AAS 25-576

A HISTORY OF COPERNICUS: THE ORIGIN, DEVELOPMENT, AND EVOLUTION OF JSC'S SPACECRAFT TRAJECTORY DESIGN AND OPTIMIZATION SYSTEM

Jacob Williams[*], **Juan S. Senent**[†], **Ravishankar Mathur**[‡], **Shaun M. Stewart**[§]

This paper describes the history of the Copernicus spacecraft trajectory design and optimization system, a software tool originally developed at The University of Texas at Austin by Dr. Cesar Ocampo, and then later at the NASA Johnson Space Center. For twenty years, Copernicus has been a workhorse tool for advanced mission design at JSC. During this time, the tool has been under constant development and has evolved significantly. The origin, development, and use of Copernicus is detailed, and some lessons learned and future possibilities are also presented.

INTRODUCTION

Advanced spacecraft trajectory design and optimization tools are a critical component for enabling human and science missions at NASA. In an earlier paper,[1] we outlined the history of on-orbit trajectory design for the Orion spacecraft at the NASA Johnson Space Center (JSC), culminating in the Artemis I mission flown in 2022. Copernicus,[2–6] the software tool used to design that trajectory, is a comprehensive and generalized spacecraft trajectory design and optimization system. Developed at The University of Texas at Austin (UT) and JSC, it forms part of the suite of tools used by NASA, industry, and academia to study, design, and execute spacecraft missions.

The Copernicus software application and environment includes a GUI with interactive 3D visualization, as well as scripting, API, and parallelization components. It facilitates the design and optimization of spacecraft trajectory problems using complex force fields, multiple types of propulsion systems (impulsive $\rightarrow$ finite burn $\rightarrow$ low to high thrust), multiple spacecraft, contingency scenarios and multiple reference frames. It is applicable to celestial body centered (planets, moon, asteroids, comets), Earth-Moon, Earth-Moon-Sun (libration points and orbits), interplanetary (asteroids, comets, cyclers), and satellite tours. Copernicus, which was the winner of the 2021 NASA Software of the Year (SOTY) award, is intended to be a tool that adapts and evolves to state-of-the-art approaches and requirements in the field of spacecraft trajectory simulation, design, optimization, and operation. The earlier paper touched on the development of Copernicus during this period, and the current paper provides more detail, telling the story of the history and development of the software from its inception. The unique architecture of Copernicus was the brainchild of its original creator, Dr. Cesar Ocampo (May 1967 – September 2024) and this paper also serves as a tribute to him and the tool he created which has had a profound influence on spacecraft trajectory design and optimization.

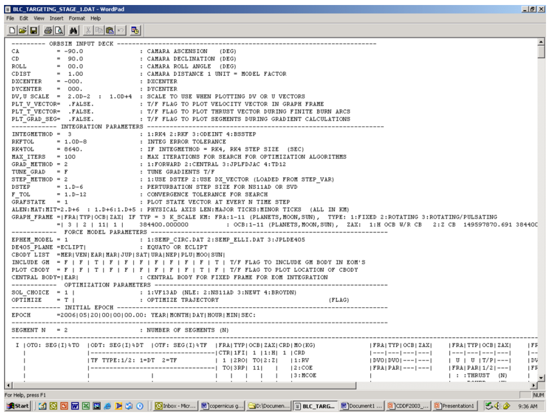
[*]EG/Aeroscience and Flight Mechanics, NASA Johnson Space Center, Houston, TX 77058

[†]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

[‡]Intuitive Machines, Houston, TX, 77058

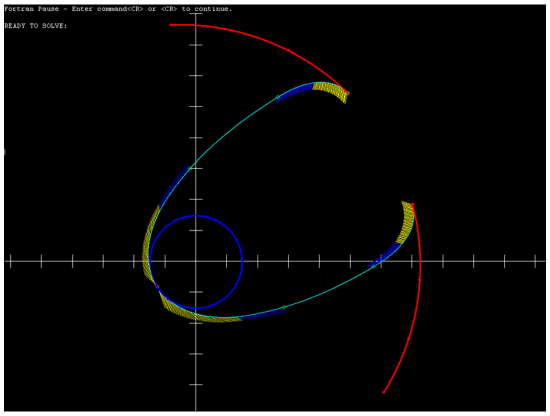
[§]Intuitive Machines, Houston, TX, 77058

ORIGINS OF COPERNICUS (2000 – 2002)



```
BLC_TARGETING_STAGE_1.DAT - WordPad
--------- ORBSIM INPUT DECK -------------------------------------------------------------------
CA          = -90.0               : CAMARA ASCENSION   (DEG)
CD          =  90.0               : CAMARA DECLINATION (DEG)
ROLL        =  00.0               : CAMARA ROLL ANGLE  (DEG)
CDIST       =  1.00               : CAMARA DISTANCE 1 UNIT = MODEL FACTOR
DXCENTER    = -000.               : DXCENTER
DYCENTER    =  000.               : DYCENTER
DV,U SCALE  =  2.0D-2  :  1.0D+4  : SCALE TO USE WHEN PLOTTING DV OR U VECTORS
PLT_V_VECTOR=  .FALSE.            : T/F FLAG TO PLOT VELOCITY VECTOR IN GRAPH FRAME
PLT_T_VECTOR=  .FALSE.            : T/F FLAG TO PLOT THRUST VECTOR DURING FINITE BURN ARCS
PLT_GRAD_SEG=  .FALSE.            : T/F FLAG TO PLOT SEGMENTS DURING GRADIENT CALCULATIONS
------------ INTEGRATION PARAMETERS ------------------------------------------------------------
INTEGMETHOD =  3                  : 1:RK4 2:RKF 3:ODEINT 4:BSSTEP
RKFTOL      = 1.0D-8              : INTEG ERROR TOLERANCE
RK4TOL      = 8640.               : IF INTEGMETHOD = RK4, RK4 STEP SIZE  (SEC)
MAX_ITERS   = 100                 : MAX ITERATIONS FOR SEARCH FOR OPTIMIZATION ALGORITHMS
GRAD_METHOD = 2                   : 1:FORWARD 2:CENTRAL 3:JPLFDJAC 4:TD12
TUNE_GRAD   = F                   : TUNE GRADIENTS T/F
STEP_METHOD = 2                   : 1:USE DSTEP 2:USE DX_VECTOR (LOADED FROM STEP_VAR)
DSTEP       = 1.D-6               : PERTURBATION STEP SIZE FOR NS11AD OR SVD
F_TOL       = 1.D-12              : CONVERGENCE TOLERANCE FOR SEARCH
GRAFSTATE   = 1                   : PLOT STATE VECTOR AT EVERY N TIME STEP
ALEN:MAT:MIT=2.D+6  : 1.D+6:1.D+5 : PHYSICAL AXIS LEN:MAJOR TICKS:MINOR TICKS  (ALL IN KM)
GRAPH_FRAME =|FRA|TYP|OCB|ZAX| IF TYP = 3 K_SCALE KM: FRA:1-11 (PLANETS,MOON,SUN),  TYPE: 1:FIXED 2:ROTATING 3:ROTATING/PULSATING
            =| 3 | 2 | 11| 1 |    384400.000000          : OCB:1-11 (PLANETS,MOON,SUN),  ZAX:  1:H OCB W/R CB   2:Z CB  149597870.691 38440
------------  FORCE MODEL PARAMETERS -----------------------------------------------------------
EPHEM_MODEL = 1                   : 1:SEMP_CIRC.DAT 2:SEMP_ELLI.DAT 3:JPLDE405
DE405_PLANE =ECLIPT|              : EQUATO OR ECLIPT
CBODY LIST  =MER|VEN|EAR|MAR|JUP|SAT|URA|NEP|PLU|MOO|SUN|
INCLUDE GM  = F | F | T | F | F | F | F | F | F | F | T | T/F FLAG TO INCLUDE GM BODY IN EOM'S
PLOT CBODY  = F | F | T | F | F | F | F | F | F | T | F | T/F FLAG TO PLOT LOCATION OF CBODY
CENTRAL BODY=|EAR|                : CENTRAL BODY FOR FIXED FRAME FOR EOM INTEGRATION
------------  OPTIMIZATION PARAMETERS ----------------------------------------------------------
SOL_CHOICE  = 1 |                 : 1:VF13AD (NLE: 2:NS11AD 3:NEWT 4:BROYDN)
OPTIMIZE    = T |                 : OPTIMIZE TRAJECTORY                            (FLAG)
------------- INITIAL EPOCH --------------------------------------------------------------------
EPOCH       =2006|05|20|00|00|00.00: YEAR|MONTH|DAY|HOUR|MIN|SEC:
-----------------------------------------------------------------------------------------------
SEGMENT N   = 2                   : NUMBER OF SEGMENTS (N)
--------------------------------------------------------------------------------------------------------------------------------------
 I |OTO: SEG(I)%TO  |ODT: SEG(I)%DT  |OTF: SEG(I)%TF  |FRA|TYP|OCB|ZAX|CRD|MO(KG)          |FRA|TYP|OCB|ZAX|    |FRA|TYP|OCB|ZAX|    |FR
   |                |---------------------------------|CTR|1FI| 1 |1:H| 1 |CRD             |---|---|---|---|    |---|---|---|---|    |--
   |                |TF TYPE:1/2: 1=DT  2=TF          | 1 |2RO| TO|2:Z|   |1:RV            |DVO|DVO|---|---|    | U | U |T/P|---|    |DV
   |                |---------------------------------| TO|3RP| 11|   |   |2:COE           |FRA|PAR|---|---|    |FRA|PAR|1/2|---|    |FR
   |                |                |                |   |   |   |   |   |3:MCOE          |                    | : :THRUST  (N)     |
For Help, press F1                                                                                                                NUM
```

(a) Original Input File Format of Copernicus. The user would edit this file manually and then execute the program, which would then open another window and display the iterations.



```
Fortran Pause - Enter command<CR> or <CR> to continue.
READY TO SOLVE:
```

(b) Example Trajectory from the Original Copernicus Prototype. The graphics were based on code Cesar had written as an undergraduate student.[7]

Figure 1: Original Copernicus Prototype (pre-GUI, circa 2002). This was a command-line tool with a 2D graphics display of the trajectory. Cesar Ocampo used these early versions of Copernicus in his courses at UT, as well as demonstrations to NASA.

Copernicus[*] and its underlying concept was invented and developed as a prototype by Professor Cesar Ocampo at The University of Texas at Austin beginning in August 2001. Ocampo joined the faculty there in Fall 2000 as an assistant professor in the Department of Aerospace Engineering & Engineering Mechanics. Up to that time he had written many programs to solve specific problems: designing a minimum energy Earth trailing solar orbit for the Spitzer telescope (named SIRTF at the time),[8] codes for computing how to use the Moon's gravity to extend the lifetime of a stranded Geosynchronous Orbit (GEO) satellite via a plane change from a high-inclination Geosynchronous Transfer Orbit (GTO),[9] tools to design low-thrust transfers to an Earth-Sun Distant Retrograde Orbit (DRO)[†],[10] a commercial tool called Xenon Ion Propulsion System Trajectory Optimization Program (XIPSTOP) for Hughes Space and Communications Company (HSC), used to move high-powered, xenon ion propulsion system based geostationary bound communications satellites from GTO to the required geostationary slot,[11] and many others. Ocampo conceived of Copernicus as the sum of all his prior work: a software tool that could solve all trajectory design and optimization problems, one that would be used for decades and never be finished nor replaced when new tools become available, but rather evolve by incorporating new algorithms and approaches over time.

Ocampo developed the initial working prototype of Copernicus single-handedly between late-2001 and early-2002 (see Figure 1). This prototype, written in Fortran 77/90[‡] using Compaq Visual Fortran (CVF) on Windows, was used to demonstrate the functionality and benefit to potential spon-

[*]Although sometimes written as "COPERNICUS" in the early days, it is not an acronym.

[†]The DRO acronym was coined by Ocampo.

[‡]Fortran was chosen for its mature numerical performance, portability, and deep heritage in the scientific computing community – especially for astrodynamics applications. This choice allowed Copernicus to leverage existing libraries and legacy trajectory tools while delivering high performance across platforms.

sors and also as a teaching tool in courses taught at the University of Texas.* This initial prototype could solve interesting problems quickly, but was limited to ten trajectory segments, had no GUI, used a simple text-file† based input (see Figure 1a), a 2D projection graphics model (see Figure 1b),[7] and included basic numerical integrators and root solvers based on Numerical Recipes.[12] A key feature of Copernicus was the use of the VF13AD optimizer from the Harwell Subroutine Library.‡ Ocampo presented this prototype to NASA JSC, and it was immediately recognized by Jerry Condon (a trajectory and mission designer there) as something innovative which had great potential to replace many of the legacy single-purpose tools that were used at that time at the center.

UT AND JSC COLLABORATION (2002 – 2006)

In June 2002 a grant from NASA JSC, which included JSC Center Director Discretionary Funding (CDDF) and internal Aeroscience and Flight Mechanics Division funding, was used to begin production-level development. Support was also received from the NASA Goddard Space Flight Center (GSFC) and NASA's In-Space Propulsion (ISP) Program Low-Thrust Trajectory Tools (LTTT) activity, which was a NASA-wide effort to develop a suite of low-thrust interplanetary trajectory tools.[13] The top level goals for Copernicus, as outlined during the LTTT activity in 2004 were:

- Efficient and intuitive data input and modification.
- Interactive running, testing, editing, and result feedback.
- Real time integrated visualization.
- Automatic bookkeeping and updating of results (results re-populate the GUI fields; update necessary support files).
- All components must be modular (integrators, optimizers, etc.).
- System must be evolutionary with growth capability. *By design the system will never be finished.*
- System should be multi-platform; use libraries and toolsets that have multi-platform capability.

During this period (2002 – 2006) Ocampo lead the small development team from UT, and coordinated with Jerry Condon at JSC who would become the overall NASA lead of the project (and remain so until his retirement in 2025). The initial prototype was expanded and refactored, and development of a GUI and integrated 3D graphics was initiated. From the outset, graphical interaction with the trajectory problem was a key design requirement. The GUI was intended not just as a convenience, but as a central component for intuitive problem setup, while the integrated graphics window provided real-time visualization of the solution. This visual workflow enabled users to quickly understand the structure of the trajectory, assess convergence behavior, and iteratively refine the setup – all without relying solely on textual input or post-processing tools.

*Ocampo also created the "Mars Intercept Game", which was a modified version of Copernicus that accepted keyboard input to change the thrust vector control of a spacecraft, with the goal of intercepting Mars. The game illustrated the basic problem of spacecraft trajectory optimization, and was used by Ocampo in his classes at UT and in other presentations to schools and students.

†The file (which used the `.DAT` extension) initially followed a fixed format, with each row and column serving a specific purpose. Later, it adopted a namelist format and became known as the "input deck" – a term rooted in the historical use of punched card decks for data input.

‡HSL Archive – `https://www.hsl.rl.ac.uk/archive/`

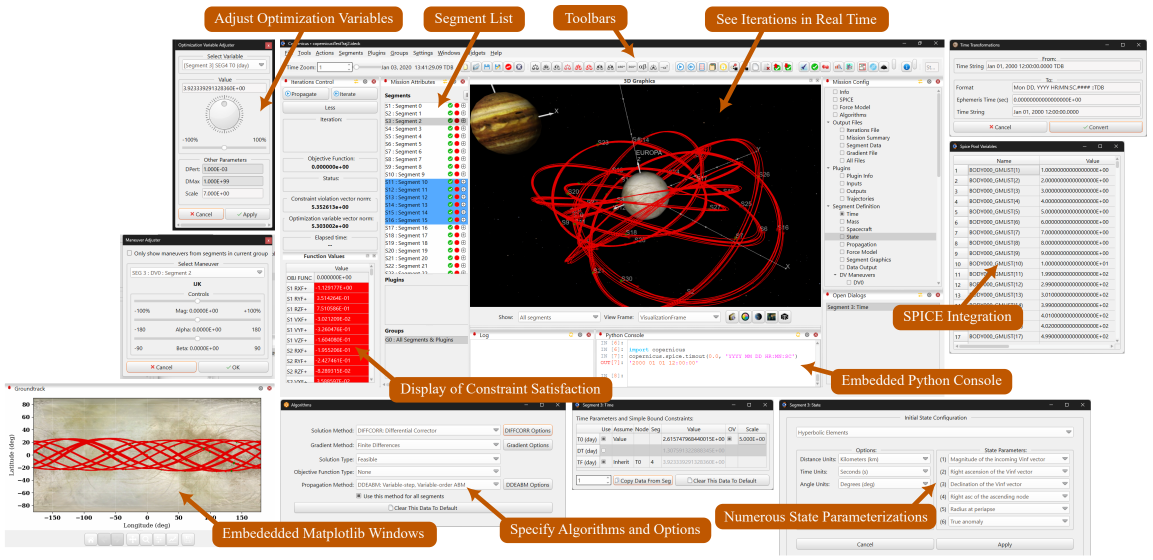

Adjust Optimization Variables
Segment List
Toolbars
See Iterations in Real Time
SPICE Integration
Display of Constraint Satisfaction
Embedded Python Console
Numerous State Parameterizations
Embededded Matplotlib Windows
Specify Algorithms and Options

Figure 2: Copernicus GUI Dialogs. The GUI is a key feature of Copernicus, allowing the user to set up the trajectory design problem, adjust the settings, and receive text and graphical feedback. It has evolved significantly over the years, but remains a core and distinctive component of the solution process.

Having a user-friendly GUI (see Figure 2) was one of Ocampo's top priorities, and indeed was one of the things that separated Copernicus from previous codes he had written and other NASA/JSC trajectory tools at the time. The GUI was always envisioned as the main interface to Copernicus, and a lot of effort was devoted to its development and integration as the original prototype was evolved into the first production release. Ocampo selected the Winteracter GUI toolkit* to build the Copernicus GUI for three main reasons: 1) it was a full-featured GUI toolkit with 3D graphics capabilities available for Fortran-based applications, 2) it supported Windows, Linux, and macOS, and 3) the built-in grid widget, which fit nicely with the concept that each variable would have many attributes that the user could edit (e.g., optimization variable settings such as scale factors and bounds). See Figure 6 for a comparison of various trajectory tool GUIs. The three phases† in the development of the original Copernicus GUI architecture at UT are shown in Table 1. The first version that included a working GUI was completed in August 2004 (see Figure 3).

The original prototype included trajectory graphics (See Figure 1b) in a separate stand-alone window that would launch when the program executed. While often referred to as "2D graphics" (it was called that up to the 4.6 release when it was removed), the original prototype rendered 3D trajectories using a basic hidden-line removal algorithm. The graphics code was partially based on code Ocampo had written as an undergraduate student,[7] converted to use the CVF native graphics API. Although the graphics were limited, they did display optimization iterations in real time as

*Winteracter was a graphics and GUI toolset for Fortran 90 or later. A commercial product, it was sold from 1997–2024, when it was coincidentally discontinued immediately after Cesar's passing in September 2024. GINOMENU, another commercial product for Fortran GUIs, was also considered at the time but not ultimately selected.

†The 4th phase of the Copernicus architecture would be the 5.0 refactor at NASA,[6] where Copernicus was rearchitected into a separate core library that is called from a Python/Qt GUI (rewritten from scratch, mirroring in many ways the original Winteracter GUI, but much improved).

Table 1: The three phases (from 2001 – 2006) in the development of the original Copernicus architecture

Phase 1	**No GUI.** Ocampo's original version written at UT. This was a command line executable that read an input file. There was no GUI, although it did use the CVF graphics routines which opened in a separate window to show the optimization problem iterations. This version was entirely written by Ocampo (see Figure 1).
Phase 2	**Prototype GUI.** An initial prototype GUI that was a separate program that existed only to create the input file that the command line Copernicus would read. It would launch the core executable to iterate.
Phase 3	**Unified GUI.** Only one program that contained the GUI and core code. The core code was called directly from the GUI. This version had integrated graphics (originally just 2D graphics, which were converted to Winteracter graphics calls) and ultimately OpenGL 3D graphics.[14] This was the state of the program in the 1.0 release in 2006, which was delivered to NASA, and it remained this way for many years afterwards.

they occurred – a feature uncommon in many tools at the time. Ocampo's vision was to provide immediate visual feedback throughout the process: from the initial guess, to each optimization step, to the final converged solution. This allowed users to quickly assess whether the solution was converging in the right direction or becoming stuck, without relying on external post-processing tools. Users could interrupt the process at any point, adjust settings such as algorithm parameters or segment configurations, and rerun the problem – enabling an interactive and iterative workflow that significantly improved usability and insight.

While Ocampo's 2D graphics were novel, he recognized several limitations. They were not extensible to new visualization components (e.g., spacecraft, general celestial bodies, etc.), they significantly slowed down the trajectory computations, they did not provide end-user interactivity, and they did not leverage the computer's graphics hardware. Having always been an avid champion of advanced 3D graphics, in 2003 Ocampo's team began design, development, and integration of modern 3D graphics. The result was a fully standalone 3D graphics API, called OpenFrames[*], that allowed any aerospace software to easily integrate high-performance interactive 3D visualizations via the industry-standard OpenGL and OpenSceneGraph[†] graphics interfaces.[14] OpenFrames provided the ability to display celestial bodies, trajectories, spacecraft models, and maneuver vectors. It enabled interactive use (pan, zoom, rotate, animate) and utilized multithreading to leverage the burgeoning trend of multi-core computers to eliminate the slowdown on Copernicus' propagators and optimizers. While Copernicus was the first software to utilize OpenFrames for its graphics, Ocampo's eager support of its development led to it eventually being incorporated as the primary visualization tool for various commercial and NASA tools, including the General Mission Analysis Tool (GMAT) at GSFC.

At the core of Ocampo's architecture was the *segment*[2] – the fundamental building block of all

[*]OpenFrames: Realtime interactive scientific visualization API. `https://github.com/ravidavi/OpenFrames`

[†]OpenSceneGraph is an open source high performance 3D graphics toolkit. `https://openscenegraph.github.io/openscenegraph.io/`

trajectories in Copernicus[*] (see Figure 4). A segment represents an arc connecting two endpoint nodes, t_0 and t_f, each of which may include velocity (Δv) and mass (Δm) discontinuities. The arc can be ballistic or include a finite-burn model. Segments are highly flexible: they can be connected or disconnected in time, state, or mass; inherit properties from other segments; and encapsulate optimization variables, constraints, and objective functions for use by nonlinear solvers or optimizers. They can be propagated both forward ($\Delta t > 0$) and backwards ($\Delta t < 0$), including with finite burns (where a backwards-propagated finite burn segment gains mass from $t_0 \to t_f$). Segments can model individual or multiple spacecraft and can be combined to construct custom transcriptions tailored to specific mission needs. This modular and flexible design enables Copernicus to handle a wide range of trajectory types, including impulsive, low-thrust, and high-thrust transfers, in both planetary and interplanetary, single-body or multi-body contexts. The segment framework also supports the simultaneous optimization of trajectories across multiple alternative scenarios. For example, a nominal trajectory can be optimized alongside several abort scenarios within a single unified formulation. The cost function can be customized to reflect complex mission objectives, such as minimizing the worst-case outcome across all scenarios, enabling robust and resilient trajectory design. Importantly, segments do not need to represent an actual spacecraft trajectory. They can also be used as computational constructs – for example, to evaluate a function or constraint that must be satisfied by another segment containing the physical trajectory. This abstraction adds further flexibility to the architecture, allowing complex dependencies and auxiliary calculations to be integrated seamlessly into the optimization problem.

In its original implementation by Ocampo, Copernicus used the JPL planetary ephemeris to model the positions of planets and satellites. However, it was quickly recognized that integrating the SPICE toolkit – a companion library developed by JPL[†] – would greatly enhance Copernicus's capabilities. SPICE provided robust support for incorporating new reference frames, converting trajectory states between different frames and parameterizations, handling multiple time systems and conversions, and generating trajectory files in SPK format. These features significantly expanded the tool's flexibility and interoperability with other mission design and analysis systems, and SPICE remains a key component of Copernicus up to the present day. Alongside these architectural improvements, Copernicus expanded its modeling capabilities to support high-fidelity, high-order gravity models, enabling accurate simulation of orbits in complex gravitational environments such as low lunar or low Earth orbit.

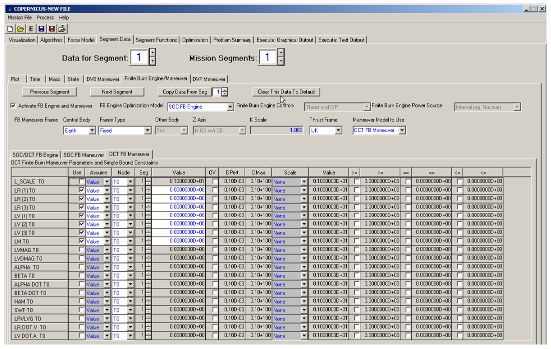
During the initial development of Copernicus, the U.S. human spaceflight program would be profoundly altered by the Space Shuttle Columbia accident in 2003 and the subsequent announcement of the Vision for Space Exploration (VSE) in 2004. Copernicus would be available at just the right time to be a major contributor to planning the next phase of human spaceflight at NASA, with its flexible architecture being exactly what was needed to design and study the required Earth-Moon trajectories for the Constellation Program and follow-on activities to return humans to the Moon and beyond.[1]

Copernicus 1.0 (2006)

The first operational version of Copernicus was released to NASA in March 2006 (v1.0) at the conclusion of the LTTT activity (see Figure 5). This release included a full working GUI with

[*]In the 4.1 release in 2015, *plugins* were added as an additional basic building block. Plugins are companions of segments, and used to introduce user-defined code or other tools into the problem.[15]

[†]The SPICE Toolkit: `https://naif.jpl.nasa.gov/naif/toolkit.html`


(a) At this point, the GUI was taking shape, which each dialog continued within a tabbed layout. The grids are almost fully realized here, a key part of Copernicus, and the main reason for the selection of Winteracter as the GUI toolkit.

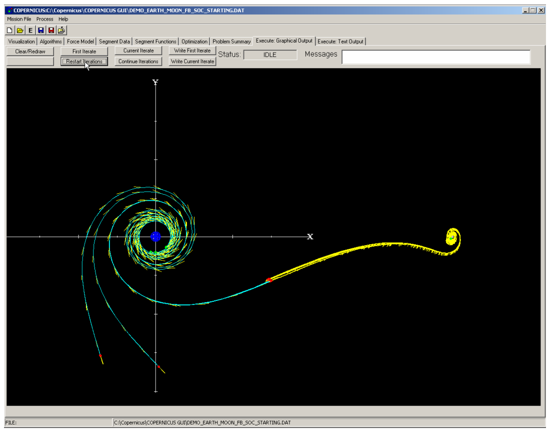

(b) The Graphics dialog was where the user would iterate the trajectory and watch the iterations as they progressed. Notably missing were the 3D graphics (Ocampo's original 2D graphics are the only available option).

Figure 3: Pre-1.0 Version of the Copernicus GUI (demoed at the 2004 Low Thrust TIM). Many of the features present in the V1.0 GUI of 2006 were already in place at this point.

integrated 3D OpenFrames-based graphics,[14] finite burn and impulsive maneuvers, SPICE integration, and various propagation methods and solvers. While the tool has evolved and become much more advanced over time, all of the fundamental concepts (and some of the original code) from this original version remain in the very latest release.

JSC DEVELOPMENT (2007 – PRESENT)

Starting circa March 2007, primary development and maintenance of Copernicus was transferred to the Flight Mechanics and Trajectory Design Branch of JSC.[*] Copernicus has been in continuous use at JSC (and other NASA centers and commercial partners) since that time,[1] and many key capabilities in the tool as it exists today were developed during this period. Ocampo's collaboration continued, first from UT, and then later as a contractor at JSC, serving as an advisor and chief "power user". During this time, Copernicus was brought up to production level, with formal releases and documentation. As the tool began to be used more, new features were added to support current studies (some were natural extensions of the original architecture, and others were new ideas not originally envisioned). Refactoring and modernization of the codebase also continued at JSC.

A distinctive aspect of Copernicus's early development was that, for the most part, the developers were also its primary users. There was little separation between development and user teams – the tool was actively used by its creators to perform mission studies as it evolved. As a result, usability was a central focus and a major driver of development. New capabilities and features were identified organically through day-to-day use and were often implemented immediately, without a formalized development process (although this would be added later). This close integration between usage and

[*]By this point, the team consisted of Ocampo and two graduate students at UT, and Jerry Condon and two developers/analysts at JSC.

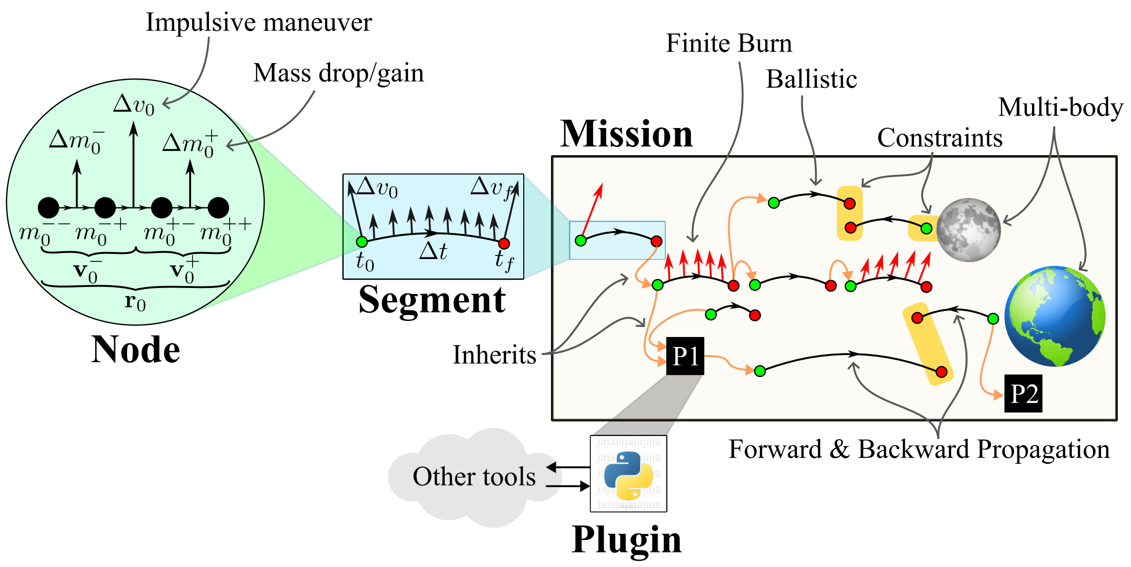
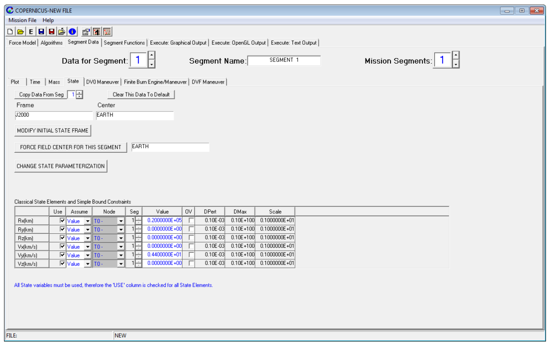


Figure 4: Copernicus Architecture. Each segment allows for velocity (Δv) and mass (Δm) discontinuities at each time node (t_0 and t_f). Input variables can be optimization variables, or inherited from other segments. The segment arc can be numerically integrated or propagated (ballistically or using a finite burn engine). Constraints and the objective functions can be specified. Segments and plugins can be combined to create a mission of almost any level of complexity. This is a key part of the philosophy of Copernicus as a modular and flexible tool where the user is a part of the solution process and has the ability to use creativity to set up the problem exactly the way they want.



Data for Segment: 1   Segment Name: SEGMENT 1   Mission Segments: 1

Copy Data From Seg 1

Frame: J2000   Center: EARTH

FORCE FIELD CENTER FOR THIS SEGMENT: EARTH

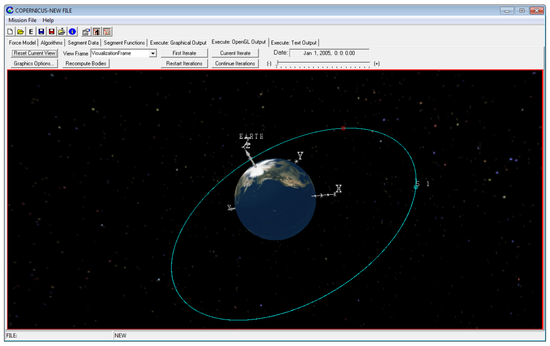
Classical State Elements and Simple Bound Constraints

	Use	Assume	Node	Seg	Value	OV	DPert	DMax	Scale
Rx(km)	✓	Value	T0 -	1	0.2000000E+05		0.10E-03	0.10E+100	0.1000000E+01
Ry(km)	✓	Value	T0 -	1	0.0000000E+00		0.10E-03	0.10E+100	0.1000000E+01
Rz(km)	✓	Value	T0 -	1	0.0000000E+00		0.10E-03	0.10E+100	0.1000000E+01
Vx(km/s)	✓	Value	T0 -	1	0.0000000E+00		0.10E-03	0.10E+100	0.1000000E+01
Vy(km/s)	✓	Value	T0 -	1	0.4400000E+01		0.10E-03	0.10E+100	0.1000000E+01
Vz(km/s)	✓	Value	T0 -	1	0.0000000E+00		0.10E-03	0.10E+100	0.1000000E+01

All State variables must be used, therefore the 'USE' column is checked for all State Elements.

FILE: NEW

(a) In the first release, the entire GUI was contained in one window with many tabs. The variable grids were a key feature that continue to the current release.



(b) The integrated 3D graphics was a key feature of Copernicus, enabling a user to see in real time the progress of the optimization iterations.

Figure 5: Copernicus 1.0 (March 2006). The first release of Copernicus included an integrated GUI and 3D OpenGL interactive graphics visualization, and ran only on Windows. A unique and innovative tool at the time, it provided NASA a freely-available, user-friendly, general-purpose, advanced spacecraft trajectory optimization capability within a sophisticated interactive GUI. This version was used at JSC for early studies for the Orion Project.

development fostered rapid iteration and ensured that the tool remained practical and responsive to immediate needs.

Another characteristic of Copernicus's development was its experimental and adaptive nature. Many features were added to support specific studies or one-off mission needs, but later removed when they proved too specialized or lacked broader applicability. This trial-and-error approach allowed the tool to remain focused, avoiding unnecessary complexity while still enabling rapid prototyping. It reflected the pragmatic philosophy behind Copernicus: to prioritize usability and flexibility for immediate mission needs, while evolving the tool based on actual usage rather than abstract requirements.

A major usability improvement implemented during this period was the ability to insert, delete, and move segments directly within the GUI. This was complemented by automatic detection and resolution of the correct propagation order for interconnected segments, eliminating the need for users to manually specify execution sequences. In the original prototype, segments were propagated in simple numerical order – a method that could produce incorrect results when segments inherited information (e.g., state or mass) from one another. Cesar Ocampo's original paper[2] emphasized the importance of propagating segments in a logically consistent order, but this feature was not implemented until after the 1.0 release. Another key enhancement was the ability to detect circular dependencies between segments – an issue that went undetected in earlier versions and could lead to invalid or misleading solutions. The eventual introduction of a dependency graph to manage segment relationships and identify such cycles significantly improved both reliability and transparency of the propagation logic. See Figure 7 for a representation of the segments and plugins in a reasonably-complex Copernicus mission.

As the overall use cases of Copernicus evolved and expanded over time, the tool was kept up-to-date, as it was being used for a wide range of analysis at JSC and other organizations. Other notable improvements and new features added to Copernicus over these years include:

- Allowing for unlimited number of trajectory segments, optimization variables, and constraints. The original prototype had various hard-coded limits to these parameters.
- GUI for working with SPICE kernels (loading and creating kernels, viewing data, etc.)
- Selectable gravity models, integrators, and force models for each segment. This was not part of the original prototype, but fit naturally into the architecture.
- The addition of other types of propagation methods for segments (e.g., Kepler, Encke, and modified equinoctial elements).
- The addition of many new state parameterizations, including a halo orbit parameterization.
- The plugin architecture,[15] introduced in response to frequent user requests for greater flexibility. While the GUI made it easy to construct and solve complex trajectory optimization problems, users often needed to implement mission-specific features that were too specialized to be incorporated directly into the core Copernicus system. The plugin framework provided a solution by allowing user-defined code to be injected into the optimization process. This enabled analysts to incorporate custom models, constraints, or objective functions tailored to their unique mission scenarios, without altering the main codebase. Additionally, the plugin interface allowed Copernicus to interact with external tools, making it a more extensible and interoperable platform.
- Various widgets and tools such as the Δv to finite burn wizard, the optimization variable and maneuver adjusters (for modifying variables and seeing in real time how they affect the

trajectory), and the randomize optimization variables widget (which allows for perturbing all optimization variables by some specified percentage).[*]
- The addition of cross-platform functionality. Originally, Copernicus ran only on Windows. One of the earliest projects after the transition to JSC was to port the code from the then-discontinued Windows CVF compiler to the cross-platform Intel Fortran Compiler.[†] This meant that Linux and Mac releases would be possible, but it was several years before those were realized (a command-line only Linux build in 2009 and an X11-based Mac GUI prototype in 2018).[‡]
- General speed and stability improvements, and code refactoring as modern Fortran features became available in newer compilers.[18]
- Various visualization updates including celestial body shadowing, trajectory intermediate point markers, and stability improvements.
- More flexible data output. For example: being able to specify which variables to output (type, frame, parameterization, etc.) and the output file format (CSV, JSON, HDF5, or SPK).

A key to the success of Copernicus was regular releases through the NASA Technology Transfer Program.[§] This was a somewhat novel approach for mission design tools at JSC at the time, which historically had mostly consisted of internal tools not widely distributed, or proprietary tools built and controlled by specific contractors with no motivation to freely distribute them. Providing Copernicus freely to all NASA centers, other government entities, and government contractors, meant a growing pool of users on a variety of different projects. This not only meant less "reinventing the wheel" for trajectory design tools, but also that funds could be more readily available for support and upgrades since it was being widely used on projects that had specific needs. Some major upgrades to Copernicus would ultimately have their origin and funding source from outside of JSC.

Command Line and Scripting

As Copernicus matured and began to be used in production, it became increasingly clear that mission design workflows demanded more than interactive GUI-based problem setup. Trade studies, by their nature, require the evaluation of many configurations, scenarios, and parameter sweeps. Performing these studies manually – by editing input decks and rerunning optimizations one at a time – was impractical, particularly as the number of combinations grew exponentially with problem complexity. To address this challenge, Copernicus began evolving toward script-driven automation.

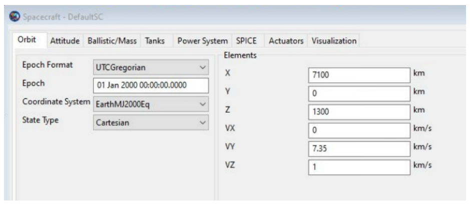
The first major step was the release of a Linux-compatible, command line only build of Copernicus, which enabled the tool to be executed on high-performance computing clusters. This version made it possible to launch large batches of runs in parallel, a critical capability for trade studies at NASA centers like JSC. Another key milestone was the redesign of the input deck, since the

[*]This feature has been used to develop a monotonic basin hopping wrapper to Copernicus.[16]

[†]CVF was the original Fortran compiler used for Copernicus. CVF was the successor to the DEC Visual Fortran compiler, after Compaq purchased DEC in 1999. In Aug. 2001, Compaq sold their Fortran business to Intel. In Dec. 2003, Intel released the successor product: Intel Visual Fortran 8.[17]

[‡]A fully feature-complete cross-platform Copernicus would not exist until the 5.0 release in 2020 with a rewrite of the GUI in Python.

[§]Copernicus Trajectory Design and Optimization System (Version 5.x) (MSC-26673-1). Copernicus is released under a "government use" license, and is freely available to any U.S. government entity or government contractor. Request here: `https://software.nasa.gov/software/MSC-26673-1`



Spacecraft - DefaultSC

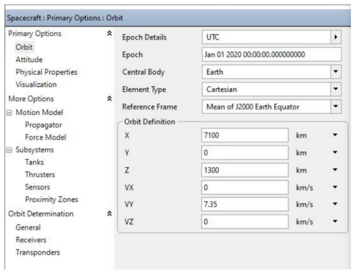
Orbit | Attitude | Ballistic/Mass | Tanks | Power System | SPICE | Actuators | Visualization

Epoch Format	UTCGregorian
Epoch	01 Jan 2000 00:00:00.0000
Coordinate System	EarthMJ2000Eq
State Type	Cartesian

Elements

X	7100	km
Y	0	km
Z	1300	km
VX	0	km/s
VY	7.35	km/s
VZ	1	km/s

(a) GMAT (an open-source tool for space mission design and navigation, developed at GSFC).



Spacecraft : Primary Options : Orbit

Primary Options
- Orbit
- Attitude
- Physical Properties
- Visualization

More Options
- Motion Model
  - Propagator
  - Force Model
- Subsystems
  - Tanks
  - Thrusters
  - Sensors
  - Proximity Zones

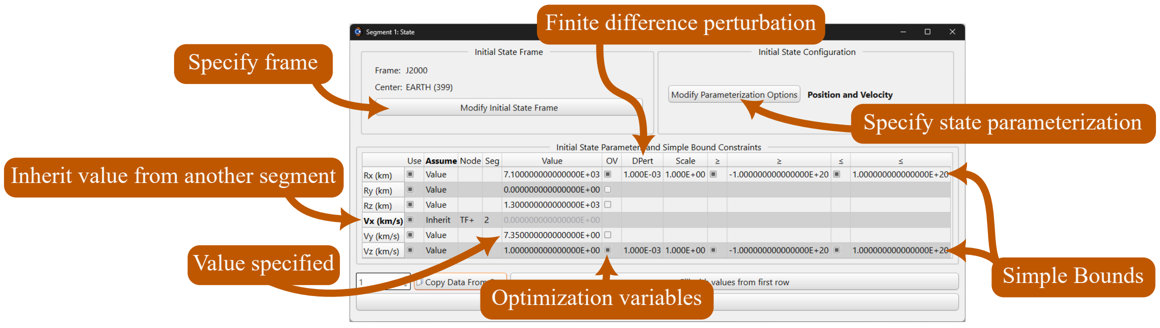
Orbit Determination
- General
- Receivers
- Transponders

Epoch Details	UTC
Epoch	Jan 01 2020 00:00:00.000000000
Central Body	Earth
Element Type	Cartesian
Reference Frame	Mean of J2000 Earth Equator

Orbit Definition

X	7100	km
Y	0	km
Z	1300	km
VX	0	km/s
VY	7.35	km/s
VZ	0	km/s

(b) FreeFlyer (a commercial astrodynamics software package).


Finite difference perturbation
Specify frame
Specify state parameterization
Inherit value from another segment
Simple Bounds
Value specified
Optimization variables
Segment 1: State
Modify Initial State Frame
Modify Parameterization Options
Copy Data From
values from first row

Initial State Frame

Frame: J2000

Center: EARTH (399)

Initial State Configuration

Position and Velocity

Initial State Parameters and Simple Bound Constraints

	Use	Assume	Node	Seg	Value	OV	DPert	Scale	≥	≥	≤	≤
Rx (km)	■	Value			7.100000000000000E+03	■	1.000E-03	1.000E+00	■	-1.000000000000000E+20	■	1.000000000000000E+20
Ry (km)	■	Value			0.000000000000000E+00	☐						
Rz (km)	■	Value			1.300000000000000E+03	☐						
Vx (km/s)	■	Inherit	TF+	2	0.000000000000000E+00							
Vy (km/s)	■	Value			7.350000000000000E+00	☐						
Vz (km/s)	■	Value			1.000000000000000E+00	■	1.000E-03	1.000E+00	■	-1.000000000000000E+20	■	1.000000000000000E+20

(c) Copernicus.

Figure 6: Comparison of Trajectory Tool State Dialogs (GMAT, FreeFlyer, and Copernicus). Copernicus uses a unique architecture where the trajectory segment is the fundamental mission unit, rather than a spacecraft or event sequence. Differing from the others, the distinctive grid interface in Copernicus allows for easily specifying variable attributes (such as optimization scales and bounds all in the same place in the GUI). No other tool is designed quite the same way (with both optimization and GUI interaction at the forefront of development).

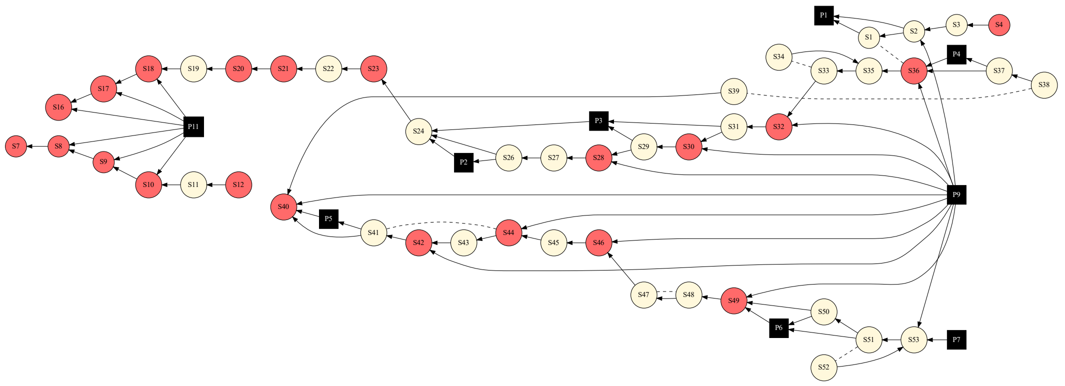
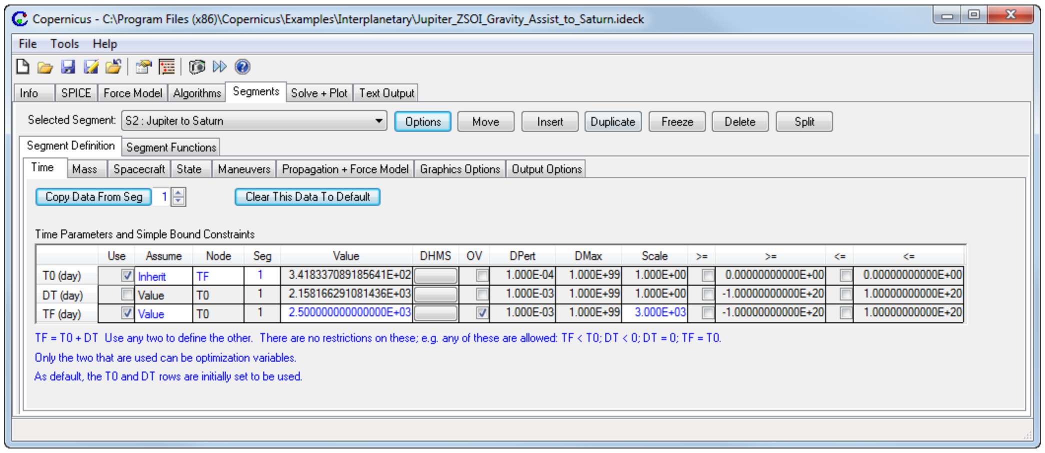


Figure 7: Mission Attribute Dependencies. In Copernicus, the user defines segments (and plugins) and the program automatically determines the order in which they are evaluated, based on the dependencies (e.g., segments can inherit data from other segments). In these diagrams, an arrow indicates a dependency, a dashed line indicates a constraint; circles are segments (red indicates a finite burn) and squares are plugins.


Copernicus - C:\Program Files (x86)\Copernicus\Examples\Interplanetary\Jupiter_ZSOI_Gravity_Assist_to_Saturn.ideck
File Tools Help
Info SPICE Force Model Algorithms Segments Solve + Plot Text Output
Selected Segment: S2 : Jupiter to Saturn
Options Move Insert Duplicate Freeze Delete Split
Segment Definition Segment Functions
Time Mass Spacecraft State Maneuvers Propagation + Force Model Graphics Options Output Options
Copy Data From Seg 1 Clear This Data To Default

Time Parameters and Simple Bound Constraints

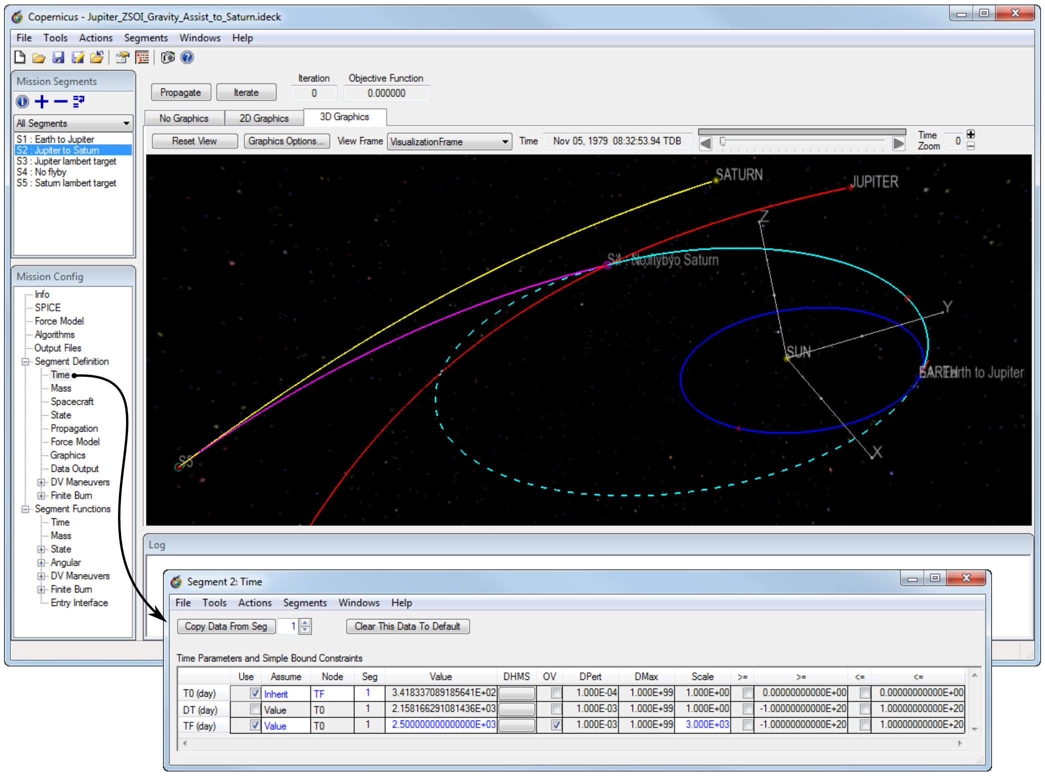
	Use	Assume	Node	Seg	Value	DHMS	OV	DPert	DMax	Scale	>=	>=	<=	<=
T0 (day)	✓	Inherit	TF	1	3.418337089185641E+02		☐	1.000E-04	1.000E+99	1.000E+00	☐	0.00000000000E+00	☐	0.00000000000E+00
DT (day)	☐	Value	T0	1	2.158166291081436E+03		☐	1.000E-03	1.000E+99	1.000E+00	☐	-1.00000000000E+20	☐	1.00000000000E+20
TF (day)	✓	Value	T0	1	2.500000000000000E+03		✓	1.000E-03	1.000E+99	3.000E+03	☐	-1.00000000000E+20	☐	1.00000000000E+20

TF = T0 + DT Use any two to define the other. There are no restrictions on these; e.g. any of these are allowed: TF < T0; DT < 0; DT = 0; TF = T0.

Only the two that are used can be optimization variables.

As default, the T0 and DT rows are initially set to be used.

(a) Copernicus GUI (v3.1). This was the last iteration of the deeply-nested tabbed GUI.

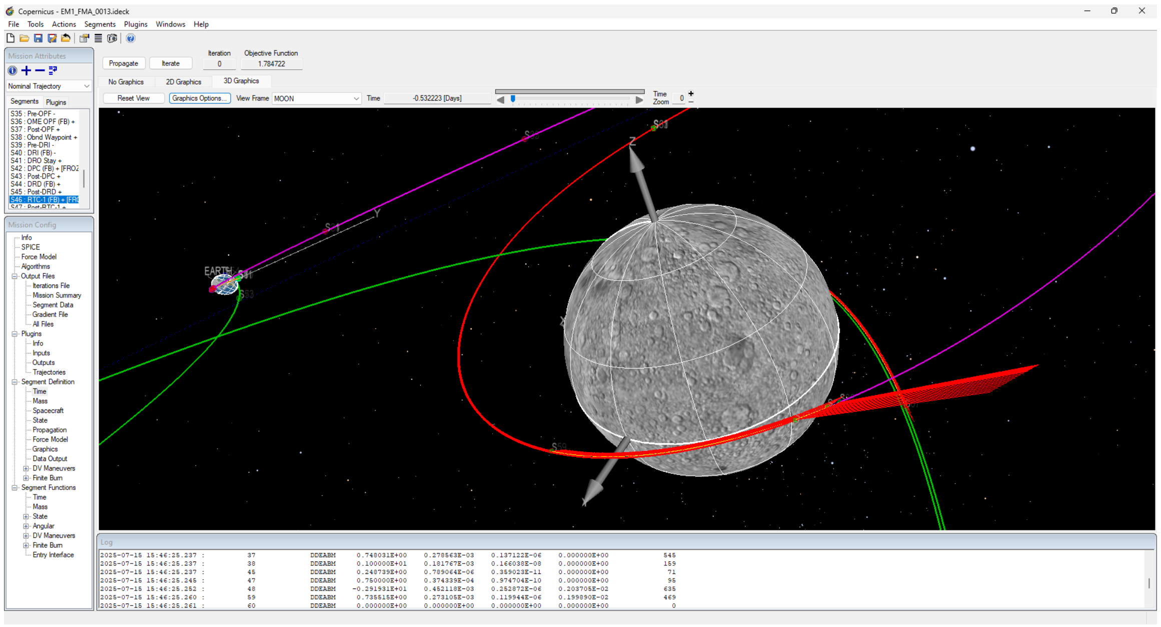


Time Parameters and Simple Bound Constraints

	Use	Assume	Node	Seg	Value	DHMS	OV	DPert	DMax	Scale	>=	>=	<=	<=
T0 (day)	✓	Inherit	TF	1	3.418337089185641E+02		☐	1.000E-04	1.000E+99	1.000E+00	☐	0.00000000000E+00	☐	0.00000000000E+00
DT (day)	☐	Value	T0	1	2.158166291081436E+03		☐	1.000E-03	1.000E+99	1.000E+00	☐	-1.00000000000E+20	☐	1.00000000000E+20
TF (day)	✓	Value	T0	1	2.500000000000000E+03		✓	1.000E-03	1.000E+99	3.000E+03	☐	-1.00000000000E+20	☐	1.00000000000E+20

(b) Copernicus GUI (v4.0). All dialogs are accessed as popup windows.

Figure 8: Copernicus GUI Evolution (2012 – 2014). The original Copernicus GUI (from 1.0 – 3.1) consisted of one main window with a deeply-nested tab structure. In the 4.0 release, the old "Solve + Plot" tab (which included the 2D and 3D graphics displays) was made the main window, with two panels on the left with the segment list and a treeview field for accessing the mission configuration dialogs (which were now separate popup windows). This upgrade allowed multiple dialogs to be viewed at the same time.



```
2025-07-15 15:46:25.237 :          37          DDEABM     0.748031E+00    0.278563E-03    0.137122E-06    0.000000E+00           545
2025-07-15 15:46:25.237 :          38          DDEABM     0.100000E+01    0.181767E-03    0.166038E-08    0.000000E+00           159
2025-07-15 15:46:25.237 :          45          DDEABM     0.248739E+00    0.789064E-06    0.359023E-11    0.000000E+00            71
2025-07-15 15:46:25.245 :          47          DDEABM     0.750000E+00    0.374339E-04    0.974704E-10    0.000000E+00            95
2025-07-15 15:46:25.252 :          48          DDEABM    -0.291931E+01    0.452118E-03    0.252872E-06    0.203705E-02           635
2025-07-15 15:46:25.260 :          59          DDEABM     0.735515E+00    0.273105E-03    0.119944E-06    0.199890E-02           469
2025-07-15 15:46:25.261 :          60          DDEABM     0.000000E+00    0.000000E+00    0.000000E+00    0.000000E+00             0
```

Figure 9: Copernicus 4.6 (2018). This was last release of the "classic" Copernicus architecture (the monolithic executable with the Winteracter-based GUI). Copernicus 5.0 was already being developed when this was released. The 4.6 release was used for Artemis I mission support at JSC.

original format was very difficult to parse.*

Recognizing the need for programmatic control over problem setup and execution, a Matlab scripting interface was developed, starting in 2007. This allowed users to manipulate input decks, configure optimization settings, execute Copernicus runs, and process results – all from within a Matlab environment. While effective, this approach required a commercial license, which limited accessibility. To overcome this constraint and further broaden adoption, a new Python scripting interface (CopPy) was developed as a replacement starting in 2014. The Python interface removed the licensing barrier and aligned Copernicus with modern, open-source scientific computing ecosystems. With this API, users could automatically modify and solve large numbers of input decks, monitor optimization status, extract key metrics, and perform post-processing – all within reproducible, scriptable workflows.

The original Matlab and Python scripting interfaces followed a similar architecture (i.e., they could be used to modify input decks, but not create them from scratch). In 2017, development began on a replacement Python scripting interface (RoboCopPy) which enables input decks to be created from scratch using only Python code. In addition, Copernicus now also incorporates a built-in Python-based parallelization engine, a powerful component that makes it relatively easy to set up and run large scans on an HPC cluster in a standardized and easy-to-configure way.[19]

These scripting capabilities have become essential for supporting design space exploration, uncertainty analysis, and multi-objective trade studies, making Copernicus not only an interactive design

*The new `.ideck` input deck format was a collection of Fortran namelists, with each variable printed on separate lines, making it more parsable and also more human readable. In recent releases, a new JSON input deck format has also been introduced, with the intent of eventually replacing the `.ideck` format.

environment but also a powerful engine for large-scale trajectory analysis.

General GUI Upgrades

The Copernicus GUI was extensively updated and improved at JSC (see Figure 8). The original GUI of one main window with a deeply-nested tab structure was eliminated in the 4.0 release in 2014. The updated 4.0 GUI layout had two panels on the left of the main window (Mission Segments and Mission Config), as well as a display of the log file at the bottom of the window.[*] The main part of the window was the old "Solve+Plot" tab, which contained the graphics tabs and the buttons used to propagate and iterate. The treeview field in the Mission Config dialog was used to access all the dialogs that were formerly in tabs (e.g., Algorithms, Segment Time, Segment State, etc.). Double-clicking an item in the treeview would open the dialog. With this upgrade, multiple dialogs could now be open at once (which was not possible before). The segment dialogs were populated based on the selected segment in the Mission Segments panel.

Models

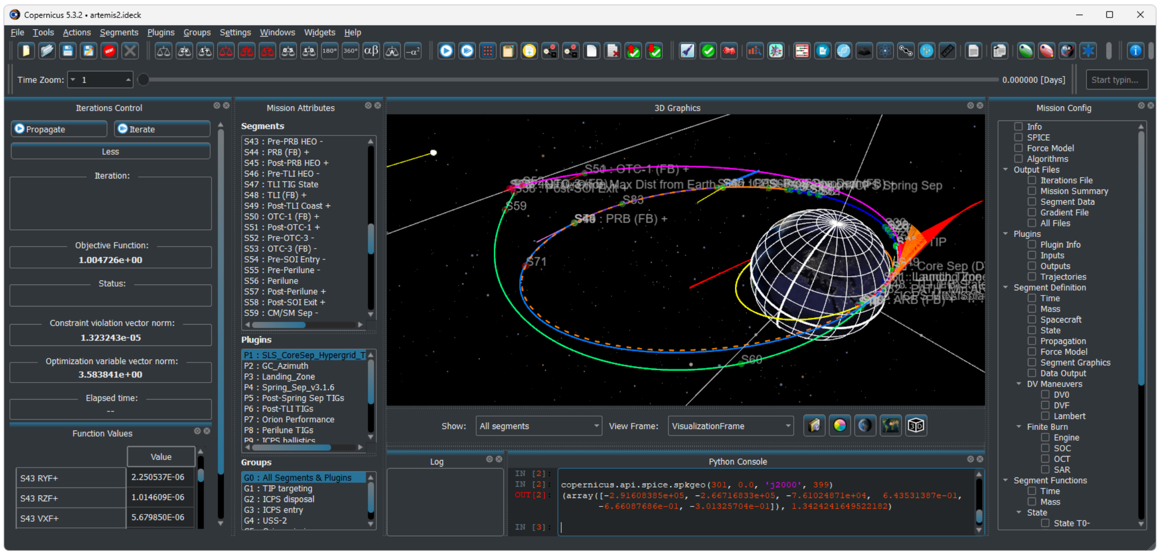
The availability of multiple models and algorithms for different domains is a key feature of Copernicus, and adding new models and options has been a key part of Copernicus development. The models are intended to be selected by the user, who has total freedom to define the problem in a way best suited to the application. The comprehensive set of models in Copernicus enables a wide range of fidelity options – from very low to very high – to be used within the same tool. Users can easily transition from lower- to higher-fidelity simulations with just a few operations in the GUI. For example, a trajectory can be quickly converged using point-mass gravity models and then seamlessly upgraded to a high-fidelity force model for final analysis. This capability reflects a core philosophy of Copernicus: to serve as a modular and flexible tool in which the user plays an active role in the solution process. By allowing users to tailor problem setups creatively and precisely, Copernicus supports both rapid prototyping and detailed, mission-level analysis within a unified environment.

Various models and algorithms in Copernicus include: integrators/propagators (many different flavors including Adams, Runge-Kutta, and Nyström), optimal control theory, parameter optimization (including industry-standard solvers such as SNOPT, SLSQP, and IPOPT), numerical differentiation (2–9 point finite difference methods, with tuning), planetary ephemerides (selectable via SPICE), reference frames (many different versions of inertial, rotating, rotating-pulsating, and quasi-inertial frames, from SPICE and user-supplied), finite burn engine models (low thrust, high thrust, SEP, etc.), finite burn maneuver models (including SOC, OCT, and SAR), impulsive maneuvers, Lambert targeting (using the impulsive Δv maneuvers to target states such as other segments or celestial bodies, including gravity assists), state parameterizations (including Cartesian, classical orbital elements, hyperbolic, equinoctial, Delaunay, etc.), maneuver parameterizations (including spherical and cartesian), gravity assists, halo orbits, atmosphere and SRP, and gravity models (pointmass, spherical harmonic, and polyhedral).

Copernicus 5.0 (2020)

Starting in 2017, the development team began a significant activity to rewrite the Copernicus GUI. This transition was motivated by several key factors. There was growing concern about relying on a commercial toolkit for the GUI, both in terms of long-term sustainability and accessibility

[*]These panels were all resizable, but not undockable or movable (although this was a frequent user request, it was not possible to implement with the Winteracter toolkit).



Iteration:

Objective Function:
1.004726e+00

Status:

Constraint violation vector norm:
1.323243e-05

Optimization variable vector norm:
3.583841e+00

Elapsed time:
--

Function Values

	Value
S43 RYF+	2.250537E-06
S43 RZF+	1.014609E-06
S43 VXF+	5.679850E-06

Segments
S43 : Pre-PRB HEO -
S44 : PRB (FB) +
S45 : Post-PRB HEO +
S46 : Pre-TLI HEO -
S47 : TLI TIG State
S48 : TLI (FB) +
S49 : Post-TLI Coast +
S50 : OTC-1 (FB) +
S51 : Post-OTC-1 +
S52 : Pre-OTC-3 -
S53 : OTC-3 (FB) -
S54 : Pre-SOI Entry -
S55 : Pre-Perilune -
S56 : Perilune
S57 : Post-Perilune +
S58 : Post-SOI Exit +
S59 : CM/SM Sep -

Plugins
P1 : SLS_CoreSep_Hypergrid_T
P2 : GC_Azimuth
P3 : Landing_Zone
P4 : Spring_Sep_v3.1.6
P5 : Post-Spring Sep TIGs
P6 : Post-TLI TIGs
P7 : Orion Performance
P8 : Perilune TIGs
P9 : ICPS ballistics

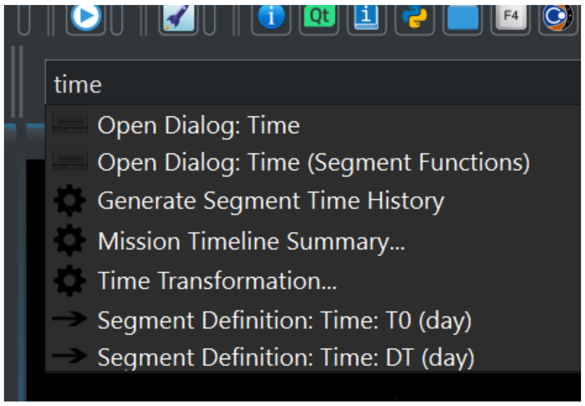
Groups
G0 : All Segments & Plugins
G1 : TIP targeting
G2 : ICPS disposal
G3 : ICPS entry
G4 : USS-2

0.000000 [Days]

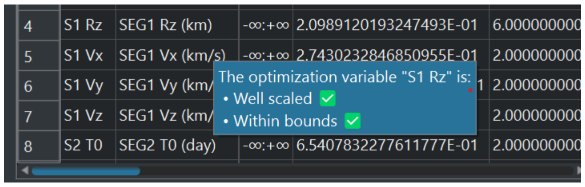
```
IN [2]: copernicus.api.spice.spkgeo(301, 0.0, 'j2000', 399)
OUT[2]: (array([-2.91608385e+05, -2.66716833e+05, -7.61024871e+04,  6.43531387e-01,
        -6.66087686e-01, -3.01325704e-01]), 1.3424241649522182)
IN [3]:
```

Mission Config
Info
SPICE
Force Model
Algorithms
Output Files
Iterations File
Mission Summary
Segment Data
Gradient File
All Files
Plugins
Plugin Info
Inputs
Outputs
Trajectories
Segment Definition
Time
Mass
Spacecraft
State
Propagation
Force Model
Segment Graphics
Data Output
DV Maneuvers
DV0
DVF
Lambert
Finite Burn
Engine
SOC
OCT
SAR
Segment Functions
Time
Mass
State
State T0-

Figure 10: Copernicus 5.3.2 (2024). The latest release of Copernicus. The rewrite of the GUI in Python for the 5.0 release was a significant update, enabling may new features, usability enhancements, and cross-platform use.


time
Open Dialog: Time
Open Dialog: Time (Segment Functions)
Generate Segment Time History
Mission Timeline Summary...
Time Transformation...
Segment Definition: Time: T0 (day)
Segment Definition: Time: DT (day)

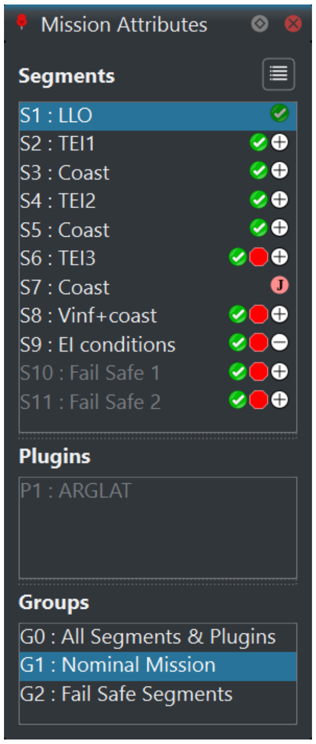
(a) A command palette allows for accessing common tasks using a text field, where the user can start typing and be presented with a list of selectable actions.



4	S1 Rz	SEG1 Rz (km)	-∞:+∞	2.0989120193247493E-01	6.000000000
5	S1 Vx	SEG1 Vx (km/s)	-∞:+∞	2.7430232846850955E-01	2.000000000
6	S1 Vy	SEG1 Vy (km/		1	2.000000000
7	S1 Vz	SEG1 Vz (km/			2.000000000
8	S2 T0	SEG2 T0 (day)	-∞:+∞	6.5407832277611777E-01	2.000000000

The optimization variable "S1 Rz" is:
• Well scaled ✓
• Within bounds ✓

(c) GUI tooltips provide help messages and useful information to the user.


Mission Attributes

Segments
S1 : LLO
S2 : TEI1
S3 : Coast
S4 : TEI2
S5 : Coast
S6 : TEI3
S7 : Coast
S8 : Vinf+coast
S9 : EI conditions
S10 : Fail Safe 1
S11 : Fail Safe 2

Plugins
P1 : ARGLAT

Groups
G0 : All Segments & Plugins
G1 : Nominal Mission
G2 : Fail Safe Segments

(b) Widgets can be docked and reconfigured.

Figure 11: Selected GUI Features. The new Python/Qt GUI has enabled the use of modern UI features and a more dynamic and configurable system than was possible before. Being user-friendly remains a core focus for Copernicus.

for potential new developers. As Copernicus's use cases expanded and expectations for user experience evolved, it became increasingly important to offer a more modern, user-friendly, and flexible interface. Additionally, the original GUI – built on the Winteracter framework – began to show its limitations. Certain features required by users could not be implemented within its constraints[*], and support for stable and full-featured cross-platform development was lacking.[†] These limitations ultimately drove the need to explore more modern solutions better aligned with the future direction of the tool. Over the years, the Fortran programming language had also incorporated new features, such as built-in interoperability with C, which meant that a Fortran-only GUI solution was no longer really required, and the Python ecosystem began to achieve very wide use, meaning there were a lot of other options available. The developer and user experiences with the Python scripting interface also indicated the desirability of a Python-based solution.

The 5.0 release, completed in 2020, included some of the most significant changes to the program since the first release.[6] The original Winteracter-based GUI was completely rewritten in Python and the open source Qt GUI toolkit, and the core Fortran code was rearchitected as a shared library that the GUI loaded (rather than having the GUI and core code intertwined). This new architecture allowed for possibilities not previously possible, such as: an API that exposed some of the underlying core routines to the user that can be used for other purposes (e.g., importing Copernicus as a generic Python library for orbital mechanics applications), Python plugins that called custom code written by the user and incorporated into the optimization problem (which can also use the API), an embedded Python console in the GUI, the ability to augment the GUI and add custom features to it, and the ability to build custom tools on top of Copernicus for other applications such as operations.[‡]

The new GUI is much more configurable than the original (e.g., users can dock/undock widgets and configure them however they want). It includes various refinements to make the tool easier to use and accessible (see Figures 10 – 11). Font size can be increased, tooltips provide useful information, variables can be clicked and the dialog automatically opened where they are defined, and there are numerous themes (possibly a unique feature among NASA's trajectory tool suite).

As part of these upgrades to the tool, the development environment was also updated. The developers switched from Subversion to Git for version control and GitLab for issue tracking.[§] An automated Continuous Integration (CI) system was set up using GitLab, where each commit is unit tested, replacing the manual running of unit tests up to that point. CMake was used to enable cross-platform building (on Windows, this auto-generates the Visual Studio project automatically, replacing the hand-modified one that had been used since 2007; on Linux and macOS, CMake-generated make files are used[¶]). Code coverage and other metrics are generated, and developer documentation is auto generated and deployed. The end result is a much more efficient developer workflow.

[*]For example, Winteracter-based GUIs were limited to one graphics window, which precluded the ability to show multiple 3D scenes simultaneously.

[†]A Mac build of Copernicus (released with v4.5) using the Winteracter/X11 OpenMotif-based library was developed, but was found to have many stability and usability issues.

[‡]The internal architecture of Copernicus separates the core computational engine as a standalone library. This library has never been publicly released, but has been used internally at JSC. The Python API is an evolution of that basic concept in a modern form that is potentially more user-friendly for trajectory analysts.

[§]Originally the JSC development team used Trac, and then later Redmine, for issue tracking.

[¶]Visual Studio (with Intel Fortran integrations) had always been used to compile Copernicus on Windows since the migration from CVF, while on Linux the SCons tool was used originally and then later the Foray tool.

RECENT WORK

At JSC, after many years of studies, planning, and analysis, Orion mission design has entered a period of pre-flight data product generation and real-time operations support, and many Copernicus users and developers have shifted to this mode as well. Design and optimization of the Orion trajectories (including nominal missions and off-nominal contingencies) remains a core function of Copernicus, but post-processing, data product generation (reports), pre-flight scans, and operations support are also major components. To prepare for this, a great deal of work has been done on the Copernicus Python API. This API provides a foundation upon which to build scripts and other tools that can perform these other functions, using the core capabilities of Copernicus. The flexibility of the Python ecosystem has been an enabler of many new use cases that were not possible with the old statically-compiled GUI. For example, user-created Python plugins can now leverage any Python library for computation, and custom displays can be added to the GUI for greater insight into a user's specific use case.

In 2024, Intel discontinued its "classic" Fortran compiler (ifort) and replaced it with a new LLVM-based version (ifx). On Windows and Linux, the next release of Copernicus (5.4) will be compiled with this new compiler (some minor updates were required in the code base). On macOS, work has proceeded on making an Apple Silicon native build. Since Intel did not port ifort or ifx to the new processor architecture, recent releases for Mac have had to run in Apple's x86 Rosetta 2 emulation mode. A native build requires switching to a native compiler, and the GNU Gfortran compiler was selected. This port required slightly more effort since Copernicus was using some nonstandard Fortran idioms and internal libraries that were supported by CVF/ifort/ifx that are not available in GFortran, so replacements had to be implemented.

MISSIONS

Copernicus has been used on numerous projects over the years and has become one of NASA's main tools for advanced spacecraft trajectory optimization. It is also the core tool used at JSC to answer the "what if" type questions that feed into the design of the overall human spaceflight architecture, from Constellation to Artemis to Human Landing System (HLS). Its use (2006–present) from the early days of the Orion project was described in an earlier paper.[1] The LCROSS mission (2009) was the first flight mission that used Copernicus, which was one of the key tools used to understand the lunar impact trajectory for different launch dates and geometries.[20] On November 14, 2022 the CAPSTONE spacecraft entered into a lunar NRHO designed using Copernicus.[21] The CAPSTONE ballistic lunar transfer trajectory was also designed using Copernicus.[22] On November 16, 2022, the Artemis I mission launched, as the first flight test of the SLS launch vehicle and Orion spacecraft.[1] The trajectory was a lunar DRO mission, designed in Copernicus. It was the most complicated trajectory flown by a crew-rated spacecraft in decades, and the first JSC-designed Copernicus trajectory to fly. The mission splashed down on December 11, 2022. For the first time, Copernicus was also used to support JSC Mission Control Center (MCC) for flight operations during the mission.[23]

Another first for Copernicus was on February 22, 2024, when the Intuitive Machines IM-1 "Odysseus" spacecraft landed on the lunar surface. IM-1 flew a trajectory designed exclusively in Copernicus, making it the trajectory design tool used for mankind's first commercial lunar lander mission. Subsequently, Copernicus was used to design the IM-2 "Athena" spacecraft's trajectory which landed on the Moon on March 6, 2025, carrying onboard a strip from Cesar Ocampo's iconic hat and a Copernicus logo (see Figure 12).[*] Copernicus is also being used to design trajectories and support operations for IM's IM-3 and IM-4 missions as part of NASA's Commercial Lunar Payload Services (CLPS) initiative, as well as IM's Lunar Data Network constellation which will be used by programs including NASA's Near Space Network Services (NSNS).

At NASA JSC and other centers, Copernicus is being used to design the Artemis II, III, IV trajectories and beyond. It is being used for designing the complex HLS trajectories to and from the lunar surface, and the deployment of the Gateway station to a lunar NRHO. This work continues, with the ultimate goal to return humans to the Moon and eventually to Mars.

FUTURE WORK

The 5.4 release of Copernicus is currently under development.[†] Copernicus continues to evolve and new features contemplated, including: graphics updates such as multiple 3D graphics windows (made possible by the new Qt-based GUI), high-fidelity celestial body terrain, and direct manipulation of the trajectory in the graphics;[24] more internal changes to allow for more flexible use cases, including a new input deck format; consolidation of the Python scripting interface and API; event finding (e.g., stopping the propagation of a segment on an event, rather than just time); expanding the API for accessing/modifying data during propagation of a segment; continuing to expand the parallelization engine;[19] handling spacecraft attitude; upgraded two-body relative motion frames; and even more GUI refinements and usability enhancements.

CONCLUSIONS

The history of the first quarter-century of Copernicus, as well as the impact of its creator Dr. Cesar Ocampo, has been presented. Copernicus is an example of a NASA software tool (developed in partnership with academia, government, and commercial partners) that has been actively maintained and developed, growing and evolving as new software technologies and mission design and optimization methods become available. It has become a workhorse tool at JSC and in commercial space for advanced mission design, and is a vital resource for the U.S. space program in the quest to develop a sustained human presence beyond Earth orbit.

ACKNOWLEDGMENTS

Special thanks to Cesar A. Ocampo (1967 – 2024), the original creator of Copernicus, a teacher, colleague, mentor, and friend who had a profound impact on trajectory design at NASA, and on the lives of many. Some of the biographical and historical material in this paper is based on unpublished material by Cesar written in 2020. The authors also wish to thank all those who have contributed to Copernicus over the years, especially: Açmae El Yacoubi, Anu Kamath, Brian Killeen, Christopher Foster, David Lee, Elizabeth Williams, Fady Morcos, Jerry Condon, Juan Senent, Keven Bokelmann, Leonard Kramer, Mark Jesick, Matthew Ruschmann, Max Widner, Nadège Pie, Quentin

[*]Cesar Ocampo passed away on September 16, 2024 in Houston, Texas, but not before learning that the Copernicus logo would be included on the lander as a tribute by his former students and colleagues. We would like to acknowledge his family for their donation of his hat. On October 25, 2024 the flag at the JSC MCC was also flown at half staff in his honor.

[†]"Copernicus has and will continue to revolutionize the way we and future analysts think about and design current and future spacecraft missions. In one single system we can solve a broad range of trajectory problems to the levels of accuracy needed. I have used the system to teach students of all ages, solve real problems, and explore new problems. A good system like this will never be completed . . . it will evolve indefinitely." – Cesar Ocampo

2002
2014
2020
NASA Johnson Space Center
Copernicus Spacecraft Trajectory Design and Optimization System
NASA

(a) Closeup of the IM-2 Lander. **(b) Copernicus logos through the years.** **(c) Cesar Ocampo at JSC in 2023.**

Figure 12: A strip of Cesar's iconic hat (zip-tied to cables) was installed on the Nova-C IM-2 lander, along with a modified Copernicus logo patch. It landed on the Moon on March 6, 2025.

Moore, Randy Eckman, Rob Falck, Robert Harpold, Ryan Whitley, Sarah Faron, Sean McArdle, Trent Couse, and Virginia Martin.

TERMINOLOGY

API	Application Programming Interface
CAPSTONE	Cislunar Autonomous Positioning System Technology Operations and Navigation Experiment
CDDF	Center Director Discretionary Funding
CLPS	Commercial Lunar Payload Services
CI	Continuous Integration
CVF	Compaq Visual Fortran
DEC	Digital Equipment Corporation
DRO	Distant Retrograde Orbit
EG	Aeroscience and Flight Mechanics Division
GEO	Geosynchronous Orbit
GMAT	General Mission Analysis Tool
GSFC	Goddard Space Flight Center
GTO	Geosynchronous Transfer Orbit
GUI	Graphical User Interface
HLS	Human Landing System
HPC	High Performance Computing
HSC	Hughes Space and Communications Company
IM	Intuitive Machines
ISP	In-Space Propulsion
JPL	Jet Propulsion Laboratory
JSC	Johnson Space Center
JSON	JavaScript Object Notation
LCROSS	Lunar Crater Observation and Sensing Satellite
LLVM	Low Level Virtual Machine
LTTT	Low-Thrust Trajectory Tools
MCC	Mission Control Center
NASA	National Aeronautics and Space Administration
NRHO	Near Rectilinear Halo Orbit
NSNS	Near Space Network Services
OCT	Optimal Control Theory
SAR	Single Axis Rotation
SEP	Solar Electric Propulsion
SIRTF	Space Infrared Telescope Facility
SLS	Space Launch System
SOC	Sub-Optimal Control
SOTY	Software of the Year
SPICE	Spacecraft, Planet, Instruments, C-matrix and Events
SRP	Solar Radiation Pressure
TIM	Technical Interchange Meeting
UT	University of Texas at Austin
UI	User Interface
VSE	Vision for Space Exploration
XIPSTOP	Xenon Ion Propulsion System Trajectory Optimization Program

REFERENCES

[1] J. Williams, T. F. Dawn, and A. L. Batcha, "A History of Orion Mission Design, Copernicus Software Development, and the Artemis I Trajectory," AAS/AIAA Astrodynamics Specialist Conference, Aug. 2023. AAS 23-241.

[2] C. Ocampo, "An Architecture for a Generalized Trajectory Design and Optimization System," *Proceedings of the Conference: Libration Point Orbits and Applications* (G. Gómez, M. W. Lo, and J. J. Masdemont, eds.), World Scientific Publishing Company, June 2003, pp. 529–572. Aiguablava, Spain.

[3] C. Ocampo, "Finite Burn Maneuver Modeling for a Generalized Spacecraft Trajectory Design and Optimization System," *Annals of the New York Academy of Science*, Vol. 1017, May 2004, pp. 210–233.

[4] C. Ocampo, J. S. Senent, and J. Williams, "Theoretical Foundation of Copernicus: A Unified System for Trajectory Design and Optimization," *4th International Conference on Astrodynamics Tools and Techniques. Madrid, Spain.*, May 2010.

[5] J. Williams, J. S. Senent, and D. E. Lee., "Recent Improvements to the Copernicus Trajectory Design and Optimization System," *Advances in the Astronautical Sciences*, Vol. 143, 2012. AAS 12-236.

[6] J. Williams, A. Kamath, R. Eckman, G. Condon, R. Mathur, and D. Davis, "Copernicus 5.0: Latest Advances in JSC's Spacecraft Trajectory Optimization and Design System," AAS/AIAA Astrodynamics Specialist Conference, Portland, ME, Aug. 2019. AAS 19-719.

[7] C. Ocampo, "Computer Graphics Applied to the Visualization of Two-Body Orbital Mechanics Problems," 28th Aerospace Sciences Meeting, Jan. 1990. AIAA 90-0075.

[8] C. Ocampo, "Solar Orbit Options for Space Infrared Telescope Facility (SIRTF)," Jet Propulsion Laboratory Inter-Office Memorandum, Aug. 1991. 312/91.2-1684.

[9] C. Ocampo, "Trajectory Analysis for the Lunar Flyby Rescue of AsiaSat-3/HGS-1," *Annals of the New York Academy of Sciences*, Vol. 1065, No. 1, 2005, pp. 232–253.

[10] C. A. Ocampo and G. W. Rosborough, "Transfer Trajectories for Distant Retrograde Orbiters of the Earth," AIAA Spaceflight Mechanics Meeting, Feb. 1993.

[11] C. A. Ocampo, "Method of Simultaneously Reducing Inclination and Eccentricity for Geostationary Orbit Transfer," Jan. 2002. US Patent 6,341,749.

[12] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in Fortran 77: The Art of Scientific Computing*. Cambridge University Press, 2nd ed., 1992.

[13] L. D. Kos, T. P. Polsgrove, R. C. Hopkins, D. Thomas, and J. A. Sims, "Overview of the Development for a Suite of Low-Thrust Trajectory Analysis Tools," *AIAA/AAS Astrodynamics Specialist Conference*, August 21-24 2006. AIAA 2006-6743.

[14] R. Mathur and C. A. Ocampo, "An Architecture for Incorporating Interactive Visualizations into Scientific Simulations," *Advances in the Astronautical Sciences*, Vol. 127, 2007. AAS 07-157.

[15] J. Williams, "A New Architecture for Extending the Capabilities of the Copernicus Trajectory Optimization Program," *Advances in the Astronautical Sciences: Astrodynamics 2015*, Vol. 156, 2016. AAS 15-606.

[16] S. L. McCarty and M. McGuire, "Parallel Monotonic Basin Hopping for Low Thrust Trajectory Optimization," *28th AIAA/AAS Space Flight Mechanics Meeting*, Jan. 2018. AIAA 2018-1452.

[17] S. Lionel, "Has it really been 35 years?," `https://stevelionel.com/drfortran/2013/10/02/has-it-really-been-35-years/`, Oct. 2013.

[18] J. Williams, R. D. Falck, and I. B. Beekman, "Application of Modern Fortran to Spacecraft Trajectory Design and Optimization," *2018 Space Flight Mechanics Meeting, AIAA SciTech Forum*, 2018. AIAA 2018-1451.

[19] Q. Moore, J. Williams, and B. J. Killeen, "A New Architecture for Parallelization of Complex Spacecraft Trajectory Optimization Scans," AAS/AIAA Astrodynamics Specialist Conference, Aug. 2023. AAS 23-252.

[20] D. S. Cooley, K. F. Galal, K. Berry, L. Janes, G. Marr, J. Carrico, and C. Ocampo, "Mission Design for the Lunar CRater Observation and Sensing Satellite (LCROSS)," AIAA/AAS Astrodynamics Specialist Conference, Toronto, Ontario Canada, Aug. 2010. AIAA 2010-8386.

[21] J. Williams, D. E. Lee, R. L. Whitley, K. A. Bokelmann, D. C. Davis, and C. F. Berry, "Targeting Cislunar Near Rectilinear Halo Orbits for Human Space Exploration," AAS/AIAA Space Flight Mechanics Meeting, Feb. 2017. AAS 17-267.

[22] N. L. Parrish, E. Kayser, S. Udupa, J. S. Parker, B. W. Cheetham, and D. C. Davis, "Survey of Ballistic Lunar Transfers to Near Rectilinear Halo Orbit," AAS/AIAA Astrodynamics Specialist Conference, Portland, ME, Aug. 2019. AAS 19-740.

[23] R. A. Eckman, C. P. Barrett, B. J. Killeen, and A. L. Batcha, "Trajectory Operations of the Artemis I Mission," AAS/AIAA Astrodynamics Specialist Conference, Aug. 2023. AAS 23-363.

[24] R. Mathur, "Visual Interactive Trajectory Design," *29th AAS/AIAA Space Flight Mechanics Meeting*, Jan. 2019. AAS 19-449.