ORBITM

Release 1.1

Samuel Y. W. Low

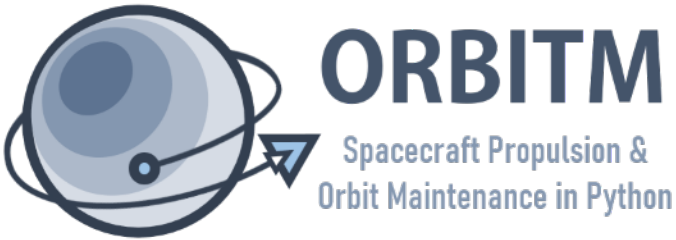**Jun 16, 2022**

GETTING STARTED

Github

https://github.com/sammmlow/ORBITM

Documents

https://orbitm.readthedocs.io/en/latest/

Version

1.1 (Latest)

Author

Samuel Y. W. Low

ORBITM stands for the **Orbit Maintenance and Propulsion Sizing Tool**, and it is an open-source, easy-to-use, free-ware orbit maintenance simulator and propulsion sizing tool, for anyone with a Python 3 installation. The software comes with its own built-in orbit decay model and maintenance simulator, a life-time estimator, and it is also interface-able with AGI's Systems Tool Kit (STK) as an alternative mode of simulation.

The objective of ORBITM is to allow for a quick sizing of low Earth orbit (LEO) mission lifetime, while sizing the mission against propulsion units of the user's choosing. The user may input into the UI their orbital parameters, spacecraft properties (mass, area, drag parameters etc), and the software will compute your desired V necessary for the mission, track the altitude profile over time, and plot the thruster-fuel profile that satisfies the orbit maintenance needs of your mission.

Document tree for ORBITM is listed out below.

ONE
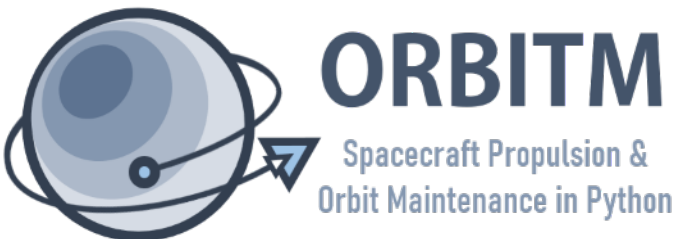
AN OVERVIEW

Why was ORBITM created? There are two key reasons behind ORBITM. First, the proliferation of tens of thousands of LEO satellites in our near future mandates a quick and efficient means of estimating the lifetime of not just the orbiters but also of debris. The second reason is that ORBITM can also be used to support rapid iterations of spacecraft designs in agile development. For example, different iterations of solar panels or radar reflector designs that are proposed frequently, would impact the drag characteristics and mission life-time. ORBITM allows for the rapid ball-park assessments of such scenarios.
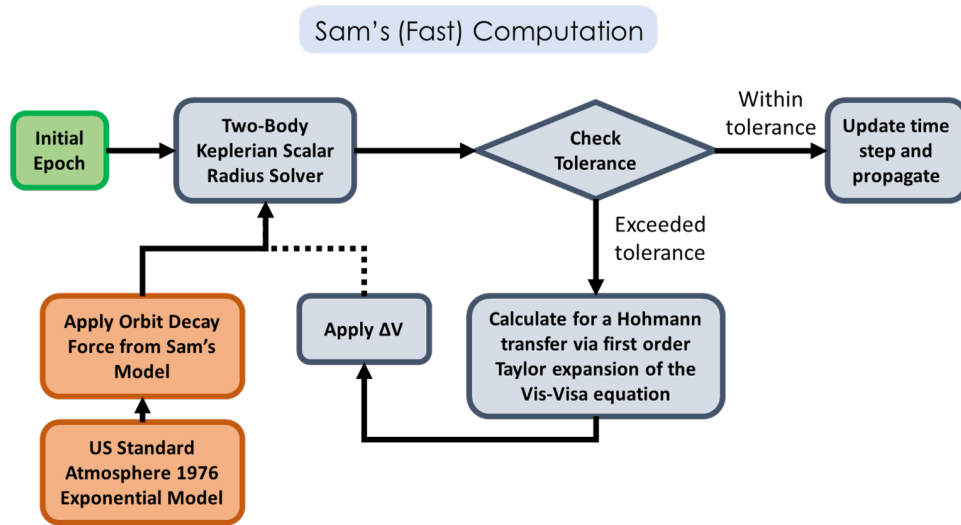

Fig. 1: Figure 1.1 - Agile development demands that the mission planner can perform frequent mission life comparisons over rapidly changing iterations of spacecraft structures.
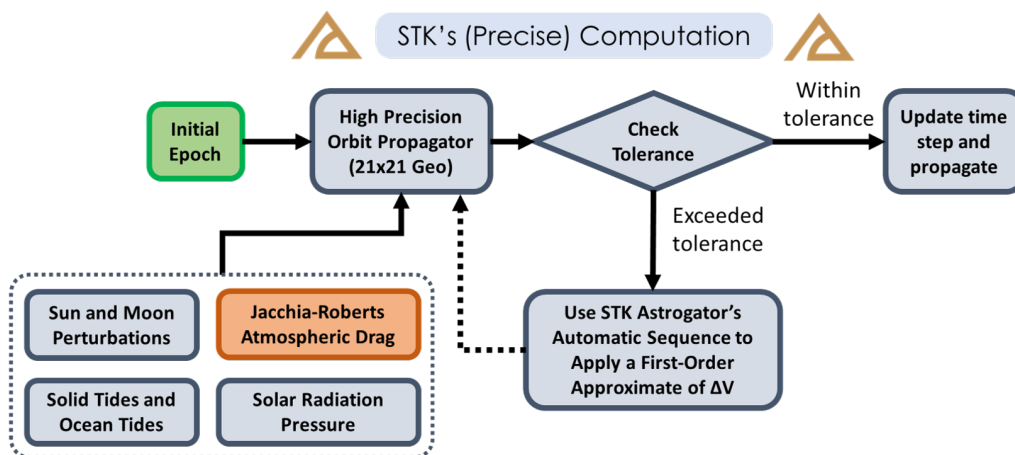
Traditional orbit maintenance simulations can be cumbersome and tedious to do on STK, GMAT, FreeFlyer, or even on your own code. Thus, ORBITM simplifies this entire drag-simulation process - in a fast and automated way that keeps up with rapidly changing iterations, without over-burdening the mission planner with repeated mission life-time simulations.

There are two modes of simulation available in ORBITM.

The first, is a fast-compute orbit decay and maintenance model considering primarily drag effects using the U.S. Standard Atmosphere 1976 model (**Sam's Simulator**).

Sam's (Fast) Computation

Initial Epoch → Two-Body Keplerian Scalar Radius Solver → Check Tolerance

Within tolerance → Update time step and propagate

Exceeded tolerance → Calculate for a Hohmann transfer via first order Taylor expansion of the Vis-Visa equation

Apply Orbit Decay Force from Sam's Model

US Standard Atmosphere 1976 Exponential Model

Apply ΔV

The second is a more accurately modelled but resource intensive mode that interfaces with and automates STK Astrogator to perform a high-precision orbit decay and maintenance simulation. The orbit model in STK uses a 21 by 21 (degree by order) geopotential and computes drag using a Jacchia-Roberts Atmospheric Density model.

STK's (Precise) Computation

Initial Epoch → High Precision Orbit Propagator (21x21 Geo) → Check Tolerance

Within tolerance → Update time step and propagate

Exceeded tolerance → Use STK Astrogator's Automatic Sequence to Apply a First-Order Approximate of ΔV

Sun and Moon Perturbations

Jacchia-Roberts Atmospheric Drag

Solid Tides and Ocean Tides

Solar Radiation Pressure

However, in the second mode, you would need a valid STK Astrogator and STK Integration license installed with STK 10 or 11, for the interfacing to work.

Note: ORBITM was designed for low Earth orbits. Thus, the working range of altitudes, before loss of atmospheric density accuracies, should be bounded within 86 - 1,000 km.

Warning: ORBITM was **not designed for high precision**, and thus the life-time estimation **cannot be used for re-entry and descent** positioning estimations.
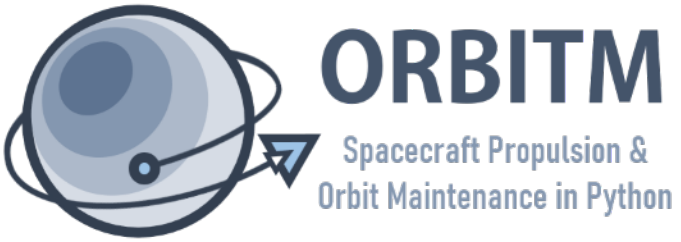
INSTALLATION

First, find the ORBITM GitHub repository in this GitHub link, and download it. Alternatively, if you have Git installed, you can open Command Prompt or your Git Bash, enter the directory of your choice, and type:
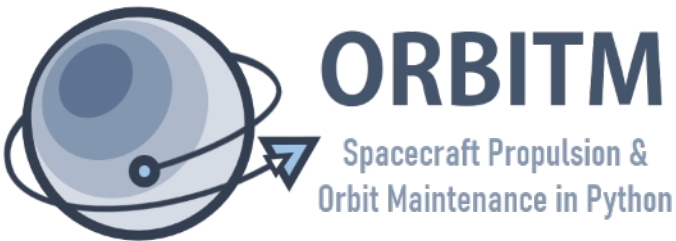
```
git clone https://github.com/sammmlow/ORBITM.git
```

That's it! No further setup is needed.

Note: Packages Required: numpy, matplotlib, tkinter, PIL, os, math, datetime, comtypes (for STK)

Next, we will need a Python interpreter (or an IDE if you'd like) to open up the main file **orbitm.py**.

FIRST STEPS

ORBITM can be launched simply by running the Python file **orbitm.py** in the main directory (equivalent to the directory on the master branch on ORBITM's GitHub). You should see a GUI that looks like the one below, pop up:
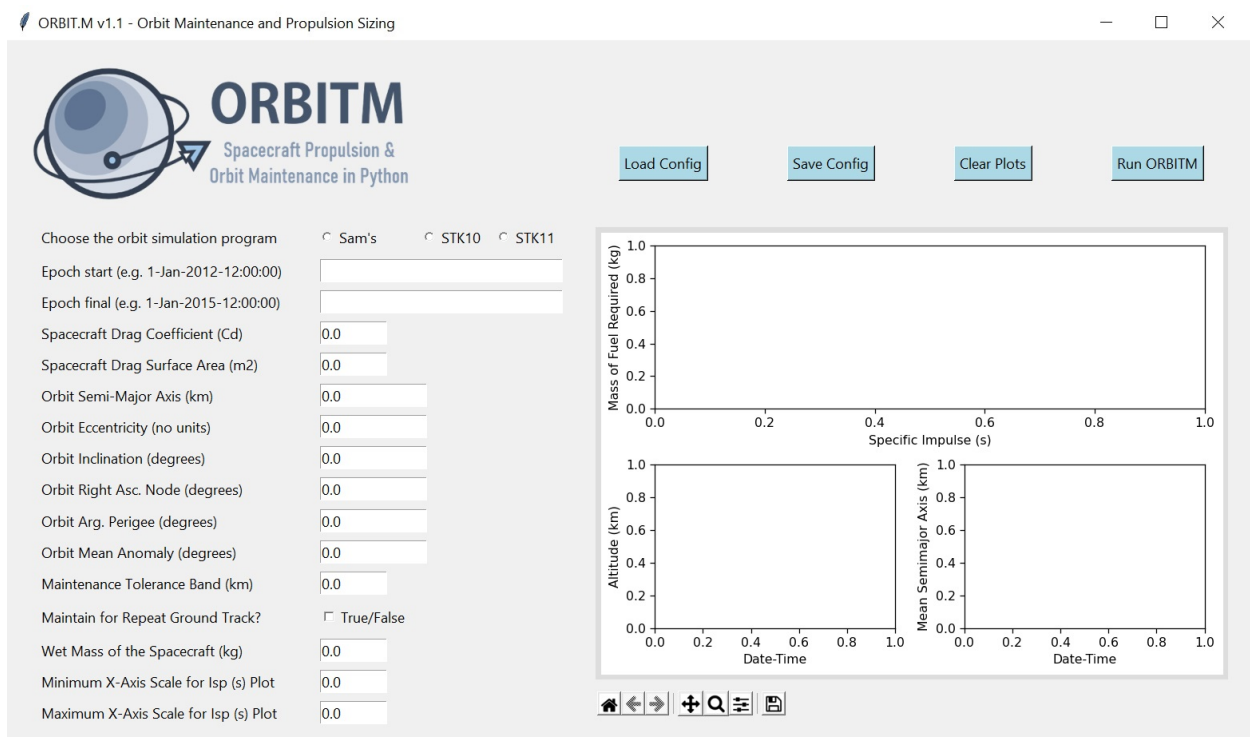

Fig. 1: Figure 2.1 - ORBITM Graphical User Interface

As a first step, you can simply load the default input parameters by clicking "Load Config". The parameters are actually saved in the **config.txt** file. Configuration parameters should be loaded through the GUI, and not manually updated through the **config.txt** file since then the software can't check for string formatting errors (e.g. a negative drag coefficient that was typed into the config file manually and by accident would crash the program).

> **Warning:** An orbit eccentricity of exactly zero is not allowed, to prevent potential divide-by-zero issues during calculations. For near-perfectly circular orbits, set an arbitrarily small eccentricity number (e.g., 0.00001).

Second, you can choose which lifetime and maintenance simulator you would like to use. By default, the native **"Sam's Simulator"** will be selected. If STK is selected, remember that **a valid STK license for STK Integration and STK Astrogator is required.**

Third, proceed to set the remainder of your orbit and spacecraft parameters. These parameters are self-explanatory. Some parameters are specific to ORBITM and are defined below:

> **Maintenance Tolerance Band**
>> Threshold altitude that triggers an orbit raise (km).

> **Frozen Repeat Ground Track**
>> Option to consider frozen repeat orbit maintenance.

Note: The altitude value used as the reference with the **Maintenance Tolerance Band** is defined as the geodetic altitude. In **"Sam's Simulator"** mode, this altitude is the Keplerian semi-major axis minus Earth's equatorial radius. However, STK-mode accounts for the oblateness of the Earth, and the presence of other perturbing forces. Thus, in STK mode, the Earth-centered altitude often oscillates with the latitude and with time, even for a "circular orbit".

In order to prevent triggering a manoeuvre on the wrong rising or falling edge of the altitude profile, the STK mode triggers manoeuvres based on the **Kozai-Izsak Mean Semi-Major Axis** rather than the nominal altitude. It is acknowledged that this changes the definition for the "threshold altitude", especially for highly elliptical orbits running in STK.

If the option for a **Frozen Repeat Ground Track** was selected, then the orbit raise will happen from the negative threshold below its initial altitude to a positive threshold above the initial altitude, and the thrusts will be done in perigee-apogee pairs. This minimises secular perturbations to the orbit mean eccentricity and the orbit nodal period. For example, if the **Frozen Repeat Ground Track** option was selected, with an initial altitude of 500km, and a threshold of 5km, the orbit raise will be triggered at 495km, and raised to 505km in two thrust pairs. Else, in the non-frozen repeat orbit maintenance case, the orbit raise will be triggered at 495km, and raised back to the original 500km.

Finally, before running the simulation (**Run ORBITM**), check the **"thrusters.txt"** file on the main ORBITM directory, and add thruster specifications that you wish to size your missions against. Some thrusters are written inside this shortlist as an example. The purpose of this shortlist is to provide the program with the thruster specifications, so that the mission planner can graphically compare its Isp and fuel capacity to the suitability of the mission (as an output of the program).

Now, you can run ORBITM. If you are a licensed STK user and you would like to use ORBITM to automate an orbit maintenance simulation in STK for you, make sure there are **no** running instances of STK before you run ORBITM in the STK 10 or 11 mode.

Note: ORBITM considers only atmospheric drag and not albedo and radiation effects.
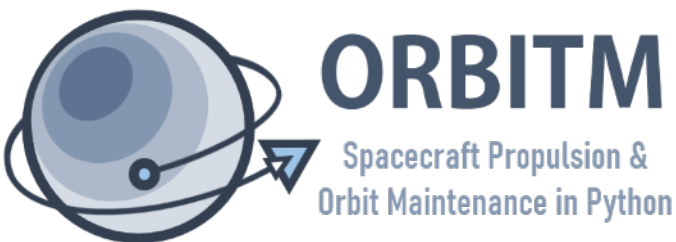
INTEPRETING RESULTS

After a successful ORBITM run, you would expect two output text files in the main directory, and three Matplotlib figures.

The first text file **"output_summary.txt"** is a summary of the lifetime (i.e. how long the spacecraft can last **without** orbit maintenance).

```
Lifetime decay is estimated to be on 13 May 2027 16:54:31.252 after 29940 orbits.
The lifetime is 5.4 years.
The total impulse needed: 2233.4388819714236
The total Delta-V (m/s) needed is 13.137875776302492
```

The second text file **"output_manoeuvres.txt"** is an estimate of the orbit maintenance manoeuvre schedule.

```
1 Maintain.Hohmann 2022-03-01 11:40:00 2.1896281088566925
2 Maintain.Hohmann 2022-04-29 12:07:49 2.1896767040214486
3 Maintain.Hohmann 2022-06-27 12:35:38 2.1896387651453866
4 Maintain.Hohmann 2022-08-25 13:03:27 2.189679576264574
5 Maintain.Hohmann 2022-10-23 13:31:16 2.1896504433114723
6 Maintain.Hohmann 2022-12-21 13:59:05 2.1896153519034436
```

To interpret the graphical results, let us run a simple orbit maintenance scenario for a satellite in a 550km altitude circular orbit at 45 degrees inclination.

The plots in blue correspond to **"Sam's Simulator"**. The plots in orange correspond to using **STK 10** Astrogator, with a high precision orbit propagator, and it uses Astrogator's Automatic Sequences feature to boost the orbit up whenever the altitude of the spacecraft crosses the minimum of the maintenance tolerance band. In both orbit maintenance modes, the V values were computed using a first order Taylor expansion to the Vis-Visa equation.

The plot most relevant to mission planning that you would need is the top-most plot - the profile of the mission lifetime (in terms of total V requirements) against the thruster specifications (bar charts). The height of the bar chart can be compared against the desired fuel mass needed, at the thruster's specific impulse values (Isp, units in seconds). In other words, for each thruster's Isp value, if the height of the bar chart (which represents the max fuel mass for that thruster, at that Isp) exceeds the fuel mass requirements of your mission at that particular Isp, then that thruster can satisfy your mission lifetime requirements.

Note: The native **"Sam's Simulator"** uses the US Standard Atmosphere 1976 as the atmospheric density model as a look-up table of coefficients in an exponential function (for speed), while the **STK Mode** uses the Jacchia Roberts atmospheric density model. Also, the native mode uses an approximated orbit decay model while **STK Mode** performs a full orbit propagation for all 6 states. Thus, running ORBITM with STK makes computation very much slower, and the V values in both modes would differ, although in most cases they are close (and definitely in the same order of magnitude).
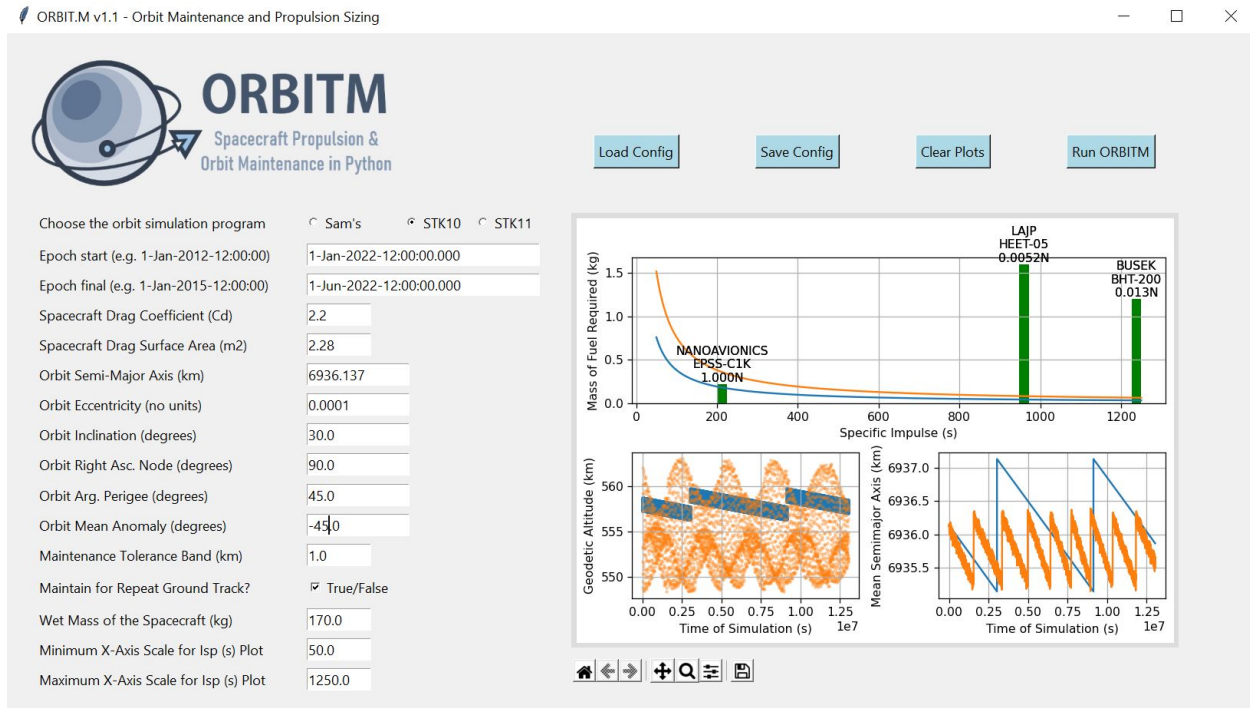

Fig. 1: Figure 3.1 - Screenshot after running ORBITM, for a 550km orbit with a tolerance band of 1km

ORBIT MAINTENANCE MODEL

This section explains the algorithm behind the fast but approximated orbit maintenance simulations (**"Sam's Simulator"**).

The key idea is that one does not need to obtain all six parameters of the spacecraft during orbit propagation! Where a sufficiently accurate drag model is available, one really just needs the radial magnitude of the spacecraft in order to characterise the orbital energy loss due to the atmospheric density. This simplifies the number of calculations needed. In this section, the derivation of the decay model will be shared.

The goal of this exercise is to derive how the rate of change of a satellite's radial distance relates to the drag forces at play. We begin with first principles - an expression on the orbit state energy - and find its derivative with respect to time and altitude.

Starting from some high school physics... let's see how we can characterise energy loss in orbit...

Total Orbital Energy

$$U = KE + GPE = \left(-\frac{GM_E m}{2R} \right)$$

Let's differentiate with R

Radial Derivative

$$\frac{dU}{dR} = \frac{GM_E m}{2R^2}$$

Let's revisit first principles of "energy and work done"

Time Derivative

$$\frac{dU}{dt} = F\frac{ds}{dt} = F \cdot R \cdot \frac{d\theta}{dt} = F \cdot R \sqrt{\frac{GM_E}{a^3}}$$
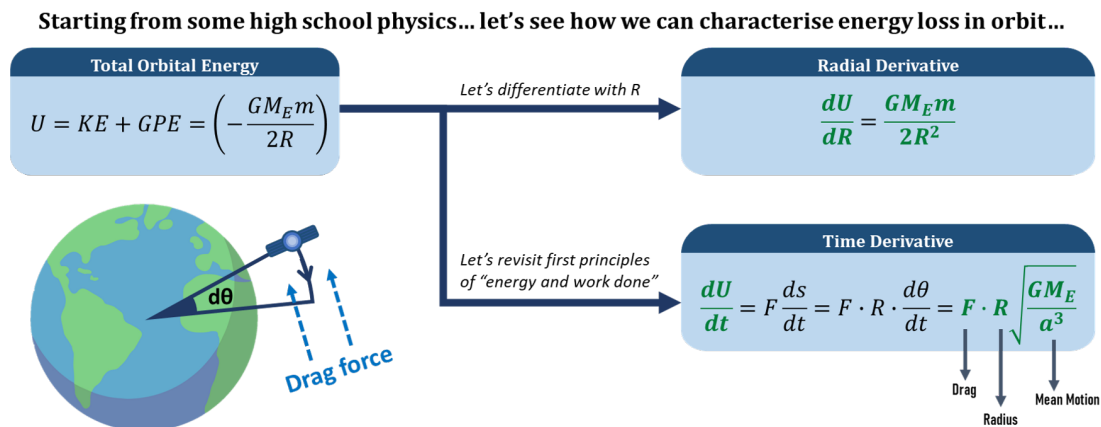
Drag · Radius · Mean Motion

Fig. 1: Figure 4.1 - Derivative of orbit state energy with respect to time and altitude.

The rate of change of work done by drag forces can be multiplied with the inverse of the altitude derivative of orbital energy, thereby removing the energy differential, **dU**, as the common term. Now, we have an expression for the rate of altitude change! We simply need to input the drag force **F**, and the radial distance of the satellite **R**.

The radial distance **R** can be solved for through Kepler's Equation in the Python file **orbitm_func_kepler.py** and using the input orbit parameters for a purely Keplerian orbit.

The drag force **F**, or equivalently the drag acceleration , is computed via the standard drag equation, as a result of the satellite area-to-mass parameters (as set by the user), and the atmopsheric density value at radial distance **R**. The density is sourced from **orbitm_func_atmos.py**. This file is basically the U.S. Standard Atmosphere 1976, represented as a look-up table of coefficients for an exponential atmospheric model.

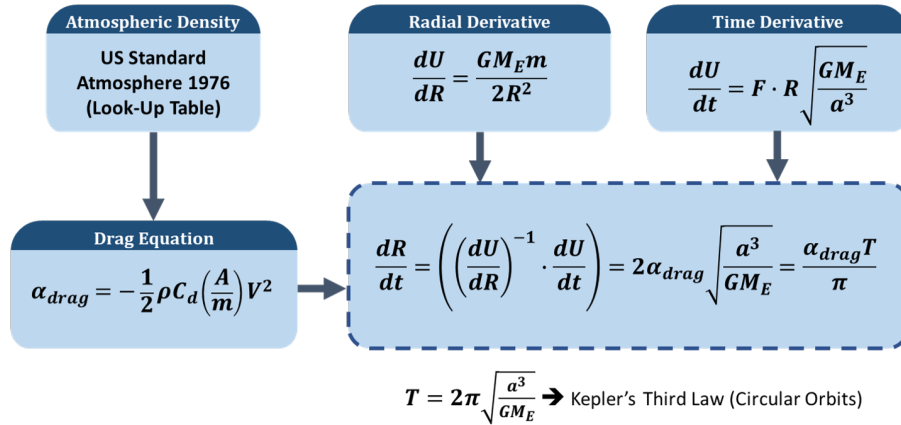Simplifying the altitude decay rate model, we arrive at the elegant statement in 4.3:

Fig. 2: Figure 4.2 - Deriving the rate of change of the satellite altitude.

$$\frac{dR}{dt} = \frac{\alpha_{drag}T}{\pi}$$

Fig. 3: **Equation 4.3 - Decay rate = product of drag deceleration and Keplerian period, divided by Pi**

This is powerful, because it proves that the rate at which the orbit decays can be solved in closed-form without any full orbit propagation of orbit states! This is the reason why orbit maintenance computations are extremely fast on ORBITM.

As the decay rate is also small, the total decay appears almost linear within the time frame of a single LEO orbital period (~95 minutes). Thus, a larger step size can also be used, without compromising too much on accuracy.

Consequently, the ORBITM program simply loops through the scenario time, and computes the decay in each time step until the spacecraft descends below the **Maintenance Tolerance Band**, which then triggers an orbit raise. This repeats until the end of the simulation.
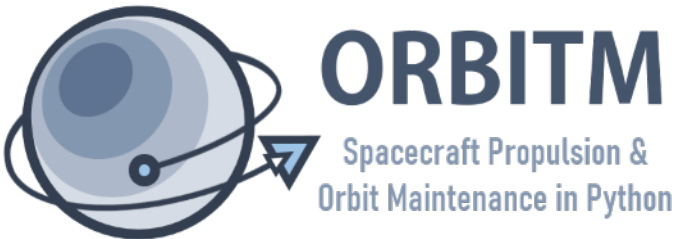
API REFERENCE

If ORBITM has made your life somewhat easier, please do cite the work behind it!

Low, S. Y. W., &; Chia, Y. X. (2018). "Assessment of Orbit Maintenance Strategies for Small Satellites", 32nd Annual AIAA/USU Conference on Small Satellites, Logan, Utah, Utah State University, USA.

For bugs, raise the issues in the GitHub repository.

For collaborations, reach out to me: sammmlow@gmail.com (Samuel Y. W. Low)

The project is licensed under the MIT license.