesis plug-in for Copernicus enables end-to-end mission planning.

F-3. Genesis produces outputs consistent with the NESC verification cases and several example analyses in legacy flight mechanics simulations.

8.2 Observations

The following observations were identified:

O-1. Trick and JEOD were developed for real-time flight simulators, which include human-in-the-loop simulators for flight procedure development and hardware-in-the-loop simulators for flight software validation.

O-2. Using tools that are not designed for a particular task imposes costs on productivity and flexibility.

O-3. Engineers increasingly develop software, but they lack formal training on software development practices (e.g., version control, unit testing, continuous integration).

O-4. Julia proved to have several benefits for aerospace analysis tasks, as it is designed for scientific computation and provides competitive run-time performance.

O-5. Functional programming, with its emphasis on immutable data and pure functions, makes dependencies between models clearer.

O-6. In modern HPC environments, software can be limited by the performance of the file system.

O-7. The NESC verification cases are a valuable resource for flight mechanics simulation and trajectory design software development.

9.0 Alternative Viewpoint(s)

There were no alternative viewpoints identified during the course of this assessment by the NESC team or the NRB quorum.

10.0 Other Deliverables

Additional deliverables for this assessment were:

- FAST
  - Source code, under version control on the FSL GitLab instance at https://gitlab-fsl.jsc.nasa.gov/FAST/FAST
  - User guide, hosted on the FSL web server at https://web-fsl.jsc.nasa.gov/userpages/fast/fast/latest/user_manual/
  - NPR 7120.2C Class C documentation, kept under configuration control by the CPAS project [refs. 13–18]
- Genesis
  - Source code, under version control on the FSL GitLab instance at https://gitlab-fsl.jsc.nasa.gov/Genesis/Genesis.jl
  - User guide, hosted on the FSL web server at https://web-fsl.jsc.nasa.gov/userpages/dmatz/Genesis.jl/latest/docs/

11.0 Lessons Learned

No lessons learned were identified as a result of this assessment.

12.0 Recommendations for NASA Standards and Specifications

No recommendations for NASA standards and specifications were identified as a result of this assessment.

13.0 Definition of Terms

Finding	A relevant factual conclusion and/or issue that is within the assessment scope and that the team has rigorously based on data from their independent analyses, tests, inspections, and/or reviews of technical documentation.
Lessons Learned	Knowledge, understanding, or conclusive insight gained by experience that may benefit other current or future NASA programs and projects. The experience may be positive, as in a successful test or mission, or negative, as in a mishap or failure.
Observation	A noteworthy fact, issue, and/or risk, which may not be directly within the assessment scope, but could generate a separate issue or concern if not addressed. Alternatively, an observation can be a positive acknowledgement of a Center/Program/Project/Organization’s operational structure, tools, and/or support provided.
Problem	The subject of the independent technical assessment.
Supporting Narrative	A paragraph, or section, in an NESC final report that provides the detailed explanation of a succinctly worded finding or observation. For example, the logical deduction that led to a finding or observation; descriptions of assumptions, exceptions, clarifications, and boundary conditions.

14.0 Acronyms and Nomenclature List

A^{2}EDL	Ascent, Aerocapture, Entry, Descent, and Landing
CCP	Commercial Crew Program
CI	Continuous Integration
COESA	Committee on Extension to the Standard Atmosphere
CPAS	Capsule Parachute Assembly System
CPU	Central Processing Unit
DOF	Degree of Freedom
DP5	Dormand-Prince 5 (adaptive-time-step integrator)
DQPDG	Dual Quaternion Powered Descent Guidance
DRM	Design Reference Mission
DSS	Decelerator System Simulation
FAST	Flight Analysis and Simulation Tool
FNPAG	Fully Numeric Predictor-corrector Aerocapture Guidance
FNPEG	Fully Numeric Predictor-corrector Entry Guidance
FSL	Flight Sciences Laboratory
GEMASS	General Electric Missile and Satellite Simulation
GN&C	Guidance, Navigation, and Control
GPU	Graphics Processing Unit
GRAM	Global Reference Atmospheric Model
HEOMD	Human Exploration and Operations Mission Directorate
HPC	High-Performance Computing
JEOD	JSC Engineering Orbital Dynamics
JSC	Johnson Space Center

L/D	Lift/Drag, Lift to Drag Ratio
MPCV	Multi-Purpose Crew Vehicle
NED	North East Down
NESC	NASA Engineering and Safety Center
NPR	NASA Procedural Requirement
PDI	Powered Descent Initiation
PEG	Powered Explicit Guidance
POST2	Program to Optimize Simulated Trajectories II
RK4	Runge-Kutta 4 (fixed-time-step integrator)
SNOPT	Sparse Nonlinear Optimizer
SORT	Simulation for the Optimization of Rocket Trajectories
STMD	Space Technology Mission Directorate

15.0 References

1. Jackson, E. B., et al., "Development of Verification Check-Cases for Six Degree-of-Freedom Flight Vehicle Simulations," AIAA Modeling and Simulation Technologies Conference, Boston, MA, AIAA 2013-5071, 2013.
2. Jackson, E. B., "Further Development of Verification Check-Cases for Six Degree-of-Freedom Flight Vehicle Simulations," AIAA 2015-1810, AIAA Modeling and Simulation Technologies Conference, Kissimmee, FL, January 5-9, 2015.
3. Murri, D. G., Jackson, E. B., and Shelton, R. O., "Check-Cases for Verification of 6-Degree-of-Freedom Flight Vehicle Simulations: Vols I & II," NASA TM-2015-218675, January 2015.
4. "JSBSim: An Open Source, Platform-independent, Flight Dynamics and Control Software Library in C++," URL: http://jsbsim.sourceforge.net, last accessed April 1, 2021.
5. Williams, J., Kamath, A. H., Eckman, R. A., Condon, G. L., Mathur, R., and Davis, D. C., "Copernicus 5.0: Latest Advances in JSC's Spacecraft Trajectory Optimization and Design System," AAS/AIAA Astrodynamics Specialist Conference, AAS 19-719, August 2019.
6. Batcha, A., Williams, J., Dawn, T., Gutkowski, J., Widner, M., Smallwood, S., Killeen, B., Williams, E., and Harpold, R., "Artemis 1 Trajectory Design and Optimization," AAS 20-649, 2020 AAS/AIAA Astrodynamics Specialist Conference, August 9–13, 2020.
7. "The Julia Programming Language," URL: https://julialang.org, last accessed April 2, 2021.
8. Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B., "Julia: A Fresh Approach to Numerical Computing," *SIAM Review*, Vol. 59, 2017, pp 65–98.
   DOI: 10.1137/141000671. pdf
9. "Jupyter," URL: https://jupyter.org, last updated February 8, 2021, last accessed April 1, 2021.
10. "Pluto.jl," URL: https://github.com/fonsp/Pluto.jl, last accessed April 1, 2021.
11. "tokei," URL: https://github.com/XAMPPRocky/tokei, last accessed March 1, 2021.
12. "NASA Software," URL: https://software.nasa.gov, NASA Technology Transfer Program.
13. "NASA Software Engineering Requirements," NPR 7120.2C, August 2, 2019.

14. “Software Requirements & Verification Document: Plan and Report for the Capsule Parachute Assembly System (CPAS),” JSC 67343, July 2019.
15. “Capsule Parachute Assembly System (CPAS) Analysis Software Development Plan,” JSC 67213, July 2019.
16. “Capsule Parachute Assembly System (CPAS) Analysis Software User Manual,” JSC 67357, September 2019.
17. “Capsule Parachute Assembly System (CPAS) Analysis Software JPR 7150.2 Compliance Matrix,” JSC 67385, September 2019.
18. “CPAS Analysis Software Development Standards,” JSC 67389, September 2019.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)* 04/27/2021	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Improvements to the Flight Analysis and Simulation Tool (FAST) and Initial Development of the Genesis Flight Mechanics Simulation for Ascent, Aerocapture, Entry, Descent, and Landing (A2EDL) Trajectory Design	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Murri, Daniel G.; Matz, Daniel A.; Hoffman, David A.; Berndt, Jon S.; Brown, Susan C.; Prokop, Lorraine E.	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 869021.01.23.01.01
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** NESC-RP-18-01309
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA/TM-20210014622

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category Space Transportation and Safety
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The NASA Engineering and Safety Center (NESC) was requested to provide significant enhancements to the Flight Analysis and Simulation Tool (FAST), a generic, variable-degree-of-freedom, multi-body ascent, aerocapture, entry, descent, and landing (A2EDL) flight simulation code and a key Agency analysis tool. This report details the upgrades to FAST that were completed, describes a change to the assessment's scope that allowed its goals to be exceeded with a new and innovative approach, and discusses the results of the modified approach.

15. SUBJECT TERMS
NASA Engineering and Safety Center; Flight Analysis and Simulation Tool; Simulation; Flight Mechanics

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	37	(443) 757-5802

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18