poliastro: a Python library for interactive astrodynamics

Juan Luis Cano Rodríguez‡*, Jorge Martínez Garrido‡

https://www.youtube.com/watch?v=VCpTgU1pb5k

Abstract—Space is more popular than ever, with the growing public awareness of interplanetary scientific missions, as well as the increasingly large number of satellite companies planning to deploy satellite constellations. Python has become a fundamental technology in the astronomical sciences, and it has also caught the attention of the Space Engineering community.

One of the requirements for designing a space mission is studying the trajectories of satellites, probes, and other artificial objects, usually ignoring non-gravitational forces or treating them as perturbations: the so-called n-body problem. However, for preliminary design studies and most practical purposes, it is sufficient to consider only two bodies: the object under study and its attractor.

Even though the two-body problem has many analytical solutions, orbit propagation (the initial value problem) and targeting (the boundary value problem) remain computationally intensive because of long propagation times, tight tolerances, and vast solution spaces. On the other hand, astrodynamics researchers often do not share the source code they used to run analyses and simulations, which makes it challenging to try out new solutions.

This paper presents poliastro, an open-source Python library for interactive astrodynamics that features an easy-to-use API and tools for quick visualization. poliastro implements core astrodynamics algorithms (such as the resolution of the Kepler and Lambert problems) and leverages numba, a Just-in-Time compiler for scientific Python, to optimize the running time. Thanks to Astropy, poliastro can perform seamless coordinate frame conversions and use proper physical units and timescales. At the moment, poliastro is the longest-lived Python library for astrodynamics, has contributors from all around the world, and several New Space companies and people in academia use it.

Index Terms—astrodynamics, orbital mechanics, orbit propagation, orbit visualization, two-body problem

Introduction

History

The term "astrodynamics" was coined by the American astronomer Samuel Herrick, who received encouragement from the space pioneer Robert H. Goddard, and refers to the branch of space science dealing with the motion of artificial celestial bodies ([Dub73], [Her71]). However, the roots of its mathematical foundations go back several centuries.

Kepler first introduced his laws of planetary motion in 1609 and 1619 and derived his famous transcendental equation (1), which we now see as capturing a restricted form of the two-body problem. This work was generalized by Newton to give birth to the n-body problem, and many other mathematicians worked on it throughout the centuries (Daniel and Johann Bernoulli, Euler, Gauss). Poincaré established in the 1890s that no general closed-form solution exists for the n-body problem, since the resulting dynamical system is chaotic [Bat99]. Sundman proved in the 1900s the existence of convergent solutions for a few restricted with $n = 3$.

$$M = E - e \sin E \tag{1}$$

In 1903 Tsiokovsky evaluated the conditions required for artificial objects to leave the orbit of the earth; this is considered as a foundational contribution to the field of astrodynamics. Tsiokovsky devised equation 2 which relates the increase in velocity with the effective exhaust velocity of thrusted gases and the fraction of used propellant.

$$\Delta v = v_e \ln \frac{m_0}{m_f} \tag{2}$$

Further developments by Kondratyuk, Hohmann, and Oberth in the early 20th century all added to the growing field of orbital mechanics, which in turn enabled the development of space flight in the USSR and the United States in the 1950s and 1960s.

The two-body problem

In a system of $i \in 1, ..., n$ bodies subject to their mutual attraction, by application of Newton's law of universal gravitation, the total force $\mathbf{f}_i$ affecting m_i due to the presence of the other $n-1$ masses is given by [Bat99]:

$$\mathbf{f}_i = -G \sum_{j \neq i}^{n} \frac{m_i m_j}{|\mathbf{r}_{ij}|^3} \mathbf{r}_{ij} \tag{3}$$

where $G = 6.67430 \cdot 10^{-11}$ N m^2 kg^{-2} is the universal gravitational constant, and $\mathbf{r}_{ij}$ denotes the position vector from m_i to m_j. Applying Newton's second law of motion results in a system of n differential equations:

$$\frac{\mathrm{d}^2 \mathbf{r}_i}{\mathrm{d}t^2} = -G \sum_{j \neq i}^{n} \frac{m_j}{|\mathbf{r}_{ij}|^3} \mathbf{r}_{ij} \tag{4}$$

By setting $n = 2$ in 4 and subtracting the two resulting equalities, one arrives to the **fundamental equation of the two-body problem**:

$$\frac{\mathrm{d}^2 \mathbf{r}}{\mathrm{d}t^2} = -\frac{\mu}{r^3} \mathbf{r} \tag{5}$$

where $\mu = G(m_1 + m_2) = G(M + m)$. When $m \ll M$ (for example, an artificial satellite orbiting a planet), one can consider $\mu = GM$ a property of the attractor.

* *Corresponding author: hello@juanlu.space*
‡ *Unaffiliated*

Copyright © 2022 Juan Luis Cano Rodríguez et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Keplerian vs non-keplerian motion

Conveniently manipulating equation 5 leads to several properties [Bat99] that were already published by Johannes Kepler in the 1610s, namely:

1) The orbit always describes a conic section (an ellipse, a parabola, or an hyperbola), with the attractor at one of the two foci and can be written in polar coordinates like $r = \frac{p}{1+e\cos\nu}$ (Kepler's first law).
2) The magnitude of the specific angular momentum $h = r^2 \frac{\mathrm{d}\theta}{\mathrm{d}t}$ is constant an equal to two times the areal velocity (Kepler's second law).
3) For closed (circular and elliptical) orbits, the period is related to the size of the orbit through $P = 2\pi\sqrt{\frac{a^3}{\mu}}$ (Kepler's third law).

For many practical purposes it is usually sufficient to limit the study to one object orbiting an attractor and ignore all other external forces of the system, hence restricting the study to trajectories governed by equation 5. Such trajectories are called "Keplerian", and several problems can be formulated for them:

- The **initial-value problem**, which is usually called **propagation**, involves determining the position and velocity of an object after an elapse period of time given some initial conditions.
- **Preliminary orbit determination**, which involves using exact or approximate methods to derive a Keplerian orbit from a set of observations.
- The **boundary-value problem**, often named **the Lambert problem**, which involves determining a Keplerian orbit from boundary conditions, usually departure and arrival position vectors and a time of flight.

Fortunately, most of these problems boil down to finding numerical solutions to relatively simple algebraic relations between time and angular variables: for elliptic motion ($0 \leq e < 1$) it is the Kepler equation, and equivalent relations exist for the other eccentricity regimes [Bat99]. Numerical solutions for these equations can be found in a number of different ways, each one with different complexity and precision tradeoffs. In the Methods section we list the ones implemented by poliastro.

On the other hand, there are many situations in which natural and artificial orbital perturbations must be taken into account so that the actual non-Keplerian motion can be properly analyzed:

- Interplanetary travel in the proximity of other planets. On a first approximation it is usually enough to study the trajectory in segments and focus the analysis on the closest attractor, hence patching several Keplerian orbits along the way (the so-called "patched-conic approximation") [Bat99]. The boundary surface that separates one segment from the other is called the sphere of influence.
- Use of solar sails, electric propulsion, or other means of continuous thrust. Devising the optimal guidance laws that minimize travel time or fuel consumption under these conditions is usually treated as an optimization problem of a dynamical system, and as such it is particularly challenging [Con14].
- Artificial satellites in the vicinity of a planet. This is the regime in which all the commercial space industry operates, especially for those satellites in Low-Earth Orbit (LEO).

State of the art

In our view, at the time of creating poliastro there were a number of issues with existing open source astrodynamics software that posed a barrier of entry for novices and amateur practitioners. Most of these barriers still exist today and are described in the following paragraphs. The goals of the project can be condensed as follows:

1) Set an example on reproducibility and good coding practices in astrodynamics.
2) Become an approachable software even for novices.
3) Offer a performant software that can be also used in scripting and interactive workflows.

The most mature software libraries for astrodynamics are arguably Orekit [noa22c], a "low level space dynamics library written in Java" with an open governance model, and SPICE [noa22d], a toolkit developed by NASA's Navigation and Ancillary Information Facility at the Jet Propulsion Laboratory. Other similar, smaller projects that appeared later on and that are still maintained to this day include PyKEP [IBD+20], beyond [noa22a], tudatpy [noa22e], sbpy [MKDVB+19], Skyfield [Rho20] (Python), CelestLab (Scilab) [noa22b], astrodynamics.jl (Julia) [noa] and Nyx (Rust) [noa21a]. In addition, there are some Graphical User Interface (GUI) based open source programs used for Mission Analysis and orbit visualization, such as GMAT [noa20] and gpredict [noa18], and complete web applications for tracking constellations of satellites like the SatNOGS project by the Libre Space Foundation [noa21b].

The level of quality and maintenance of these packages is somewhat heterogeneous. Community-led projects with a strong corporate backing like Orekit are in excellent health, while on the other hand smaller projects developed by volunteers (beyond, astrodynamics.jl) or with limited institutional support (PyKEP, GMAT) suffer from lack of maintenance. Part of the problem might stem from the fact that most scientists are never taught how to build software efficiently, let alone the skills to collaboratively develop software in the open [WAB+14], and astrodynamicists are no exception.

On the other hand, it is often difficult to translate the advances in astrodynamics research to software. Classical algorithms developed throughout the 20th century are described in papers that are sometimes difficult to find, and source code or validation data is almost never available. When it comes to modern research carried in the digital era, source code and validation data is still difficult, even though they are supposedly provided "upon reasonable request" [SSM18] [GBP22].

It is no surprise that astrodynamics software often requires deep expertise. However, there are often implicit assumptions that are not documented with an adequate level of detail which originate widespread misconceptions and lead even seasoned professionals to make conceptual mistakes. Some of the most notorious misconceptions arise around the use of general perturbations data (OMMs and TLEs) [Fin07], the geometric interpretation of the mean anomaly [Bat99], or coordinate transformations [VCHK06].

Finally, few of the open source software libraries mentioned above are amenable to scripting or interactive use, as promoted by computational notebooks like Jupyter [KRKP+16].

The following sections will now discuss the various areas of current research that an astrodynamicist will engage in, and how poliastro improves their workflow.

Methods

Software Architecture

The architecture of poliastro emerges from the following set of conflicting requirements:

1) There should be a high-level API that enables users to perform orbital calculations in a straightforward way and prevent typical mistakes.
2) The running time of the algorithms should be within the same order of magnitude of existing compiled implementations.
3) The library should be written in a popular open-source language to maximize adoption and lower the barrier to external contributors.

One of the most typical mistakes we set ourselves to prevent with the high-level API is dimensional errors. Addition and substraction operations of physical quantities are defined only for quantities with the same units [Dro53]: for example, the operation $1\ \mathrm{km} + 100\ \mathrm{m}$ requires a scale transformation of at least one of the operands, since they have different units (kilometers and meters) but the same dimension (length), whereas the operation $1\ \mathrm{km} + 1\ \mathrm{kg}$ is directly not allowed because dimensions are incompatible (length and mass). As such, software systems operating with physical quantities should raise exceptions when adding different dimensions, and transparently perform the required scale transformations when adding different units of the same dimension.

With this in mind, we evaluated several Python packages for unit handling (see [JGAZJT+18] for a recent survey) and chose `astropy.units` [TPWS+18].

```
radius = 6000  # km
altitude = 500  # m

# Wrong!
distance = radius + altitude

from astropy import units as u

# Correct
distance = (radius << u.km) + (altitude << u.m)
```

This notion of providing a "safe" API extends to other parts of the library by leveraging other capabilities of the Astropy project. For example, timestamps use `astropy.time` objects, which take care of the appropriate handling of time scales (such as TDB or UTC), reference frame conversions leverage `astropy.coordinates`, and so forth.

One of the drawbacks of existing unit packages is that they impose a significant performance penalty. Even though `astropy.units` is integrated with NumPy, hence allowing the creation of array quantities, all the unit compatibility checks are implemented in Python and require lots of introspection, and this can slow down mathematical operations by several orders of magnitude. As such, to fulfill our desired performance requirement for poliastro, we envisioned a two-layer architecture:

- The **Core API** follows a procedural style, and all the functions receive Python numerical types and NumPy arrays for maximum performance.
- The **High level API** is object-oriented, all the methods receive Astropy `Quantity` objects with physical units, and computations are deferred to the Core API.

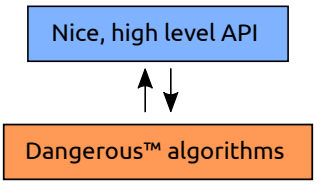

Fig. 1: *poliastro two-layer architecture*

Most of the methods of the High level API consist only of the necessary unit compatibility checks, plus a wrapper over the corresponding Core API function that performs the actual computation.

```
@u.quantity_input(E=u.rad, ecc=u.one)
def E_to_nu(E, ecc):
    """True anomaly from eccentric anomaly."""
    return (
        E_to_nu_fast(
            E.to_value(u.rad),
            ecc.value
        ) << u.rad
    ).to(E.unit)
```

As a result, poliastro offers a unit-safe API that performs the least amount of computation possible to minimize the performance penalty of unit checks, and also a unit-unsafe API that offers maximum performance at the cost of not performing any unit validation checks.

Finally, there are several options to write performant code that can be used from Python, and one of them is using a fast, compiled language for the CPU intensive parts. Successful examples of this include NumPy, written in C [HMvdW+20], SciPy, featuring a mix of FORTRAN, C, and C++ code [VGO+20], and pandas, making heavy use of Cython [BBC+11]. However, having to write code in two different languages hinders the development speed, makes debugging more difficult, and narrows the potential contributor base (what Julia creators called "The Two Language Problem" [BEKS17]).

As authors of poliastro we wanted to use Python as the sole programming language of the implementation, and the best solution we found to improve its performance was to use Numba, a LLVM-based Python JIT compiler [LPS15].

Usage

Basic `Orbit` and `Ephem` creation

The two central objects of the poliastro high level API are `Orbit` and `Ephem`:

- `Orbit` objects represent an osculating (hence Keplerian) orbit of a dimensionless object around an attractor at a given point in time and a certain reference frame.
- `Ephem` objects represent an ephemerides, a sequence of spatial coordinates over a period of time in a certain reference frame.

There are six parameters that uniquely determine a Keplerian orbit, plus the gravitational parameter of the corresponding attractor (k or μ). Optionally, an epoch that contextualizes the orbit can be included as well. This set of six parameters is not unique, and several of them have been developed over the years to serve different purposes. The most widely used ones are:

- **Cartesian elements**: Three components for the position (x, y, z) and three components for the velocity (v_x, v_y, v_z). This set has no singularities.

- **Classical Keplerian elements**: Two components for the shape of the conic (usually the semimajor axis a or semiparameter p and the eccentricity e), three Euler angles for the orientation of the orbital plane in space (inclination i, right ascension of the ascending node Ω, and argument of periapsis ω), and one polar angle for the position of the body along the conic (usually true anomaly f or ν). This set of elements has an easy geometrical interpretation and the advantage that, in pure two-body motion, five of them are fixed $(a, e, i, \Omega, \omega)$ and only one is time-dependent (ν), which greatly simplifies the analytical treatment of orbital perturbations. However, they suffer from singularities steming from the Euler angles ("gimbal lock") and equations expressed in them are ill-conditioned near such singularities.
- **Walker modified equinoctial elements**: Six parameters (p, f, g, h, k, L). Only L is time-dependent and this set has no singularities, however the geometrical interpretation of the rest of the elements is lost [WIO85].

Here is how to create an `Orbit` from cartesian and from classical Keplerian elements. Walker modified equinoctial elements are supported as well.

```
from astropy import units as u

from poliastro.bodies import Earth, Sun
from poliastro.twobody import Orbit
from poliastro.constants import J2000

# Data from Curtis, example 4.3
r = [-6045, -3490, 2500] << u.km
v = [-3.457, 6.618, 2.533] << u.km / u.s

orb_curtis = Orbit.from_vectors(
    Earth,  # Attractor
    r, v  # Elements
)

# Data for Mars at J2000 from JPL HORIZONS
a = 1.523679 << u.au
ecc = 0.093315 << u.one
inc = 1.85 << u.deg
raan = 49.562 << u.deg
argp = 286.537 << u.deg
nu = 23.33 << u.deg

orb_mars = Orbit.from_classical(
    Sun,
    a, ecc, inc, raan, argp, nu,
    J2000  # Epoch
)
```

When displayed on an interactive REPL, `Orbit` objects provide basic information about the geometry, the attractor, and the epoch:

```
>>> orb_curtis
7283 x 10293 km x 153.2 deg (GCRS) orbit
around Earth (X) at epoch J2000.000 (TT)

>>> orb_mars
1 x 2 AU x 1.9 deg (HCRS) orbit
around Sun (X) at epoch J2000.000 (TT)
```

Similarly, `Ephem` objects can be created using a variety of classmethods as well. Thanks to `astropy.coordinates` built-in low-fidelity ephemerides, as well as its capability to remotely access the JPL HORIZONS system, the user can seamlessly build an object that contains the time history of the position of any Solar System body:

```
from astropy.time import Time
from astropy.coordinates import solar_system_ephemeris
from poliastro.ephem import Ephem

# Configure high fidelity ephemerides globally
# (requires network access)
solar_system_ephemeris.set("jpl")

# For predefined poliastro attractors
earth = Ephem.from_body(Earth, Time.now().tdb)

# For the rest of the Solar System bodies
ceres = Ephem.from_horizons("Ceres", Time.now().tdb)
```

There are some crucial differences between `Orbit` and `Ephem` objects:

- `Orbit` objects have an attractor, whereas `Ephem` objects do not. Ephemerides can originate from complex trajectories that don't necessarily conform to the ideal two-body problem.
- `Orbit` objects capture a precise instant in a two-body motion plus the necessary information to propagate it forward in time indefinitely, whereas `Ephem` objects represent a bounded time history of a trajectory. This is because the equations for the two-body motion are known, whereas an ephemeris is either an observation or a prediction that cannot be extrapolated in any case without external knowledge. As such, `Orbit` objects have a `.propagate` method, but `Ephem` ones do not. This prevents users from attempting to propagate the position of the planets, which will always yield poor results compared to the excellent ephemerides calculated by external entities.

Finally, both types have methods to convert between them:

- `Ephem.from_orbit` is the equivalent of sampling a two-body motion over a given time interval. As explained above, the resulting `Ephem` loses the information about the original attractor.
- `Orbit.from_ephem` is the equivalent of calculating the osculating orbit at a certain point of a trajectory, assuming a given attractor. The resulting `Orbit` loses the information about the original, potentially complex trajectory.

Orbit propagation

`Orbit` objects have a `.propagate` method that takes an elapsed time and returns another `Orbit` with new orbital elements and an updated epoch:

```
>>> from poliastro.examples import iss

>>> iss
>>> 6772 x 6790 km x 51.6 deg (GCRS) ...

>>> iss.nu.to(u.deg)
<Quantity 46.59580468 deg>

>>> iss_30m = iss.propagate(30 << u.min)

>>> (iss_30m.epoch - iss.epoch).datetime
datetime.timedelta(seconds=1800)

>>> (iss_30m.nu - iss.nu).to(u.deg)
<Quantity 116.54513153 deg>
```

The default propagation algorithm is an analytical procedure described in [FCM13] that works seamlessly in the near parabolic region. In addition, poliastro implements analytical propagation algorithms as described in [DB83], [OG86], [Mar95], [Mik87], [PP13], [Cha22], and [VM07].

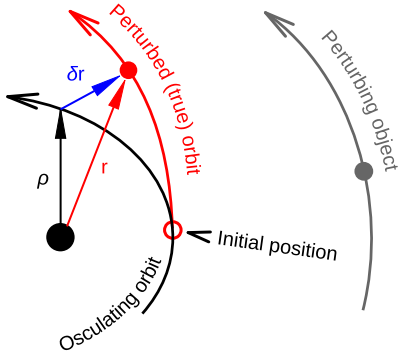

Fig. 2: *Osculating (Keplerian) vs perturbed (true) orbit (source: Wikipedia, CC BY-SA 3.0)*

Natural perturbations

As showcased in Figure 2, at any point in a trajectory we can define an ideal Keplerian orbit with the same position and velocity under the attraction of a point mass: this is called the osculating orbit. Some numerical propagation methods exist that model the true, perturbed orbit as a deviation from an evolving, osculating orbit. poliastro implements Cowell's method [CC10], which consists in adding all the perturbation accelerations and then integrating the resulting differential equation with any numerical method of choice:

$$\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} = -\frac{\mu}{r^3}\mathbf{r} + \mathbf{a}_d \tag{6}$$

The resulting equation is usually integrated using high order numerical methods, since the integration times are quite large and the tolerances comparatively tight. An in-depth discussion of such methods can be found in [HNW09]. poliastro uses Dormand-Prince 8(5,3) (`DOP853`), a commonly used method available in SciPy [HMvdW+20].

There are several natural perturbations included: J2 and J3 gravitational terms, several atmospheric drag models (exponential, [Jac77], [AAAA62], [AAA+76]), and helpers for third body gravitational attraction and radiation pressure as described in [**?**].

```
@njit
def combined_a_d(
    t0, state, k, j2, r_eq, c_d, a_over_m, h0, rho0
):
    return (
        J2_perturbation(
            t0, state, k, j2, r_eq
        ) + atmospheric_drag_exponential(
            t0, state, k, r_eq, c_d, a_over_m, h0, rho0
        )
    )

def f(t0, state, k):
    du_kep = func_twobody(t0, state, k)
    ax, ay, az = combined_a_d(
        t0,
        state,
        k,
        R=R,
        C_D=C_D,
        A_over_m=A_over_m,
        H0=H0,
        rho0=rho0,
        J2=Earth.J2.value,
    )
    du_ad = np.array([0, 0, 0, ax, ay, az])

    return du_kep + du_ad
```

```
rr = propagate(
    orbit,
    tofs,
    method=cowell,
    f=f,
)
```

Continuous thrust control laws

Beyond natural perturbations, spacecraft can modify their trajectory on purpose by using impulsive maneuvers (as explained in the next section) as well as continuous thrust guidance laws. The user can define custom guidance laws by providing a perturbation acceleration in the same way natural perturbations are used. In addition, poliastro includes several analytical solutions for continuous thrust guidance laws with specific purposes, as studied in [CR17]: optimal transfer between circular coplanar orbits [Ede61] [Bur67], optimal transfer between circular inclined orbits [Ede61] [Kec97], quasi-optimal eccentricity-only change [Pol97], simultaneous eccentricity and inclination change [Pol00], and agument of periapsis adjustment [Pol98]. A much more rigorous analysis of a similar set of laws can be found in [DCV21].

```
from poliastro.twobody.thrust import change_ecc_inc

ecc_f = 0.0 << u.one
inc_f = 20.0 << u.deg
f = 2.4e-6 << (u.km / u.s**2)

a_d, _, t_f = change_ecc_inc(orbit, ecc_f, inc_f, f)
```

Impulsive maneuvers

Impulsive maneuvers are modeled considering a change in the velocity of a spacecraft while its position remains fixed. The `poliastro.maneuver.Maneuver` class provides various constructors to instantiate popular impulsive maneuvers in the framework of the non-perturbed two-body problem:

- `Maneuver.impulse`
- `Maneuver.hohmann`
- `Maneuver.bielliptic`
- `Maneuver.lambert`

```
from poliastro.maneuver import Maneuver

orb_i = Orbit.circular(Earth, alt=700 << u.km)
hoh = Maneuver.hohmann(orb_i, r_f=36000 << u.km)
```

Once instantiated, `Maneuver` objects provide information regarding total Δv and Δt:

```
>>> hoh.get_total_cost()
<Quantity 3.6173981270031357 km / s>

>>> hoh.get_total_time()
<Quantity 15729.741535747102 s>
```

`Maneuver` objects can be applied to `Orbit` instances using the `apply_maneuver` method.

```
>>> orb_i
7078 x 7078 km x 0.0 deg (GCRS) orbit
around Earth (X)

>>> orb_f = orb_i.apply_maneuver(hoh)
>>> orb_f
36000 x 36000 km x 0.0 deg (GCRS) orbit
around Earth (X)
```

Targeting

Targeting is the problem of finding the orbit connecting two positions over a finite amount of time. Within the context of the non-perturbed two-body problem, targeting is just a matter of solving the BVP, also known as Lambert's problem. Because targeting tries to find for an orbit, the problem is included in the Initial Orbit Determination field.

The `poliastro.iod` package contains `izzo` and `vallado` modules. These provide a `lambert` function for solving the targeting problem. Nevertheless, a `Maneuver.lambert` constructor is also provided so users can keep taking advantage of `Orbit` objects.

```
# Declare departure and arrival datetimes
date_launch = time.Time(
    '2011-11-26 15:02', scale='tdb'
)
date_arrival = time.Time(
    '2012-08-06 05:17', scale='tdb'
)

# Define initial and final orbits
orb_earth = Orbit.from_ephem(
    Sun, Ephem.from_body(Earth, date_launch),
    date_launch
)
orb_mars = Orbit.from_ephem(
    Sun, Ephem.from_body(Mars, date_arrival),
    date_arrival
)

# Compute targetting maneuver and apply it
man_lambert = Maneuver.lambert(orb_earth, orb_mars)
orb_trans, orb_target = ss0.apply_maneuver(
    man_lambert, intermediate=true
)
```

Targeting is closely related to quick mission design by means of porkchop diagrams. These are contour plots showing all combinations of departure and arrival dates with the specific energy for each transfer orbit. They allow for quick identification of the most optimal transfer dates between two bodies.

The `poliastro.plotting.porkchop` provides the `PorkchopPlotter` class which allows the user to generate these diagrams.

```
from poliastro.plotting.porkchop import (
    PorkchopPlotter
)
from poliastro.utils import time_range

# Generate all launch and arrival dates
launch_span = time_range(
    "2020-03-01", end="2020-10-01", periods=int(150)
)
arrival_span = time_range(
    "2020-10-01", end="2021-05-01", periods=int(150)
)

# Create an instance of the porkchop and plot it
porkchop = PorkchopPlotter(
    Earth, Mars, launch_span, arrival_span,
)
```

Previous code, with some additional customization, generates figure 3.

Plotting

For visualization purposes, poliastro provides the `poliastro.plotting` package, which contains various utilities for generating 2D and 3D graphics using different backends such as matplotlib [Hun07] and Plotly [Inc15].

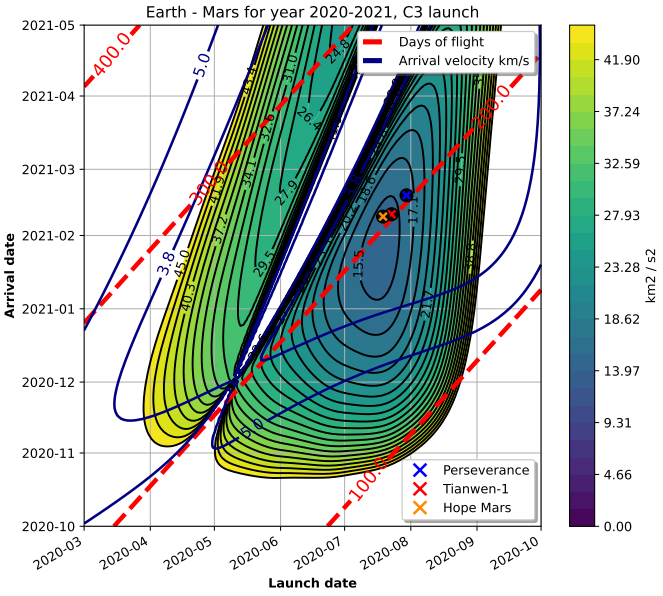

Fig. 3: *Porkchop plot for Earth-Mars transfer arrival energy showing latest missions to the Martian planet.*

Generated graphics can be static or interactive. The main difference between these two is the ability to modify the camera view in a dynamic way when using interactive plotters.

The most important classes in the `poliastro.plotting` package are `StaticOrbitPlotter` and `OrbitPlotter3D`. In addition, the `poliastro.plotting.misc` module contains the `plot_solar_system` function, which allows the user to visualize inner and outter both in 2D and 3D, as requested by users.

The following example illustrates the plotting capabilities of poliastro. At first, orbits to be plotted are computed and their plotting style is declared:

```
from poliastro.plotting.misc import plot_solar_system

# Current datetime
now = Time.now().tdb

# Obtain Florence and Halley orbits
florence = Orbit.from_sbdb("Florence")
halley_1835_ephem = Ephem.from_horizons(
    "90000031", now
)
halley_1835 = Orbit.from_ephem(
    Sun, halley_1835_ephem, halley_1835_ephem.epochs[0]
)

# Define orbit labels and color style
florence_style = {label: "Florence", color: "#000000"}
halley_style = {label: "Florence", color: "#84B0B8"}
```

The static two-dimensional plot can be created using the following code:

```
# Generate a static 2D figure
frame2D = rame = plot_solar_system(
    epoch=now, outer=False
)
frame2D.plot(florence, **florence_style)
frame2D.plot(florence, **halley_style)
```

As a result, figure 4 is obtained.

The interactive three-dimensional plot can be created using the following code:

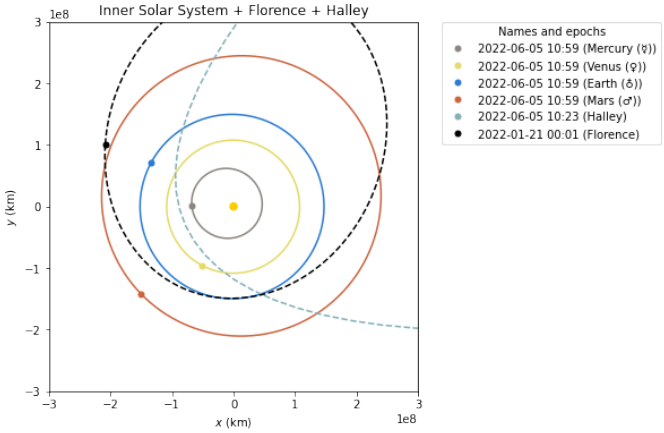

Fig. 4: *Two-dimensional view of the inner Solar System, Florence, and Halley.*

```
# Generate an interactive 3D figure
frame3D = rame = plot_solar_system(
    epoch=now, outer=False,
    use_3d=True, interactive=true
)
frame3D.plot(florence, **florence_style)
frame3D.plot(florence, **halley_style)
```

As a result, figure 5 is obtained.

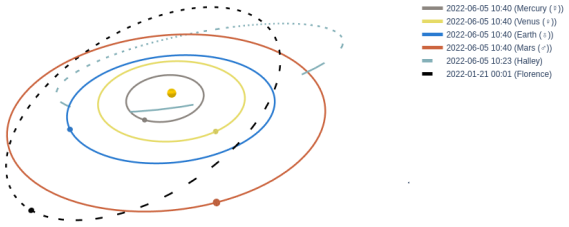

Fig. 5: *Three-dimensional view of the inner Solar System, Florence, and Halley.*

Commercial Earth satellites

Figure 6 gives a clear picture of the most important natural perturbations affecting satellites in LEO, namely: the first harmonic of the geopotential field J_2 (representing the attractor oblateness), the atmospheric drag, and the higher order harmonics of the geopotential field.

At least the most significant of these perturbations need to be taken into account when propagating LEO orbits, and therefore the methods for purely Keplerian motion are not enough. As seen above, poliastro implements a number of these perturbations already - however, numerical methods are much slower than analytical ones, and this can render them unsuitable for large scale simulations, satellite conjunction assesment, propagation in constrained hardware, and so forth.

To address this issue, semianalytical propagation methods were devised that attempt to strike a balance between the fast running times of analytical methods and the necessary inclusion of perturbation forces. One of such semianalytical methods are

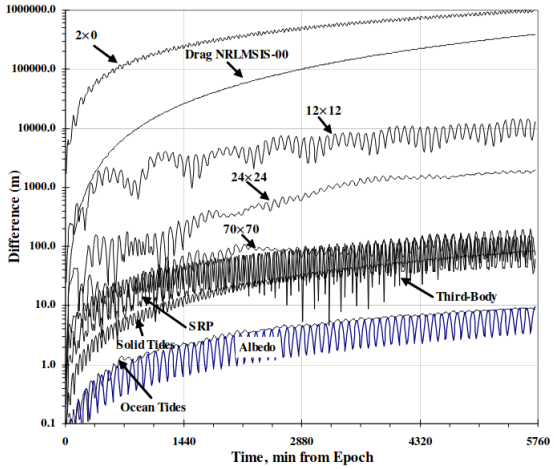

Fig. 6: *Natural perturbations affecting Low-Earth Orbit (LEO) motion (source: [VM07])*

the Simplified General Perturbation (SGP) models, first developed in [HK66] and then refined in [LC69] into what we know these days as the SGP4 propagator [HR80] [VCHK06]. Even though certain elements of the reference frame used by SGP4 are not properly specified [VCHK06] and that its accuracy might still be too limited for certain applications [Ko09] [Lar16], it is nowadays the most widely used propagation method thanks in large part to the dissemination of General Perturbations orbital data by the US 501(c)(3) CelesTrak (which itself obtains it from the 18th Space Defense Squadron of the US Space Force).

The starting point of SGP4 is a special element set that uses Brouwer mean orbital elements [Bro59] plus a ballistic coefficient based on an approximation of the atmospheric drag [LC69], and its results are expressed in a special coordinate system called True Equator Mean Equinox (TEME). Special care needs to be taken to avoid mixing mean elements with osculating elements, and to convert the output of the propagation to the appropriate reference frame. These element sets have been traditionally distributed in a compact text representation called Two-Line Element sets (TLEs) (see 7 for an example). However this format is quite cryptic and suffers from a number of shortcomings, so recently there has been a push to use the Orbit Data Messages international standard developed by the Consultive Committee for Space Data Systems (CCSDS 502.0-B-2).

```
1 25544U 98067A   22156.15037205  .00008547  00000+0  15823-3 0  9994
2 25544  51.6449  36.2070 0004577 196.3587 298.4146 15.49876730343319
```

Fig. 7: *Two-Line Element set (TLE) for the ISS (retrieved on 2022-06-05)*

At the moment, general perturbations data both in OMM and TLE format can be integrated with poliastro thanks to the `sgp4` Python library and the `Ephem` class as follows:

```
from astropy.coordinates import TEME, GCRS

from poliastro.ephem import Ephem
from poliastro.frames import Planes


def ephem_from_gp(sat, times):
    errors, rs, vs = sat.sgp4_array(times.jd1, times.jd2)
    if not (errors == 0).all():
        warn(
            "Some objects could not be propagated, "
```

```
            "proceeding with the rest",
            stacklevel=2,
        )
        rs = rs[errors == 0]
        vs = vs[errors == 0]
        times = times[errors == 0]

    cart_teme = CartesianRepresentation(
        rs << u.km,
        xyz_axis=-1,
        differentials=CartesianDifferential(
            vs << (u.km / u.s),
            xyz_axis=-1,
        ),
    )
    cart_gcrs = (
        TEME(cart_teme, obstime=times)
        .transform_to(GCRS(obstime=times))
        .cartesian
    )

    return Ephem(
        cart_gcrs,
        times,
        plane=Planes.EARTH_EQUATOR
    )
```

However, no native integration with SGP4 has been implemented yet in poliastro, for technical and non-technical reasons. On one hand, this propagator is too different from the other methods, and we have not yet devised how to add it to the library in a way that does not create confusion. On the other hand, adding such a propagator to poliastro would probably open the flood gates of corporate users of the library, and we would like to first devise a sustainability strategy for the project, which is addressed in the next section.

Future work

Despite the fact that poliastro has existed for almost a decade, for most of its history it has been developed by volunteers on their free time, and only in the past five years it has received funding through various Summer of Code programs (SOCIS 2017, GSOC 2018-2021) and institutional grants (NumFOCUS 2020, 2021). The funded work has had an overwhemingly positive impact on the project, however the lack of a dedicated maintainer has caused some technical debt to accrue over the years, and some parts of the project are in need of refactoring or better documentation.

Historically, poliastro has tried to implement algorithms that were applicable for all the planets in the Solar System, however some of them have proved to be very difficult to generalize for bodies other than the Earth. For cases like these, poliastro ships a `poliastro.earth` package, but going forward we would like to continue embracing a generic approach that can serve other bodies as well.

Several open source projects have successfully used poliastro or were created taking inspiration from it, like spacetech-ssa by IBM[1] or mubody [BBVPFSC22]. AGI (previously Analytical Graphics, Inc., now Ansys Government Initiatives) published a series of scripts to automate the commercial tool STK from Python leveraging poliastro[2]. However, we have observed that there is still lots of repeated code across similar open source libraries written in Python, which means that there is an opportunity to provide a "kernel" of algorithms that can be easily reused. Although `poliastro.core` started as a separate layer to isolate fast, non-safe functions as described above, we think we could move it to an external package so it can be depended upon by projects that do not want to use some of the higher level poliastro abstractions or drag its large number of heavy dependencies.

Finally, the sustainability of the project cannot yet be taken for granted: the project has reached a level of complexity that already warrants dedicated development effort that cannot be covered with short-lived grants. Such funding could potentially come from the private sector, but although there is evidence that several for-profit companies are using poliastro, we have very little information of how is it being used and what problems are those users having, let alone what avenues for funded work could potentially work. Organizations like the Libre Space Foundation advocate for a strong copyleft licensing model to convince commercial actors to contribute to the commons, but in principle that goes against the permissive licensing that the wider Scientific Python ecosystem, including poliastro, has adopted. With the advent of new business models and the ever increasing reliance in open source by the private sector, a variety of ways to engage commercial users and include them in the conversation exist. However, these have not been explored yet.

Acknowledgements

The authors would like to thank Prof. Michèle Lavagna for her original guidance and inspiration, David A. Vallado for his encouragement and for publishing the source code for the algorithms from his book for free, Dr. T.S. Kelso for his tireless efforts in maintaining CelesTrak, Alejandro Sáez for sharing the dream of a better way, Prof. Dr. Manuel Sanjurjo Rivo for believing in my work, Helge Eichhorn for his enthusiasm and decisive influence in poliastro, the whole OpenAstronomy collaboration for opening the door for us, the NumFOCUS organization for their immense support, and Alexandra Elbakyan for enabling scientific progress worldwide.

References

[AAA+76] United States Committee on Extension to the Standard Atmosphere, United States National Aeronautics, Space Administration, United States National Oceanic, Atmospheric Administration, and United States Air Force. *U.S. Standard Atmosphere, 1976*. NOAA - SIT 76-1562. National Oceanic and Amospheric [sic] Administration, 1976. URL: https://books.google.es/books?id=x488AAAAIAAJ.

[AAAA62] United States Committee on Extension to the Standard Atmosphere, United States National Aeronautics, Space Administration, and United States Environmental Science Services Administration. *U.S. Standard Atmosphere, 1962: ICAO Standard Atmosphere to 20 Kilometers; Proposed ICAO Extension to 32 Kilometers; Tables and Data to 700 Kilometers*. U.S. Government Printing Office, 1962. URL: https://books.google.es/books?id=fWdTAAAAMAAJ.

[Bat99] Richard H. Battin. *An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition*. American Institute of Aeronautics and Astronautics, Inc., Reston, VA, January 1999. URL: https://arc.aiaa.org/doi/book/10.2514/4.861543, doi:10.2514/4.861543.

[BBC+11] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The Best of Both Worlds. *Computing in Science & Engineering*, 13(2):31–39, March 2011. URL: http://ieeexplore.ieee.org/document/5582062/, doi:10.1109/MCSE.2010.118.

[BBVPFSC22] Juan Bermejo Ballesteros, José María Vergara Pérez, Alejandro Fernández Soler, and Javier Cubas. Mubody, an astrodynamics open-source Python library focused on libration points. Barcelona, Spain, April 2022. URL: https://sseasymposium.org/wp-content/uploads/2022/04/4thSSEA_AllAbstracts.pdf.

1. https://github.com/IBM/spacetech-ssa
2. https://github.com/AnalyticalGraphicsInc/STKCodeExamples/

[BEKS17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, January 2017. URL: https://epubs.siam.org/doi/10.1137/141000671, doi:10.1137/141000671.

[Bro59] Dirk Brouwer. Solution of the problem of artificial satellite theory without drag. *The Astronomical Journal*, 64:378, November 1959. URL: http://adsabs.harvard.edu/cgi-bin/bib_query?1959AJ.....64..378B, doi:10.1086/107958.

[Bur67] E.G.C. Burt. On space manoeuvres with continuous thrust. *Planetary and Space Science*, 15(1):103–122, January 1967. URL: https://linkinghub.elsevier.com/retrieve/pii/0032063367900700, doi:10.1016/0032-0633(67)90070-0.

[CC10] Philip Herbert Cowell and Andrew Claude Crommelin. *Investigation of the Motion of Halley's Comet from 1759 to 1910*. Neill & Company, limited, 1910.

[Cha22] Kevin Charls. Recursive solution to Kepler's problem for elliptical orbits - application in robust Newton-Raphson and co-planar closest approach estimation. 2022. Publisher: Unpublished Version Number: 1. URL: https://rgdoi.net/10.13140/RG.2.2.18578.58563/1, doi:10.13140/RG.2.2.18578.58563/1.

[Con14] Bruce A. Conway. *Spacecraft trajectory optimization*. Number 29 in Cambridge aerospace series. Cambridge university press, Cambridge (GB), 2014.

[CR17] Juan Luis Cano Rodríguez. Study of analytical solutions for low-thrust trajectories. Master's thesis, Universidad Politécnica de Madrid, March 2017.

[DB83] J. M. A. Danby and T. M. Burkardt. The solution of Kepler's equation, I. *Celestial Mechanics*, 31(2):95–107, October 1983. URL: http://link.springer.com/10.1007/BF01686811, doi:10.1007/BF01686811.

[DCV21] Marilena Di Carlo and Massimiliano Vasile. Analytical solutions for low-thrust orbit transfers. *Celestial Mechanics and Dynamical Astronomy*, 133(7):33, July 2021. URL: https://link.springer.com/10.1007/s10569-021-10033-9, doi:10.1007/s10569-021-10033-9.

[Dro53] S. Drobot. On the foundations of Dimensional Analysis. *Studia Mathematica*, 14(1):84–99, 1953. URL: http://www.impan.pl/get/doi/10.4064/sm-14-1-84-99, doi:10.4064/sm-14-1-84-99.

[Dub73] G. N. Duboshin. Book Review: Samuel Herrick. Astrodynamics. *Soviet Astronomy*, 16:1064, June 1973. ADS Bibcode: 1973SvA....16.1064D. URL: https://ui.adsabs.harvard.edu/abs/1973SvA....16.1064D.

[Ede61] Theodore N. Edelbaum. Propulsion Requirements for Controllable Satellites. *ARS Journal*, 31(8):1079–1089, August 1961. URL: https://arc.aiaa.org/doi/10.2514/8.5723, doi:10.2514/8.5723.

[FCM13] Davide Farnocchia, Davide Bracali Cioci, and Andrea Milani. Robust resolution of Kepler's equation in all eccentricity regimes. *Celestial Mechanics and Dynamical Astronomy*, 116(1):21–34, May 2013. URL: http://link.springer.com/10.1007/s10569-013-9476-9, doi:10.1007/s10569-013-9476-9.

[Fin07] D Finkleman. "TLE or Not TLE?" That is the Question (AAS 07-126). *ADVANCES IN THE ASTRONAUTICAL SCIENCES*, 127(1):401, 2007. Publisher: Published for the American Astronautical Society by Univelt; 1999.

[GBP22] Mirko Gabelica, Ružica Bojčić, and Livia Puljak. Many researchers were not compliant with their published data sharing statement: mixed-methods study. *Journal of Clinical Epidemiology*, page S089543562200141X, May 2022. URL: https://linkinghub.elsevier.com/retrieve/pii/S089543562200141X, doi:10.1016/j.jclinepi.2022.05.019.

[Her71] Samuel Herrick. *Astrodynamics*. Van Nostrand Reinhold Co, London, New York, 1971.

[HK66] CG Hilton and JR Kuhlman. Mathematical models for the space defense center. *Philco-Ford Publication No. U-3871*, 17:28, 1966.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. URL: https://www.nature.com/articles/s41586-020-2649-2, doi:10.1038/s41586-020-2649-2.

[HNW09] E. Hairer, S. P. Nørsett, and Gerhard Wanner. *Solving ordinary differential equations I: nonstiff problems*. Number 8 in Springer series in computational mathematics. Springer, Heidelberg ; London, 2nd rev. ed edition, 2009. OCLC: ocn620251790.

[HR80] Felix R. Hoots and Ronald L. Roehrich. Models for propagation of NORAD element sets. Technical report, Defense Technical Information Center, Fort Belvoir, VA, December 1980. URL: http://www.dtic.mil/docs/citations/ADA093554.

[Hun07] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. Publisher: IEEE COMPUTER SOC. doi:10.1109/MCSE.2007.55.

[IBD+20] Dario Izzo, Will Binns, Dariomm098, Alessio Mereta, Christopher Iliffe Sprague, Dhennes, Bert Van Den Abbeele, Chris Andre, Krzysztof Nowak, Nat Guy, Alberto Isaac Barquín Murguía, Pablo, Frédéric Chapoton, GiacomoAcciarini, Moritz V. Looz, Dietmarwo, Mike Heddes, Anatoli Babenia, Baptiste Fournier, Johannes Simon, Jonathan Willitts, Mateusz Polnik, Sanjeev Narayanaswamy, The Gitter Badger, and Jack Yarndley. esa/pykep: Optimize, October 2020. URL: https://zenodo.org/record/4091753, doi:10.5281/ZENODO.4091753.

[Inc15] Plotly Technologies Inc. Collaborative data science, 2015. Place: Montreal, QC Publisher: Plotly Technologies Inc. URL: https://plot.ly.

[Jac77] L. G. Jacchia. Thermospheric Temperature, Density, and Composition: New Models. *SAO Special Report*, 375, March 1977. ADS Bibcode: 1977SAOSR.375.....J. URL: https://ui.adsabs.harvard.edu/abs/1977SAOSR.375.....J.

[JGAZJT+18] Nathan J. Goldbaum, John A. ZuHone, Matthew J. Turk, Kacper Kowalik, and Anna L. Rosen. unyt: Handle, manipulate, and convert data with units in Python. *Journal of Open Source Software*, 3(28):809, August 2018. URL: http://joss.theoj.org/papers/10.21105/joss.00809, doi:10.21105/joss.00809.

[Kec97] Jean Albert Kechichian. Reformulation of Edelbaum's Low-Thrust Transfer Problem Using Optimal Control Theory. *Journal of Guidance, Control, and Dynamics*, 20(5):988–994, September 1997. URL: https://arc.aiaa.org/doi/10.2514/2.4145, doi:10.2514/2.4145.

[Ko09] TS Kelso and others. Analysis of the Iridium 33-Cosmos 2251 collision. *Advances in the Astronautical Sciences*, 135(2):1099–1112, 2009. Publisher: Citeseer.

[KRKP+16] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, and others. *Jupyter Notebooks-a publishing format for reproducible computational workflows.*, volume 2016. 2016.

[Lar16] Martin Lara. Analytical and Semianalytical Propagation of Space Orbits: The Role of Polar-Nodal Variables. In Gerard Gómez and Josep J. Masdemont, editors, *Astrodynamics Network AstroNet-II*, volume 44, pages 151–166. Springer International Publishing, Cham, 2016. Series Title: Astrophysics and Space Science Proceedings. URL: http://link.springer.com/10.1007/978-3-319-23986-6_11, doi:10.1007/978-3-319-23986-6_11.

[LC69] M. H. Lane and K. Cranford. An improved analytical drag theory for the artificial satellite problem. In *Astrodynamics Conference*, Princeton,NJ,U.S.A., August 1969. American Institute of Aeronautics and Astronautics. URL: https://arc.aiaa.org/doi/10.2514/6.1969-925, doi:10.2514/6.1969-925.

[LPS15] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: a LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC - LLVM '15*, pages 1–6, Austin, Texas, 2015. ACM Press. URL: http://dl.acm.org/citation.cfm?doid=2833157.2833162, doi:10.1145/2833157.2833162.

[Mar95] F. Landis Markley. Kepler Equation solver. *Celestial Mechanics & Dynamical Astronomy*, 63(1):101–111,

1995. URL: http://link.springer.com/10.1007/BF00691917, doi:10.1007/BF00691917.

[Mik87] Seppo Mikkola. A cubic approximation for Kepler's equation. *Celestial Mechanics*, 40(3-4):329–334, September 1987. URL: http://link.springer.com/10.1007/BF01235850, doi:10.1007/BF01235850.

[MKDVB+19] Michael Mommert, Michael Kelley, Miguel De Val-Borro, Jian-Yang Li, Giannina Guzman, Brigitta Sipőcz, Josef Ďurech, Mikael Granvik, Will Grundy, Nick Moskovitz, Antti Penttilä, and Nalin Samarasinha. sbpy: A Python module for small-body planetary astronomy. *Journal of Open Source Software*, 4(38):1426, June 2019. URL: http://joss.theoj.org/papers/10.21105/joss.01426, doi:10.21105/joss.01426.

[noa] Astrodynamics.jl. URL: https://github.com/JuliaSpace/Astrodynamics.jl.

[noa18] gpredict, January 2018. URL: https://github.com/csete/gpredict/releases/tag/v2.2.1.

[noa20] GMAT, July 2020. URL: https://sourceforge.net/projects/gmat/files/GMAT/GMAT-R2020a/.

[noa21a] nyx, November 2021. URL: https://gitlab.com/nyx-space/nyx/-/tags/1.0.0.

[noa21b] SatNOGS, October 2021. URL: https://gitlab.com/librespacefoundation/satnogs/satnogs-client/-/tags/1.7.

[noa22a] beyond, January 2022. URL: https://pypi.org/project/beyond/0.7.4/.

[noa22b] celestlab, January 2022. URL: https://atoms.scilab.org/toolboxes/celestlab/3.4.1.

[noa22c] Orekit, June 2022. URL: https://gitlab.orekit.org/orekit/orekit/-/releases/11.2.

[noa22d] SPICE, January 2022. URL: https://naif.jpl.nasa.gov/naif/toolkit.html.

[noa22e] tudatpy, January 2022. URL: https://github.com/tudat-team/tudatpy/releases/tag/0.6.0.

[OG86] A. W. Odell and R. H. Gooding. Procedures for solving Kepler's equation. *Celestial Mechanics*, 38(4):307–334, April 1986. URL: http://link.springer.com/10.1007/BF01238923, doi:10.1007/BF01238923.

[Pol97] James E Pollard. Simplified approach for assessment of low-thrust elliptical orbit transfers. In *25th International Electric Propulsion Conference, Cleveland, OH*, pages 97–160, 1997.

[Pol98] James Pollard. Evaluation of low-thrust orbital maneuvers. In *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, Cleveland,OH,U.S.A., July 1998. American Institute of Aeronautics and Astronautics. URL: https://arc.aiaa.org/doi/10.2514/6.1998-3486, doi:10.2514/6.1998-3486.

[Pol00] J. E. Pollard. Simplified analysis of low-thrust orbital maneuvers. Technical report, Defense Technical Information Center, Fort Belvoir, VA, August 2000. URL: http://www.dtic.mil/docs/citations/ADA384536.

[PP13] Adonis Reinier Pimienta-Penalver. *Accurate Kepler equation solver without transcendental function evaluations*. State University of New York at Buffalo, 2013.

[Rho20] Brandon Rhodes. Skyfield: Generate high precision research-grade positions for stars, planets, moons, and Earth satellites, February 2020.

[SSM18] Victoria Stodden, Jennifer Seiler, and Zhaokun Ma. An empirical analysis of journal policy effectiveness for computational reproducibility. *Proceedings of the National Academy of Sciences*, 115(11):2584–2589, March 2018. URL: https://pnas.org/doi/full/10.1073/pnas.1708290115, doi:10.1073/pnas.1708290115.

[TPWS+18] The Astropy Collaboration, A. M. Price-Whelan, B. M. Sipőcz, H. M. Günther, P. L. Lim, S. M. Crawford, S. Conseil, D. L. Shupe, M. W. Craig, N. Dencheva, A. Ginsburg, J. T. VanderPlas, L. D. Bradley, D. Pérez-Suárez, M. de Val-Borro, (Primary Paper Contributors), T. L. Aldcroft, K. L. Cruz, T. P. Robitaille, E. J. Tollerud, (Astropy Coordination Committee), C. Ardelean, T. Babej, Y. P. Bach, M. Bachetti, A. V. Bakanov, S. P. Bamford, G. Barentsen, P. Barmby, A. Baumbach, K. L. Berry, F. Biscani, M. Boquien, K. A. Bostroem, L. G. Bouma, G. B. Brammer, E. M. Bray, H. Breytenbach, H. Buddelmeijer, D. J. Burke, G. Calderone, J. L. Cano Rodríguez, M. Cara, J. V. M. Cardoso, S. Cheedella, Y. Copin, L. Corrales, D. Crichton, D. D'Avella, C. Deil, É. Depagne, J. P. Dietrich, A. Donath, M. Droettboom, N. Earl, T. Erben, S. Fabbro, L. A. Ferreira, T. Finethy, R. T. Fox, L. H. Garrison, S. L. J. Gibbons, D. A. Goldstein, R. Gommers, J. P. Greco, P. Greenfield, A. M. Groener, F. Grollier, A. Hagen, P. Hirst, D. Homeier, A. J. Horton, G. Hosseinzadeh, L. Hu, J. S. Hunkeler, Ž. Ivezić, A. Jain, T. Jenness, G. Kanarek, S. Kendrew, N. S. Kern, W. E. Kerzendorf, A. Khvalko, J. King, D. Kirkby, A. M. Kulkarni, A. Kumar, A. Lee, D. Lenz, S. P. Littlefair, Z. Ma, D. M. Macleod, M. Mastropietro, C. McCully, S. Montagnac, B. M. Morris, M. Mueller, S. J. Mumford, D. Muna, N. A. Murphy, S. Nelson, G. H. Nguyen, J. P. Ninan, M. Nöthe, S. Ogaz, S. Oh, J. K. Parejko, N. Parley, S. Pascual, R. Patil, A. A. Patil, A. L. Plunkett, J. X. Prochaska, T. Rastogi, V. Reddy Janga, J. Sabater, P. Sakurikar, M. Seifert, L. E. Sherbert, H. Sherwood-Taylor, A. Y. Shih, J. Sick, M. T. Silbiger, S. Singanamalla, L. P. Singer, P. H. Sladen, K. A. Sooley, S. Sornarajah, O. Streicher, P. Teuben, S. W. Thomas, G. R. Tremblay, J. E. H. Turner, V. Terrón, M. H. van Kerkwijk, A. de la Vega, L. L. Watkins, B. A. Weaver, J. B. Whitmore, J. Woillez, V. Zabalza, and (Astropy Contributors). The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. *The Astronomical Journal*, 156(3):123, August 2018. URL: https://iopscience.iop.org/article/10.3847/1538-3881/aabc4f, doi:10.3847/1538-3881/aabc4f.

[VCHK06] David Vallado, Paul Crawford, Ricahrd Hujsak, and T.S. Kelso. Revisiting Spacetrack Report #3. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Keystone, Colorado, August 2006. American Institute of Aeronautics and Astronautics. URL: https://arc.aiaa.org/doi/10.2514/6.2006-6753, doi:10.2514/6.2006-6753.

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, SciPy 1.0 Contributors, Aditya Vijaykumar, Alessandro Pietro Bardelli, Alex Rothberg, Andreas Hilboll, Andreas Kloeckner, Anthony Scopatz, Antony Lee, Ariel Rokem, C. Nathan Woods, Chad Fulton, Charles Masson, Christian Häggström, Clark Fitzgerald, David A. Nicholson, David R. Hagen, Dmitrii V. Pasechnik, Emanuele Olivetti, Eric Martin, Eric Wieser, Fabrice Silva, Felix Lenders, Florian Wilhelm, G. Young, Gavin A. Price, Gert-Ludwig Ingold, Gregory E. Allen, Gregory R. Lee, Hervé Audren, Irvin Probst, Jörg P. Dietrich, Jacob Silterra, James T Webber, Janko Slavič, Joel Nothman, Johannes Buchner, Johannes Kulick, Johannes L. Schönberger, José Vinícius de Miranda Cardoso, Joscha Reimer, Joseph Harrington, Juan Luis Cano Rodríguez, Juan Nunez-Iglesias, Justin Kuczynski, Kevin Tritz, Martin Thoma, Matthew Newville, Matthias Kümmerer, Maximilian Bolingbroke, Michael Tartre, Mikhail Pak, Nathaniel J. Smith, Nikolai Nowaczyk, Nikolay Shebanov, Oleksandr Pavlyk, Per A. Brodtkorb, Perry Lee, Robert T. McGibbon, Roman Feldbauer, Sam Lewis, Sam Tygier, Scott Sievert, Sebastiano Vigna, Stefan Peterson, Surhud More, Tadeusz Pudlik, Takuya Oshima, Thomas J. Pingel, Thomas P. Robitaille, Thomas Spura, Thouis R. Jones, Tim Cera, Tim Leslie, Tiziano Zito, Tom Krauss, Utkarsh Upadhyay, Yaroslav O. Halchenko, and Yoshiki Vázquez-Baeza. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, March 2020. URL: http://www.nature.com/articles/s41592-019-0686-2, doi:10.1038/s41592-019-0686-2.

[VM07] David A. Vallado and Wayne D. McClain. *Fundamentals of astrodynamics and applications*. Number 21 in Space technology library. Microcosm Press [u.a.], Hawthorne, Calif., 3. ed., 1. printing edition, 2007.

[WAB+14] Greg Wilson, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumbley, Ben Waugh, Ethan P. White, and Paul Wilson. Best Practices for Scientific Computing. *PLoS Biology*, 12(1):e1001745, January 2014. URL: https://dx.plos.org/10.1371/journal.pbio.

1001745, doi:10.1371/journal.pbio.1001745.

[WIO85] M. J. H. Walker, B. Ireland, and Joyce Owens. A set modified equinoctial orbit elements. *Celestial Mechanics*, 36(4):409–419, August 1985. URL: http://link.springer.com/10.1007/BF01227493, doi:10.1007/BF01227493.