Chemical vs. Electric Propulsion

A Self-Contained Python Package

ASTE 404: Mini-Project
Professor: Daniel Depew
Fall 2025
Dylan Pena Perez

Table of Contents

1. Project Overview

This mini project centers around building a self-contained numerical tool that compares chemical (impulsive) and electric propulsion (EP) (low thrust) for orbital transfers at different altitudes from the Earth. Maing motivator for this project is based on a fundamental question:

> *Given a set of constraints for a specific mission that dictate orbit altitude and inclination change as well as transfer time, when is an EP or a Chemical propulsion system appropriate?*

Electric propulsion is commonly used for orbit raising, attitude control and station-keeping. As learned in ASTE 475, it is extremely efficient, requiring less delta V than chemical propulsion for the same maneuvers; but this comes at the cost of longer maneuver durations. Using ODE integration, convergence proof, and event detection, this project seeks to answer that fundamental question at a simplified level.

This package can be found in the GitHub repository:
https://github.com/Dylnpn/OrbitAnalysis.git

This package, written in Python, contains the following:

- Computation of an analytical chemical Hohmann + plane change orbit transfer
- Numerical simulation of a low-thrust EP transfer
- Evaluation of feasibility under time and thruster constraints
- Trade sweeps and convergence studies
- Command line interface (CLI) for user

2. Numerical Methods and Equations

Note that the simulations were performed assuming two-body problem for Earth-spacecraft interactions with no J2 perturbations due to Earth's oblateness. Drag is also ignored as well as solar density flux. Note this is an extremely simplified model of spacecraft orbital mechanics.

2.1 Chemical Propulsion Model (Impulsive)

The chemical baseline for this project uses a basic Hohmann transfer between circular orbits of radii $r_1$and r_2. The following equations model this type of chemical orbital transfer:

Acceleration expressed by:

$$a_t = \frac{r_1 + r_2}{2}$$

And delta Vs expressed by:

$$\Delta v_1 = \sqrt{\frac{\mu}{r_1}}\left(\sqrt{\frac{2r_2}{r_1 + r_2}} - 1\right) \qquad \Delta v_2 = \sqrt{\frac{\mu}{r_2}}\left(1 - \sqrt{\frac{2r_1}{r_1 + r_2}}\right)$$

The inclination change is represented through:

$$\Delta v_{plane} = 2vsin\left(\frac{\Delta i}{2}\right)$$

Using this model, the solution is of closed-form, enabling a facilitated verification baseline to compare to the EP system.

For this model, keep in mind that the following assumptions are made:

- Impulsive burns
- Circular orbits
- Burn location for combined plane change is not optimized

2.2 Electric Propulsion Model (Low Thrust)

EP was solved using an ODE style integrator, with the following propagated states:

$$y(t) = [a(t), i(t), m(t)]$$

Where $a(t)$ is semimajor $i(t)$ is inclination, and $m(t)$ is mass, all as a function of time. These quantities are governed by their change propagated through time in the following manner:

$$\frac{da}{dt} = \frac{2a^{\frac{3}{2}}}{\sqrt{\mu}} a_t$$

$$\frac{di}{dt} = \frac{a_n}{\sqrt{\mu a}}$$

$$\frac{dm}{dt} = -\frac{T}{g_0 I_{sp}}$$

Here, the thrust, T, is assumed to be constant with simplified vector steering.

These equations allow us to mimic general low thrust behavior as per the two-body problem, producing smooth dynamics that can be integrated for long periods of time with a fixed-step integration numerical solver. However, due to its simplified physical behavior, the model's accuracy greatly depends on the time step size, only providing accurate results for low time steps. Additionally, no logic for optimal control was implemented.

2.3 Fourth-Order Runge-Kutta (RK4) Numerical Solver

The EP system of ordinary differential equations is integrated using a fixed-step RK4 method.

$$y_{k+1} = y_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

In the python script, we compute the weighted average of four slope estimates which is used to approximate the state derivative over the time step. This is much more accurate for orbital mechanic applications compared to the Euler method, which only uses one slope approximation).

In RK4, the derivative is sampled at 4 points (4th-order)

- K1: slope at the start of step
- K2 : slope at the midpoint using k1
- K3: slope at the midpoint using k2
- K4: slope are the end of the step

These slopes are interpreted in the following manner :

$$k_1 = f(x_1)$$

$$k_2 = f\left(x_k + \frac{\Delta t}{2}k_1\right)$$

$$k_3 = f\left(x_k + \frac{\Delta t}{2}k_2\right)$$

$$k_4 = f(x_k + \Delta t k_3)$$

Each k is treated as a vector, which is the same size as the state vector defined in the simulation.

RK4 uses Taylor series to create a weighted average combination for the slope:

$$slope = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Using weights 1, 2, 2, 1, lower-order truncation errors are cancelled out, and provides fourth-order accuracy.

In the code, this is done for every component of the state vector. This slope is used in the following manner:

$$y_{n+1} = y_n + h * slope$$

This essentially gives us the best estimate of the derivate over the step. This is imperative to use in EP models because EP equations are on linear, thrust, mass and orbital elements are coupled, and no closed form solutions exist. Using the weighted slope method from RK4, we are able to capture curvature in dynamics while observing convergence of solutions.

3. Implementation

The code is organized in a Python package as follows:

```
orbitAnalysis/
|
|---chem.py # chemical (impulsive) simulation
|---cli.py # Facilitated user interface
|---constants.py # constants used
|---ep.py # EP numerical simulation
|---rungeKutta4.py # RK4 integration utilities
|---trade.py # compares prop systems and makes rec.
|
|---examples/
| |---convergence.py # plots convergence
| |---trade_sweep.py # plots trade analysis
|
|---pyproject.toml
|---README.md\
```

3.1 Installation

Download the source code or clone the repo locally. In the project root directory, open a terminal and create/activate a fresh Conda environment (or reuse an existing one):

```
bash
conda create-n orbitTool python=3.14
conda activate orbitTool
pip install -e .
```

3.2 Quickstart

To run the trade study: Replace all "<#>" with preferred input:

```
orbitAnalysis --h1-km <#> --h2-km <#> --inc1-deg <#> --inc2-deg <#> --t-days
<#> --m0-kg <#> --ep-thrust <#> --ep-isp <#> --dt <#>
```

```
Example 1:
orbitAnalysis --h1-km 200 --h2-km 300 --inc1-deg 30 --inc2-deg 32 --t-days 40 --
m0-kg 1000 --ep-thrust 40 --ep-isp 1600 --dt 1
--> Should return EP YES Feasible
```

```
Example 2:
orbitAnalysis --h1-km 200 --h2-km 700 --inc1-deg 30 --inc2-deg 40 --t-days 4 --
m0-kg 1000 --ep-thrust 2 --ep-isp 160 --dt 1
--> Should return chem NOT EP
```

To run the convergence plot use the following. Note that you must go into "convergence.py" if you wiish to change the step size. The output should be two plots showing convergence.

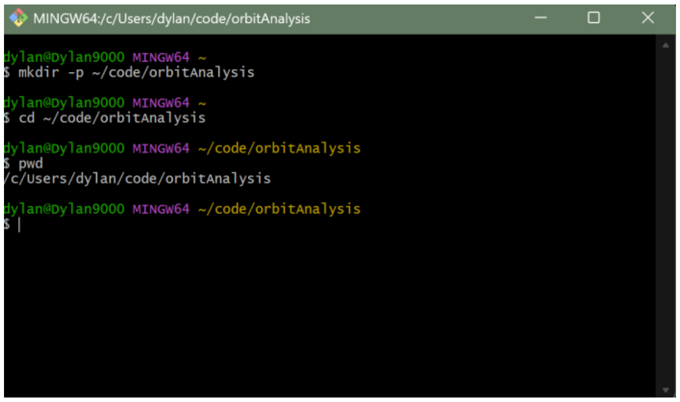
```
python examples/convergence.py
```

To run the sweep trade analysis plot, run the following line. Note that in order to change the analysis trade parameters, you must go into "trade_sweep.py". The output should show EP prop used vs transfer time.

```
python examples/trade_sweep.py
```

4. Activity Log

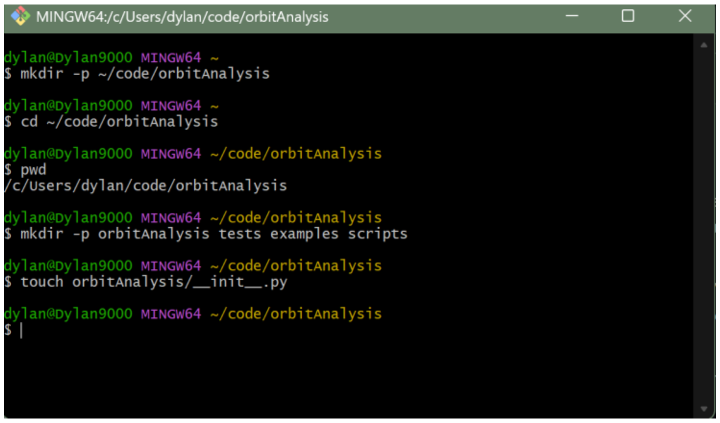
12/15 - 12:55 AM


```
MINGW64:/c/Users/dylan/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~
$ mkdir -p ~/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~
$ cd ~/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ pwd
/c/Users/dylan/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ |
```

Made project folder

-

12/15 - 12:58 AM

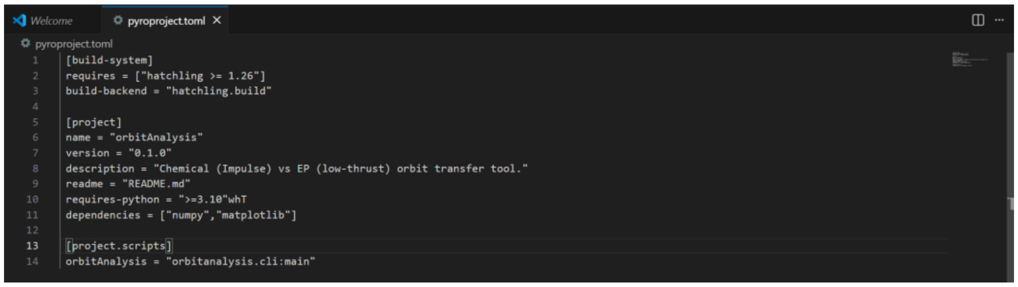


```
dylan@Dylan9000 MINGW64 ~
$ mkdir -p ~/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~
$ cd ~/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ pwd
/c/Users/dylan/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ mkdir -p orbitAnalysis tests examples scripts

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ touch orbitAnalysis/__init__.py

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ |
```

Created package structure

-

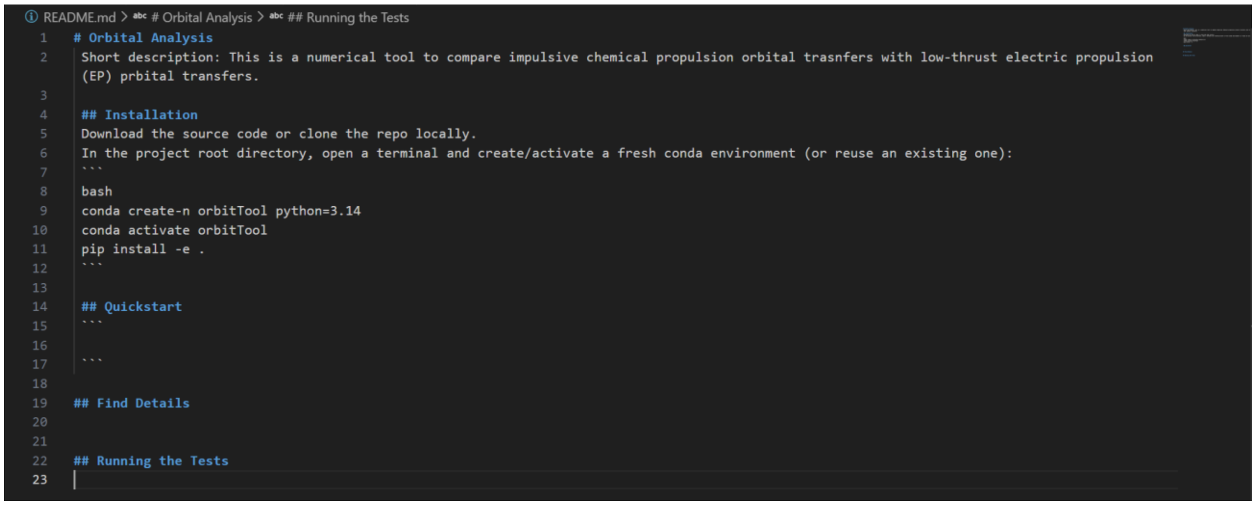
12/15 - 1:09 AM



```
[build-system]
requires = ["hatchling >= 1.26"]
build-backend = "hatchling.build"

[project]
name = "orbitAnalysis"
version = "0.1.0"
description = "Chemical (Impulse) vs EP (low-thrust) orbit transfer tool."
readme = "README.md"
requires-python = ">=3.10"whT
dependencies = ["numpy","matplotlib"]

[project.scripts]
orbitAnalysis = "orbitanalysis.cli:main"
```

Created pyproject.toml file

12/14 - 1:13 AM



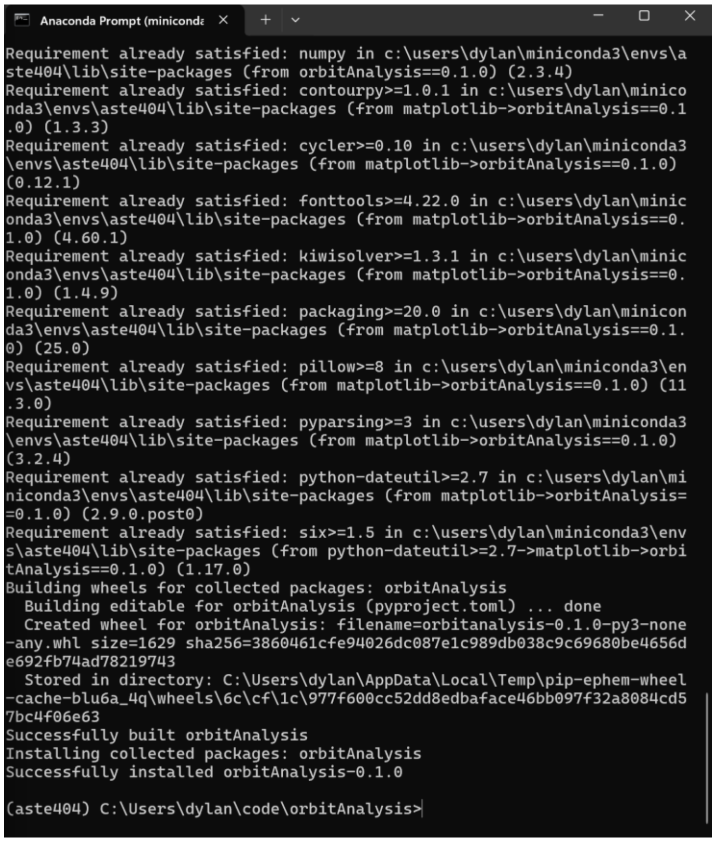
```
# Orbital Analysis
Short description: This is a numerical tool to compare impulsive chemical propulsion orbital trasnfers with low-thrust electric propulsion
(EP) prbital transfers.

## Installation
Download the source code or clone the repo locally.
In the project root directory, open a terminal and create/activate a fresh conda environment (or reuse an existing one):
```
bash
conda create-n orbitTool python=3.14
conda activate orbitTool
pip install -e .
```

## Quickstart
```

```

## Find Details


## Running the Tests

```

Created README.md skeleton

-

12/15 - 1:22 AM

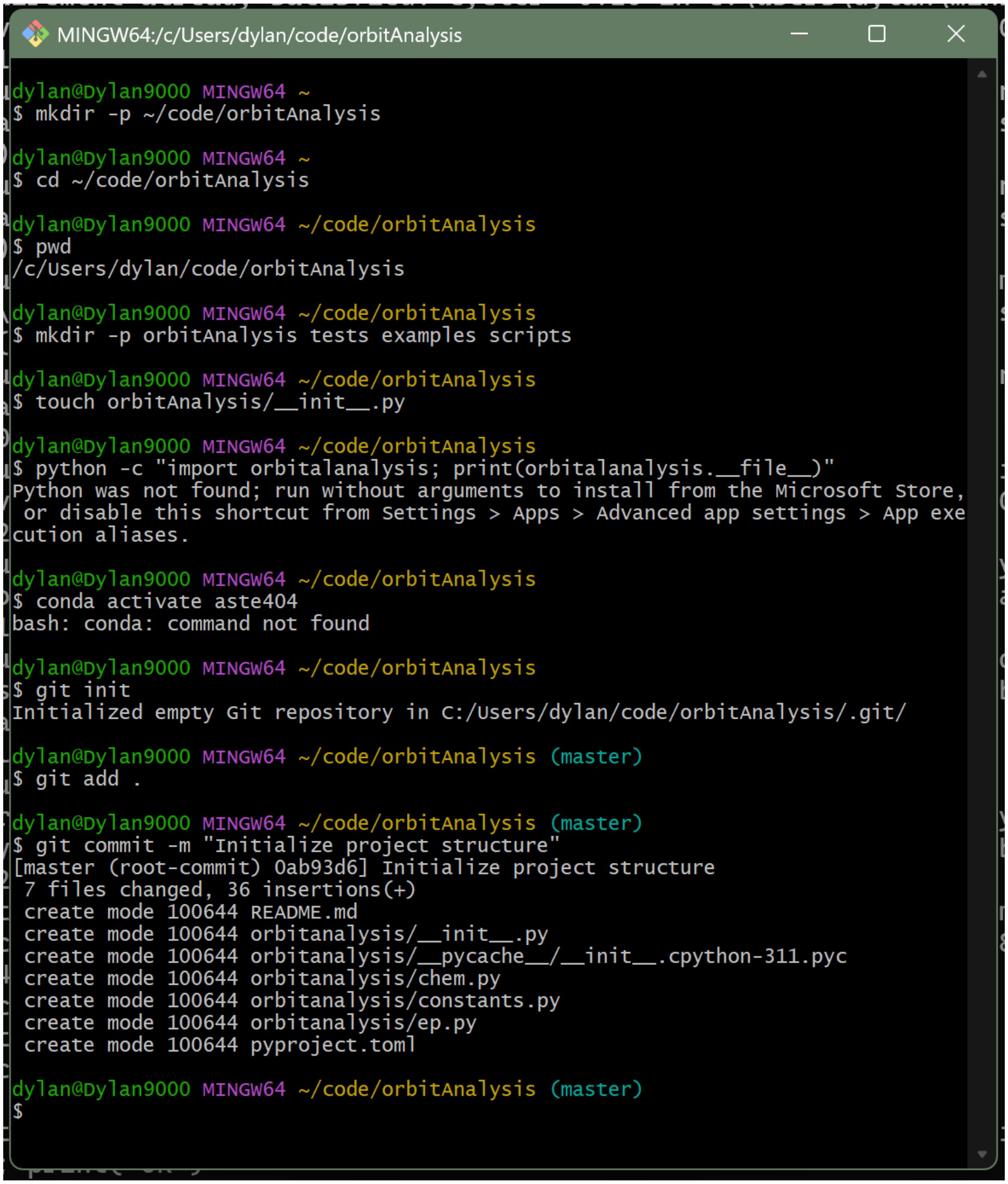


```
Requirement already satisfied: numpy in c:\users\dylan\miniconda3\envs\a
ste404\lib\site-packages (from orbitAnalysis==0.1.0) (2.3.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\dylan\minico
nda3\envs\aste404\lib\site-packages (from matplotlib->orbitAnalysis==0.1
.0) (1.3.3)
Requirement already satisfied: cycler>=0.10 in c:\users\dylan\miniconda3
\envs\aste404\lib\site-packages (from matplotlib->orbitAnalysis==0.1.0)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\dylan\minic
onda3\envs\aste404\lib\site-packages (from matplotlib->orbitAnalysis==0.
1.0) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\dylan\minic
onda3\envs\aste404\lib\site-packages (from matplotlib->orbitAnalysis==0.
1.0) (1.4.9)
Requirement already satisfied: packaging>=20.0 in c:\users\dylan\minicon
da3\envs\aste404\lib\site-packages (from matplotlib->orbitAnalysis==0.1.
0) (25.0)
Requirement already satisfied: pillow>=8 in c:\users\dylan\miniconda3\en
vs\aste404\lib\site-packages (from matplotlib->orbitAnalysis==0.1.0) (11
.3.0)
Requirement already satisfied: pyparsing>=3 in c:\users\dylan\miniconda3
\envs\aste404\lib\site-packages (from matplotlib->orbitAnalysis==0.1.0)
(3.2.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dylan\mi
niconda3\envs\aste404\lib\site-packages (from matplotlib->orbitAnalysis=
=0.1.0) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\dylan\miniconda3\env
s\aste404\lib\site-packages (from python-dateutil>=2.7->matplotlib->orbi
tAnalysis==0.1.0) (1.17.0)
Building wheels for collected packages: orbitAnalysis
  Building editable for orbitAnalysis (pyproject.toml) ... done
  Created wheel for orbitAnalysis: filename=orbitanalysis-0.1.0-py3-none
-any.whl size=1629 sha256=3860461cfe94026dc087e1c989db038c9c69680be4656d
e692fb74ad78219743
  Stored in directory: C:\Users\dylan\AppData\Local\Temp\pip-ephem-wheel
-cache-blu6a_4q\wheels\6c\cf\1c\977f600cc52dd8edbaface46bb097f32a8084cd5
7bc4f06e63
Successfully built orbitAnalysis
Installing collected packages: orbitAnalysis
Successfully installed orbitAnalysis-0.1.0

(aste404) C:\Users\dylan\code\orbitAnalysis>
```

Successfully installed my package

-

12/15 - 1:25 AM



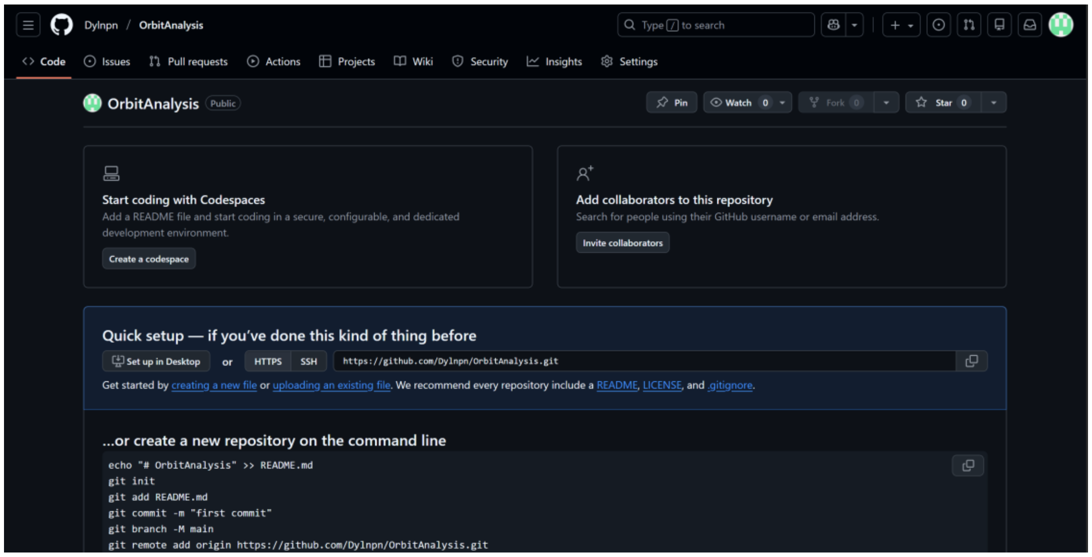
MINGW64:/c/Users/dylan/code/orbitAnalysis

```
dylan@Dylan9000 MINGW64 ~
$ mkdir -p ~/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~
$ cd ~/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ pwd
/c/Users/dylan/code/orbitAnalysis

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ mkdir -p orbitAnalysis tests examples scripts

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ touch orbitAnalysis/__init__.py

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ python -c "import orbitalanalysis; print(orbitalanalysis.__file__)"
Python was not found; run without arguments to install from the Microsoft Store,
 or disable this shortcut from Settings > Apps > Advanced app settings > App exe
cution aliases.

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ conda activate aste404
bash: conda: command not found

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis
$ git init
Initialized empty Git repository in C:/Users/dylan/code/orbitAnalysis/.git/

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (master)
$ git add .

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (master)
$ git commit -m "Initialize project structure"
[master (root-commit) 0ab93d6] Initialize project structure
 7 files changed, 36 insertions(+)
 create mode 100644 README.md
 create mode 100644 orbitanalysis/__init__.py
 create mode 100644 orbitanalysis/__pycache__/__init__.cpython-311.pyc
 create mode 100644 orbitanalysis/chem.py
 create mode 100644 orbitanalysis/constants.py
 create mode 100644 orbitanalysis/ep.py
 create mode 100644 pyproject.toml

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (master)
$
```

initialized git locally

-

12/15 - 1:26 AM



Quick setup — if you've done this kind of thing before

https://github.com/Dylnpn/OrbitAnalysis.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

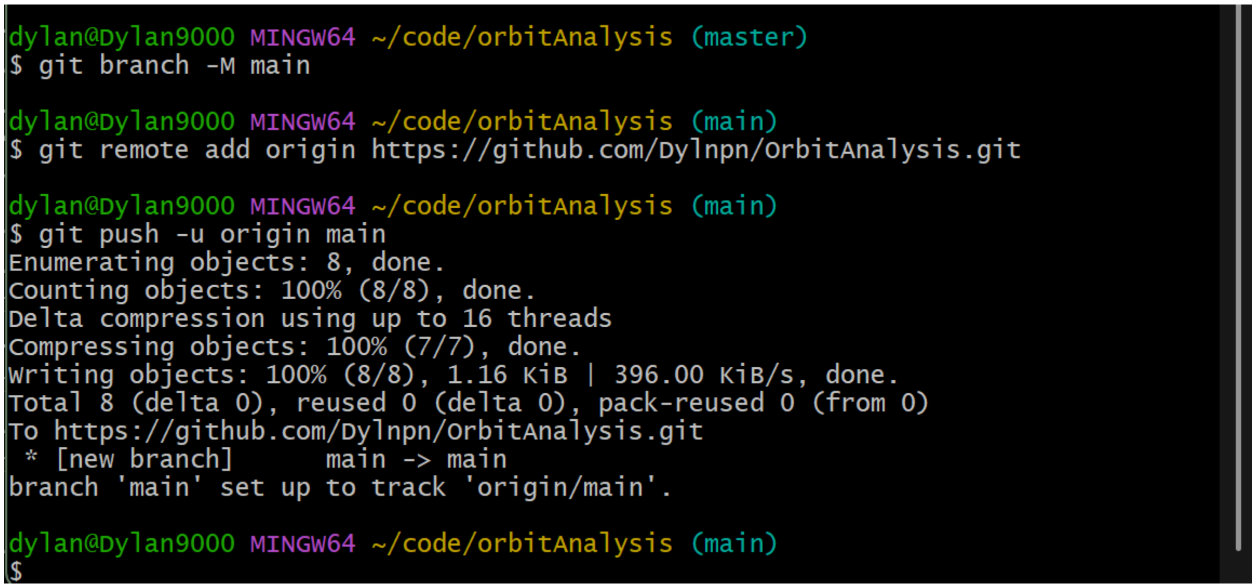
```
echo "# OrbitAnalysis" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Dylnpn/OrbitAnalysis.git
```

Created github repository!

-

12/15 - 1:28 AM

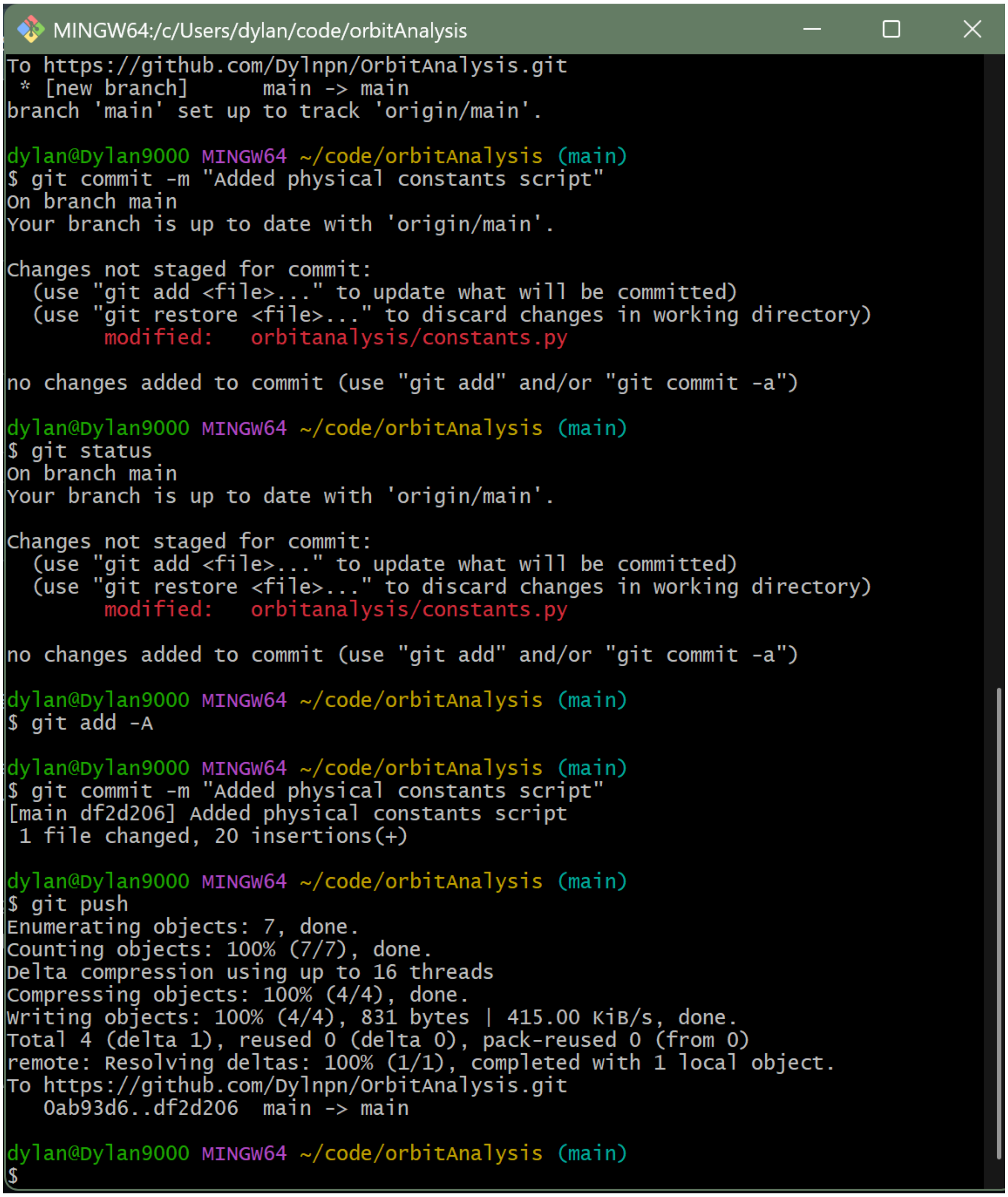
Made initial git push and commit



```
dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (master)
$ git branch -M main

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git remote add origin https://github.com/Dylnpn/OrbitAnalysis.git

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git push -u origin main
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 16 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 1.16 KiB | 396.00 KiB/s, done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Dylnpn/OrbitAnalysis.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$
```

-

12/15 - 11:39 AM

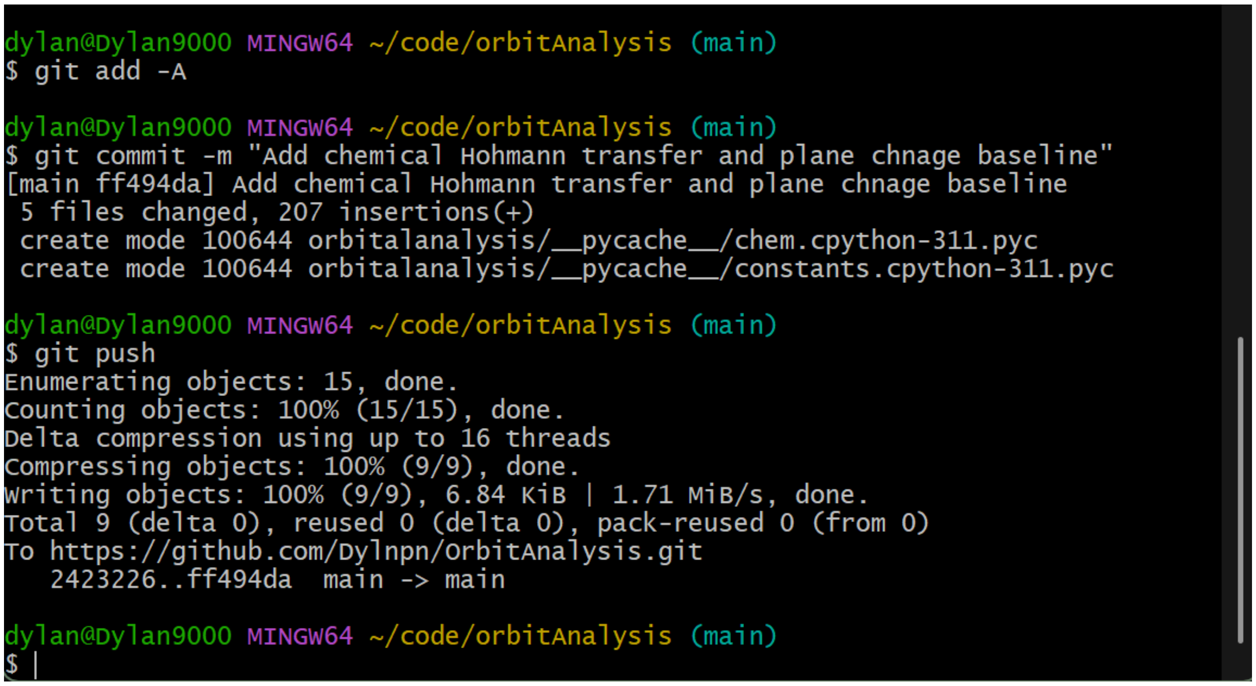


```
To https://github.com/Dylnpn/OrbitAnalysis.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git commit -m "Added physical constants script"
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   orbitanalysis/constants.py

no changes added to commit (use "git add" and/or "git commit -a")

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   orbitanalysis/constants.py

no changes added to commit (use "git add" and/or "git commit -a")

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git add -A

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git commit -m "Added physical constants script"
[main df2d206] Added physical constants script
 1 file changed, 20 insertions(+)

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 831 bytes | 415.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Dylnpn/OrbitAnalysis.git
   0ab93d6..df2d206  main -> main

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$
```

pushed constants.py file and committed

-

12/15 - 1:11 PM

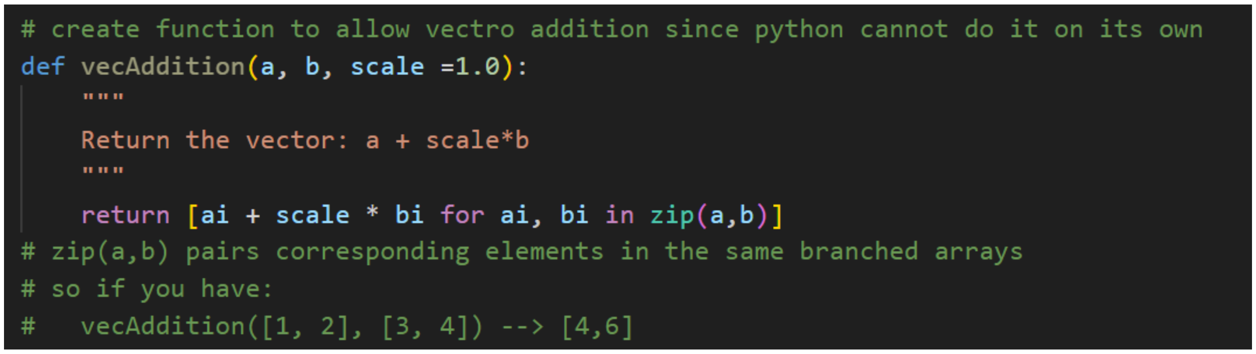


```
dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git add -A

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git commit -m "Add chemical Hohmann transfer and plane chnage baseline"
[main ff494da] Add chemical Hohmann transfer and plane chnage baseline
 5 files changed, 207 insertions(+)
 create mode 100644 orbitalanalysis/__pycache__/chem.cpython-311.pyc
 create mode 100644 orbitalanalysis/__pycache__/constants.cpython-311.pyc

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git push
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 16 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 6.84 KiB | 1.71 MiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Dylnpn/OrbitAnalysis.git
   2423226..ff494da  main -> main

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ |
```

Finished chem.py with initial orbital change baseline

-

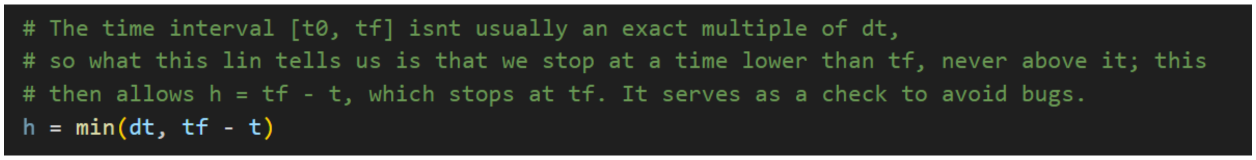
12/15 - 1:52 PM



```python
# create function to allow vectro addition since python cannot do it on its own
def vecAddition(a, b, scale =1.0):
    """
    Return the vector: a + scale*b
    """
    return [ai + scale * bi for ai, bi in zip(a,b)]
# zip(a,b) pairs corresponding elements in the same branched arrays
# so if you have:
#   vecAddition([1, 2], [3, 4]) --> [4,6]
```

The RK4 integrator is implemented for this project by using explicit vector arithmetic. This is done in order to keep track of all of the computations going one. The helper function “vecAddition” allow to evaluate the expressions of y(vector) + alpha*k(vector), which makes the steps in the integration easily verifiable.

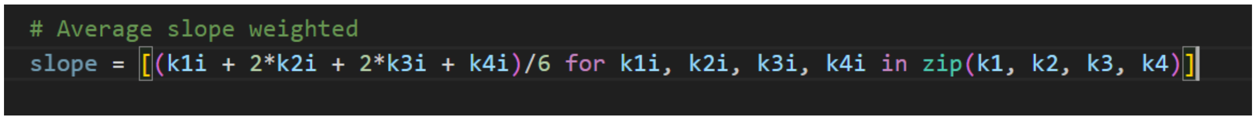
12/15 - 1:59 PM



```python
# The time interval [t0, tf] isnt usually an exact multiple of dt,
# so what this lin tells us is that we stop at a time lower than tf, never above it; this
# then allows h = tf - t, which stops at tf. It serves as a check to avoid bugs.
h = min(dt, tf - t)
```

This line avoids overshooting the final integration time, allow the RK4 integrator to dynamically reduce the final step size. This allows the program to terminate at the specified final time without effecting results.

12/ 15 - 2:04 PM



```python
# Average slope weighted
slope = [(k1i + 2*k2i + 2*k3i + k4i)/6 for k1i, k2i, k3i, k4i in zip(k1, k2, k3, k4)]
```

This line computes the weighted average of four slope estimates which is used to approximate the state derivative over the time step.

In RK4, the derivative is sampled at 4 points (4th-order)

- K1: slope at the start of step
- K2 : slope at the midpoint using k1
- K3: slope at the midpoint using k2
- K4: slope are the end of the step

Each k is treated as a vector, which is the same size as my state vector.

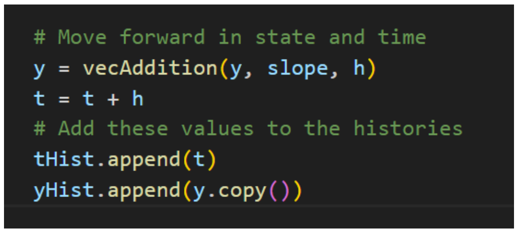
RK4 uses Taylor series to create a weighted average combination for the slope:

$$slope = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Using weights 1, 2, 2, 1, lower-order truncation errors are cancelled out, and provides fourth-order accuracy.

In the code, this is done for every component of the state vector.

This slope is used shortly after:



```python
# Move forward in state and time
y = vecAddition(y, slope, h)
t = t + h
# Add these values to the histories
tHist.append(t)
yHist.append(y.copy())
```

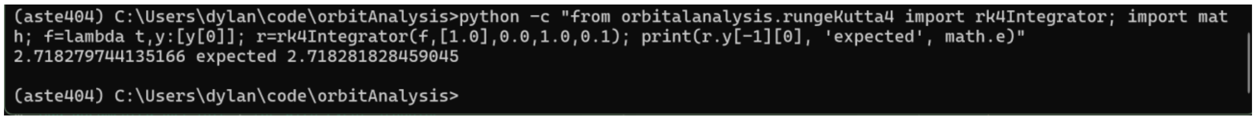
Which is essentially the RK4 update formula:

$$y_{n+1} = y_n + h * slope$$

This essentially gives us the best estimate of the derivate over the step. This is imperative to use in EP models because EP equations are on linear, thrust, mass and orbital elements are coupled, and no closed form solutions exist. Using the weighted slope method from RK4, we are able to capture curvature in dynamics while observing convergence of solutions.

12/15 - 2:23 PM

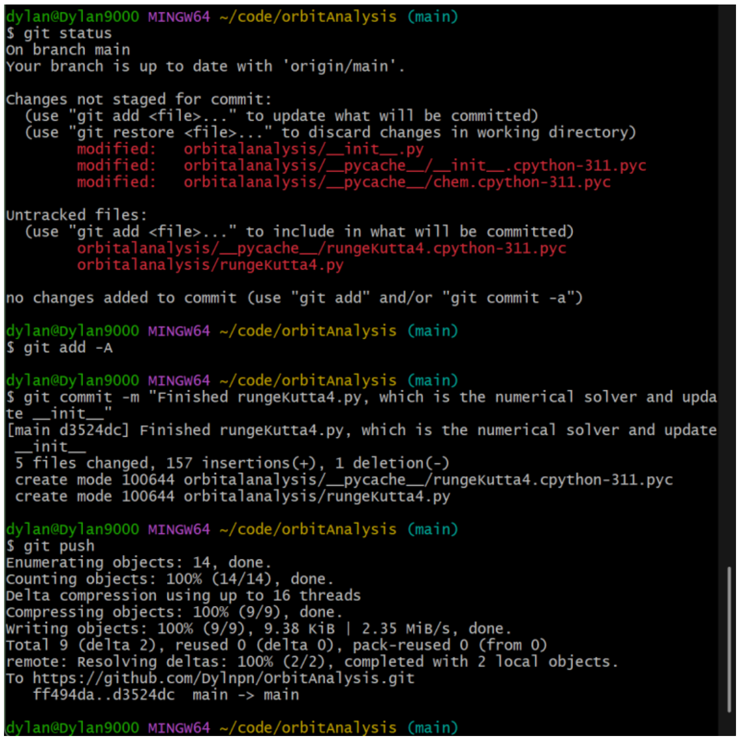
Finished the rungeKutta4.py script and ran a test to see if It was working:



```
(aste404) C:\Users\dylan\code\orbitAnalysis>python -c "from orbitalanalysis.rungeKutta4 import rk4Integrator; import mat
h; f=lambda t,y:[y[0]]; r=rk4Integrator(f,[1.0],0.0,1.0,0.1); print(r.y[-1][0], 'expected', math.e)"
2.718279744135166 expected 2.718281828459045

(aste404) C:\Users\dylan\code\orbitAnalysis>
```

This integrated dy/dt = y from 0 to 1, which outputted 2.718, the value of e. It works!

Committed and pushed my changes:

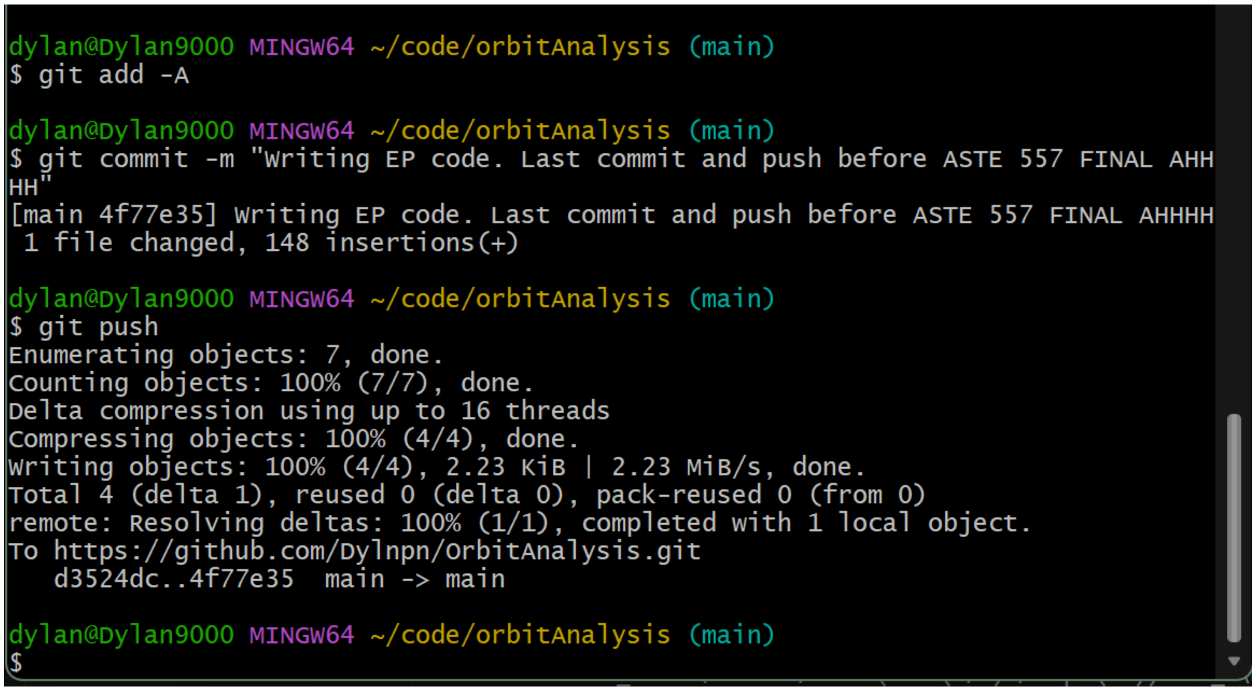


```
dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   orbitalanalysis/__init__.py
        modified:   orbitalanalysis/__pycache__/__init__.cpython-311.pyc
        modified:   orbitalanalysis/__pycache__/chem.cpython-311.pyc

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        orbitalanalysis/__pycache__/rungeKutta4.cpython-311.pyc
        orbitalanalysis/rungeKutta4.py

no changes added to commit (use "git add" and/or "git commit -a")

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git add -A

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git commit -m "Finished rungeKutta4.py, which is the numerical solver and upda
te __init__"
[main d3524dc] Finished rungeKutta4.py, which is the numerical solver and update
 __init__
 5 files changed, 157 insertions(+), 1 deletion(-)
 create mode 100644 orbitalanalysis/__pycache__/rungeKutta4.cpython-311.pyc
 create mode 100644 orbitalanalysis/rungeKutta4.py

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git push
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 16 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 9.38 KiB | 2.35 MiB/s, done.
Total 9 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Dylnpn/OrbitAnalysis.git
   ff494da..d3524dc  main -> main

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
```

-

12/15 - 5:54 PM

Started with the EP script. Pushed and committed my changes to GitHub before my ASTE 557 final

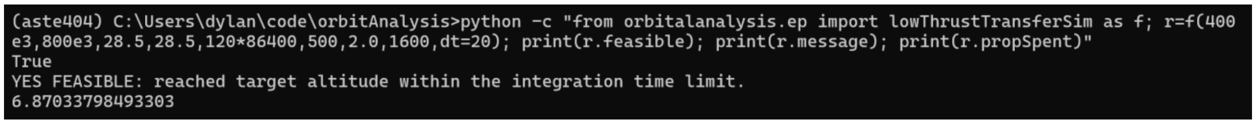


```
dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git add -A

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git commit -m "Writing EP code. Last commit and push before ASTE 557 FINAL AHH
HH"
[main 4f77e35] Writing EP code. Last commit and push before ASTE 557 FINAL AHHHH
 1 file changed, 148 insertions(+)

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 2.23 KiB | 2.23 MiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Dylnpn/OrbitAnalysis.git
   d3524dc..4f77e35  main -> main

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$
```

-

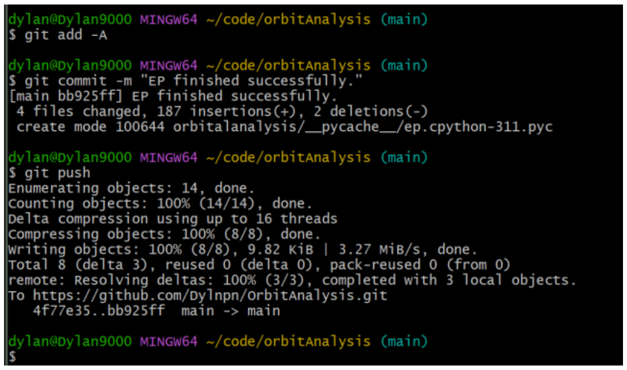
12/15 - 11:30 PM

Finished EP, ran a feasibility test and it worked!



```
(aste404) C:\Users\dylan\code\orbitAnalysis>python -c "from orbitalanalysis.ep import lowThrustTransferSim as f; r=f(400
e3,800e3,28.5,28.5,120*86400,500,2.0,1600,dt=20); print(r.feasible); print(r.message); print(r.propSpent)"
True
YES FEASIBLE: reached target altitude within the integration time limit.
6.87033798493303
```

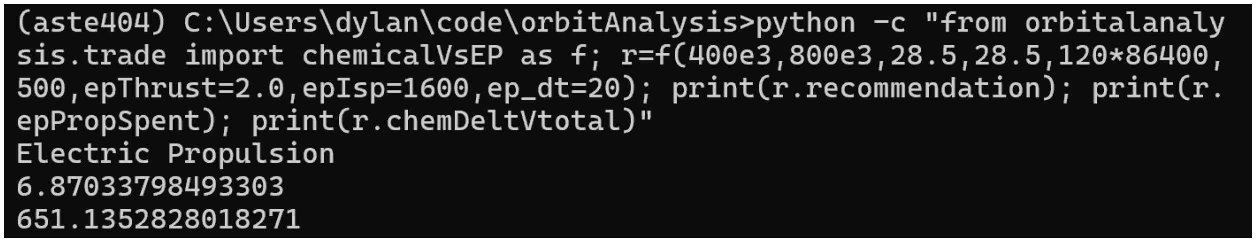
Pushed my commits:



```
dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git add -A

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git commit -m "EP finished successfully."
[main bb925ff] EP finished successfully.
 4 files changed, 187 insertions(+), 2 deletions(-)
 create mode 100644 orbitalanalysis/__pycache__/ep.cpython-311.pyc

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git push
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 16 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 9.82 KiB | 3.27 MiB/s, done.
Total 8 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/Dylnpn/OrbitAnalysis.git
   4f77e35..bb925ff  main -> main

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$
```

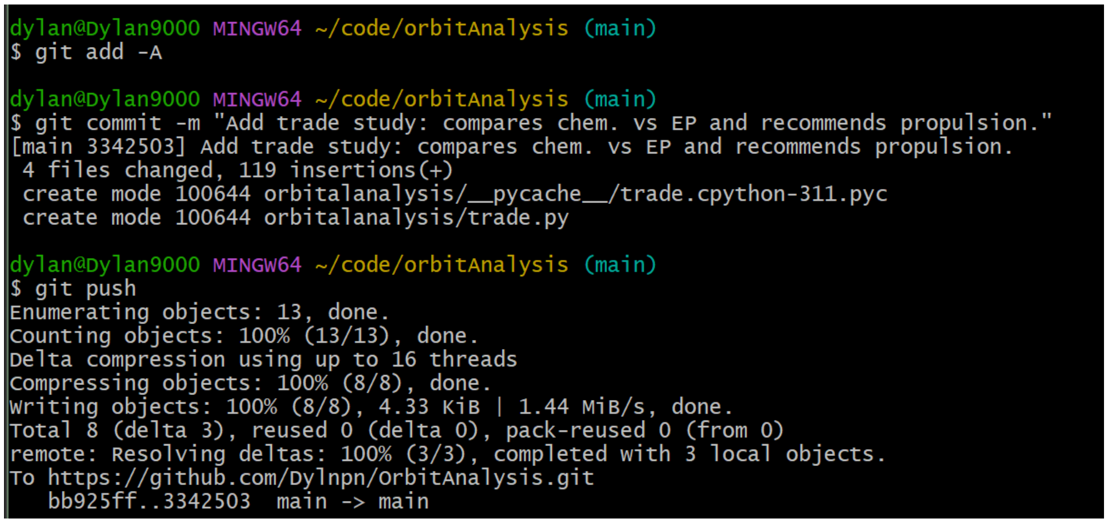
12/16 - 12:21 AM

Finished trade.py and ran a test to see which propulsion system is recommended. Nominal output!



```
(aste404) C:\Users\dylan\code\orbitAnalysis>python -c "from orbitalanaly
sis.trade import chemicalVsEP as f; r=f(400e3,800e3,28.5,28.5,120*86400,
500,epThrust=2.0,epIsp=1600,ep_dt=20); print(r.recommendation); print(r.
epPropSpent); print(r.chemDeltVtotal)"
Electric Propulsion
6.870337984933O3
651.1352828018271
```

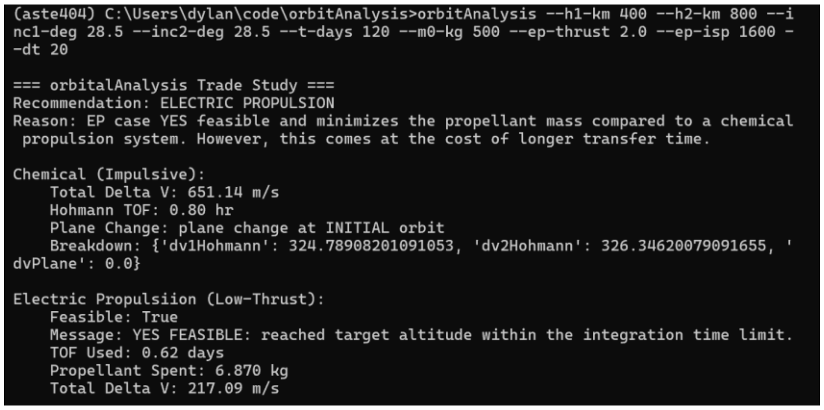
Committed and pushed:



```
dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git add -A

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git commit -m "Add trade study: compares chem. vs EP and recommends propulsion."
[main 3342503] Add trade study: compares chem. vs EP and recommends propulsion.
 4 files changed, 119 insertions(+)
 create mode 100644 orbitalanalysis/__pycache__/trade.cpython-311.pyc
 create mode 100644 orbitalanalysis/trade.py

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git push
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 16 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 4.33 KiB | 1.44 MiB/s, done.
Total 8 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/Dylnpn/OrbitAnalysis.git
   bb925ff..3342503  main -> main
```

12/16 - 1:15 AM

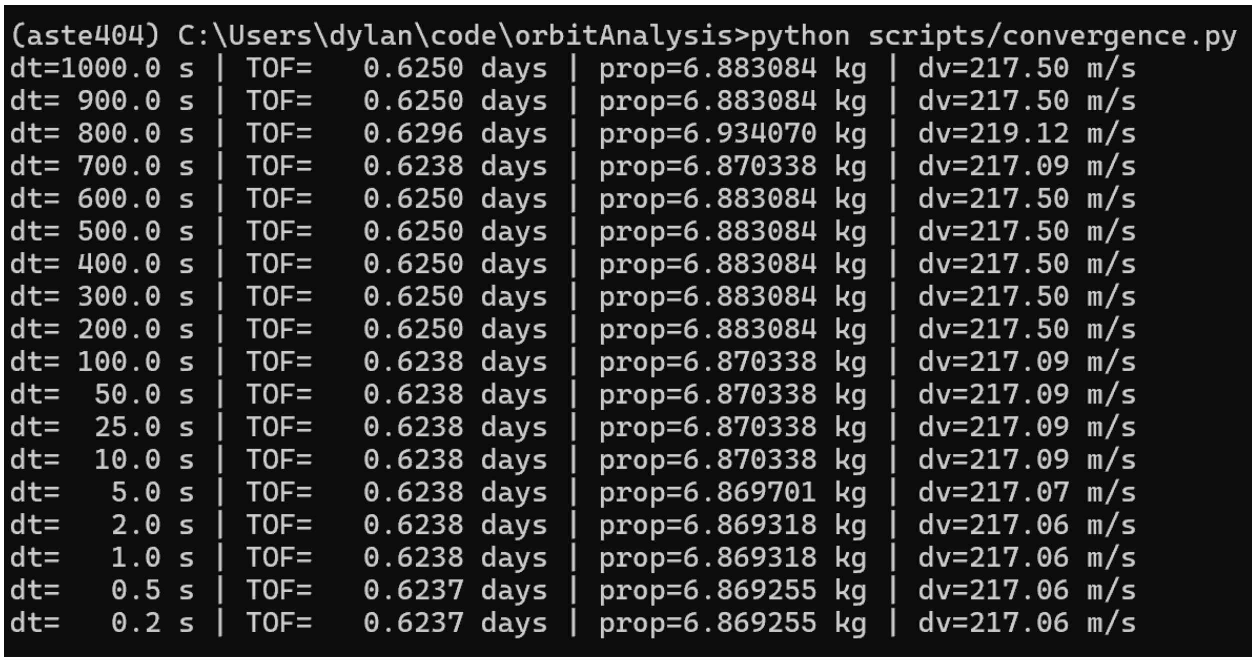
Wrote cli.py and ran a test:



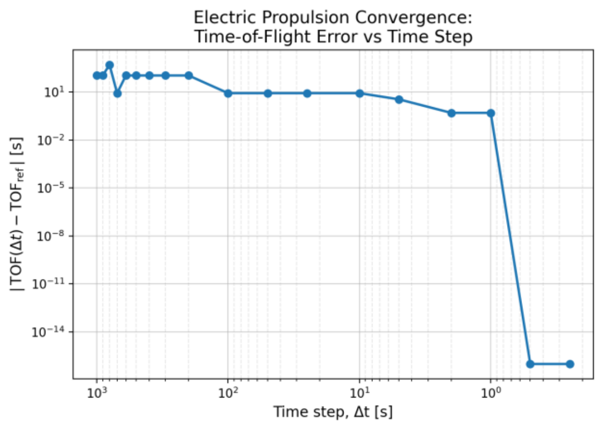
```
(aste404) C:\Users\dylan\code\orbitAnalysis>orbitAnalysis --h1-km 400 --h2-km 800 --i
nc1-deg 28.5 --inc2-deg 28.5 --t-days 120 --m0-kg 500 --ep-thrust 2.0 --ep-isp 1600 -
-dt 20

=== orbitalAnalysis Trade Study ===
Recommendation: ELECTRIC PROPULSION
Reason: EP case YES feasible and minimizes the propellant mass compared to a chemical
 propulsion system. However, this comes at the cost of longer transfer time.

Chemical (Impulsive):
    Total Delta V: 651.14 m/s
    Hohmann TOF: 0.80 hr
    Plane Change: plane change at INITIAL orbit
    Breakdown: {'dv1Hohmann': 324.78908201091053, 'dv2Hohmann': 326.34620079091655, '
dvPlane': 0.0}

Electric Propulsiion (Low-Thrust):
    Feasible: True
    Message: YES FEASIBLE: reached target altitude within the integration time limit.
    TOF Used: 0.62 days
    Propellant Spent: 6.870 kg
    Total Delta V: 217.09 m/s
```

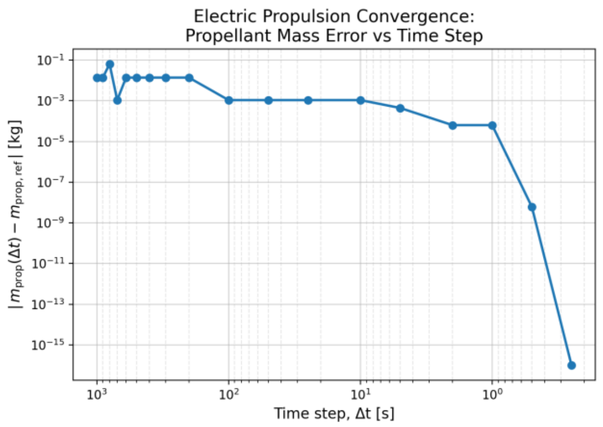
12/16 - 1:53 AM

Finished convergence.py script to show convergence of TOF and propellant as dt decreases. The following are the results:



```
(aste404) C:\Users\dylan\code\orbitAnalysis>python scripts/convergence.py
dt=1000.0 s | TOF=   0.6250 days | prop=6.883084 kg | dv=217.50 m/s
dt= 900.0 s | TOF=   0.6250 days | prop=6.883084 kg | dv=217.50 m/s
dt= 800.0 s | TOF=   0.6296 days | prop=6.934070 kg | dv=219.12 m/s
dt= 700.0 s | TOF=   0.6238 days | prop=6.870338 kg | dv=217.09 m/s
dt= 600.0 s | TOF=   0.6250 days | prop=6.883084 kg | dv=217.50 m/s
dt= 500.0 s | TOF=   0.6250 days | prop=6.883084 kg | dv=217.50 m/s
dt= 400.0 s | TOF=   0.6250 days | prop=6.883084 kg | dv=217.50 m/s
dt= 300.0 s | TOF=   0.6250 days | prop=6.883084 kg | dv=217.50 m/s
dt= 200.0 s | TOF=   0.6250 days | prop=6.883084 kg | dv=217.50 m/s
dt= 100.0 s | TOF=   0.6238 days | prop=6.870338 kg | dv=217.09 m/s
dt=  50.0 s | TOF=   0.6238 days | prop=6.870338 kg | dv=217.09 m/s
dt=  25.0 s | TOF=   0.6238 days | prop=6.870338 kg | dv=217.09 m/s
dt=  10.0 s | TOF=   0.6238 days | prop=6.870338 kg | dv=217.09 m/s
dt=   5.0 s | TOF=   0.6238 days | prop=6.869701 kg | dv=217.07 m/s
dt=   2.0 s | TOF=   0.6238 days | prop=6.869318 kg | dv=217.06 m/s
dt=   1.0 s | TOF=   0.6238 days | prop=6.869318 kg | dv=217.06 m/s
dt=   0.5 s | TOF=   0.6237 days | prop=6.869255 kg | dv=217.06 m/s
dt=   0.2 s | TOF=   0.6237 days | prop=6.869255 kg | dv=217.06 m/s
```



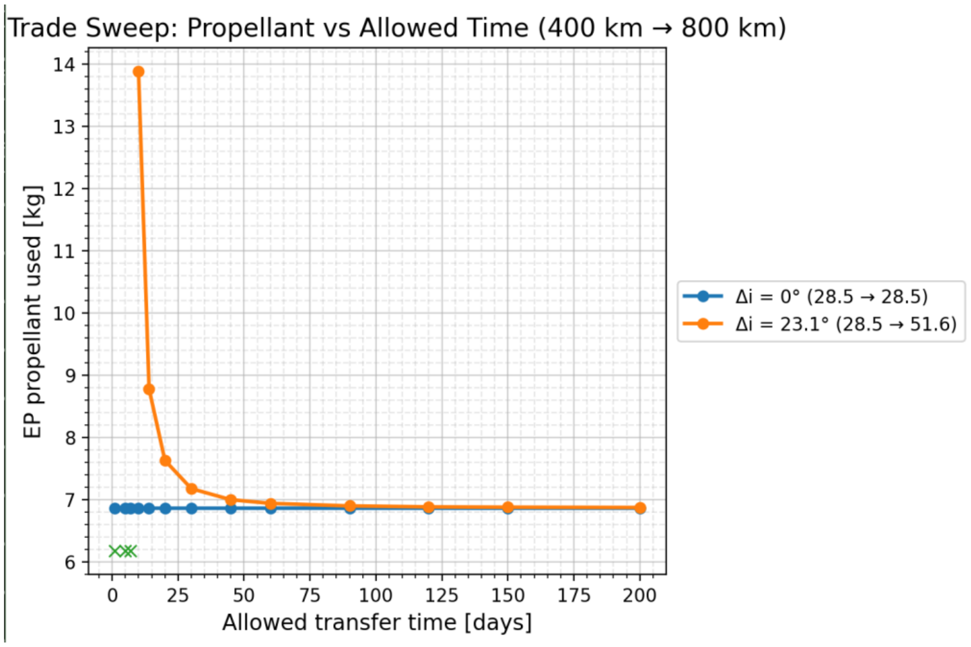


The plots look similar because the mass flow is constant in the EP model, meaning that the propellant consumed is simply the mass flow rate times the TOF, which should show a similar shape.

This was committed and pushed successfully!

12/16 - 2:18 AM

Completed trade_sweep.py, which plots propellant vs time allowed for different orbital changes.

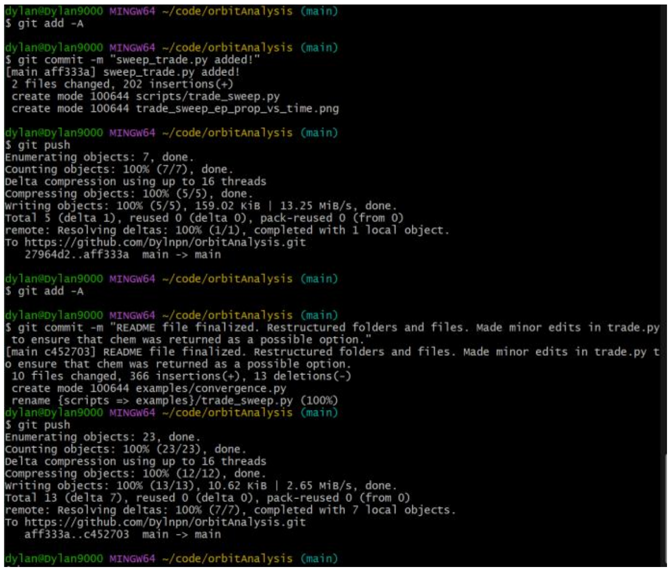


Committed and pushed!

-

12/16 - 12:13 PM

Finished writing the README.md file and made minor edits to debug:



```
dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git add -A

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git commit -m "sweep_trade.py added!"
[main aff333a] sweep_trade.py added!
 2 files changed, 202 insertions(+)
 create mode 100644 scripts/trade_sweep.py
 create mode 100644 trade_sweep_ep_prop_vs_time.png

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 16 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 159.02 KiB | 13.25 MiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Dylnpn/OrbitAnalysis.git
   27964d2..aff333a  main -> main

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git add -A

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git commit -m "README file finalized. Restructured folders and files. Made minor edits in trade.py
 to ensure that chem was returned as a possible option."
[main c452703] README file finalized. Restructured folders and files. Made minor edits in trade.py t
o ensure that chem was returned as a possible option.
 10 files changed, 366 insertions(+), 13 deletions(-)
 create mode 100644 examples/convergence.py
 rename {scripts => examples}/trade_sweep.py (100%)
dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
$ git push
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 16 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (13/13), 10.62 KiB | 2.65 MiB/s, done.
Total 13 (delta 7), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (7/7), completed with 7 local objects.
To https://github.com/Dylnpn/OrbitAnalysis.git
   aff333a..c452703  main -> main

dylan@Dylan9000 MINGW64 ~/code/orbitAnalysis (main)
```

5. Verification/Validation

5.1 Convergence Study

A numerical convergence is verified using **time-step refinement**, which reduces the Δt until changes in the following become negligible:

- Final orbital radius
- Total propellant spent

This ensures the numerical error doesn't affect the propulsion trade. Two plots are produced to ensure convergence:

i. TOF error vs time step
ii. Prop used error vs time step

These plots show error convergence towards zero as the time step decreases (finer calculations). The convergence metrics are based on the integrated history:

- Time to reach target (event time)
- Propellant used (mass history)
- Integrated found delta V (post integral)

5.2 Convergence Future Work

A different numerical error could be used in order to compare error produced for each program, allowing for interpolation between each software's output to create a more conservative result that takes into account multiple numerical approximations.

6. Results

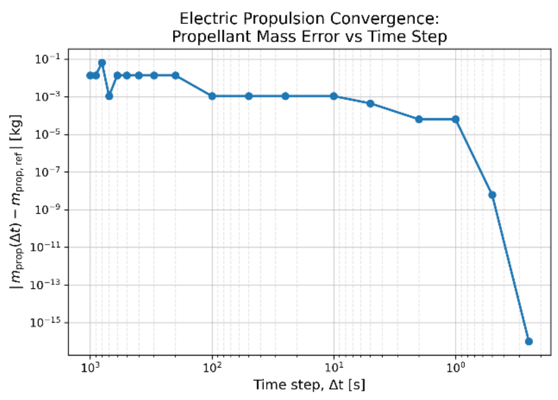
The main takeaways from this software are the following:

- Chemical transfers are always feasible but not propellant-efficient
- EP transfers use much less propellant, but infeasible for short periods
- Trade sweeps show EP feasibility boundaries
- Inclination change affects EP feasibility since it occupies more delta V

From this package, we obtain plots for convergence and an analysis sweep.

6.1 Convergence Results

The following plots are obtained:



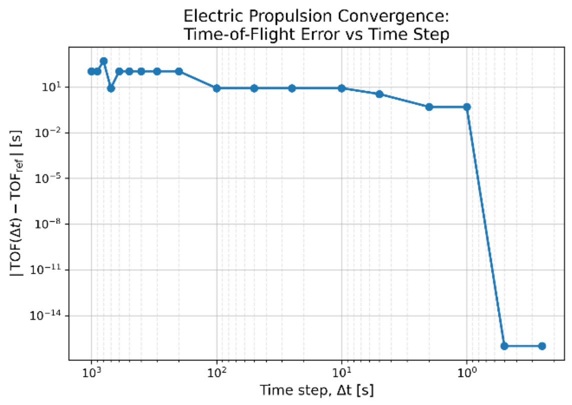


Figure 1: Convergence results are shown. Left plot shows error in propellant spent over time steps, while the right plot shows error in time of flight over time steps.

The in Figure 1 show convergence as the time step decreases, verifying that the model becomes highly accurate as the time step difference approaches an infinitesimally small quantity. Note that both plots have the same shape, and this is due to the fact that mass spent is proportional to flight time. This is because mass spent is obtained from the mass flow rate times the time spent thrusting.

6.2 Trade Sweep Results

To visualize the feasibility of EP with respect to inclination change and time, the following plot is produced.

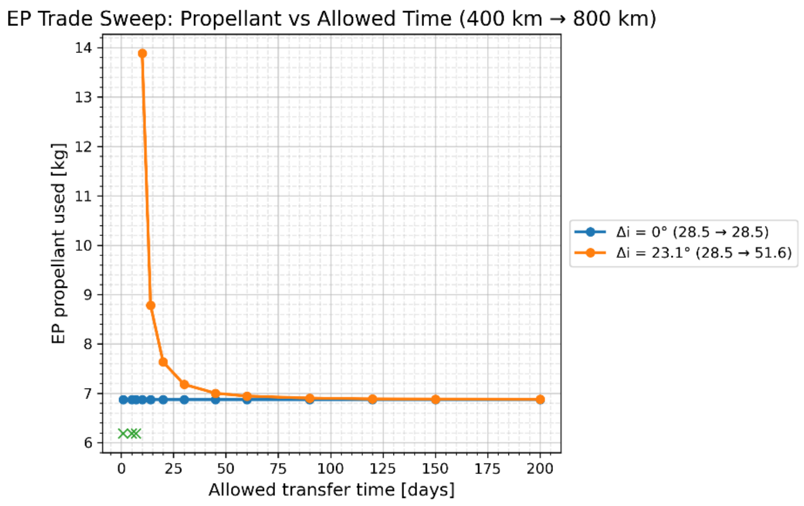


Figure 2: Two mission designs are shown. These results show the propellant mass used as a function of time given different desired inclination changes. Note that when there is no inclination change, only the orbital altitude change is performed, so the propellant used remains constant. The green data points indicate non-feasible EP capabilities.

Figure 2 shows that the amount of propellant used to perform an orbital altitude rise and inclination change are drastically larger than for a non-inclination change mission (blue). This allows user to see EP's limitations.

7. Reflection and Next Steps

While working on this project, I was able to learn a lot about python packages, structures, and the overall complexity of a multi-dependent program. I was able to observe how low thrust approximations are extremely sensitive to time steps, how event detection are imperative to ensure the integrator stops at desired point, and how numerical programs require a lot of system-level thinking. Additionally, while working on this project, I used a lot of knowledge from ASTE 475 and ASTE 580 to base the dynamics for orbital transfers and propellant consumption. Being able to adapt and use this knowledge practically allowed me to see how astronautical engineering essentially comes together.

Moving forward, in order to make this package more useful, the following implementations should be made:

- Addition of steering laws for normal and tangential velocity vectors
- Non constant thrust profiles
- Realistic orbital mechanics that don't neglect J2 perturbations or drag
- Unit testing to ensure output makes sense physically
- Give user ability to design thruster geometry to acquire different thrust, Isp, and mass flow rate profiles
- Allow for calculation of multi-stage rockets that could use a combination of chemical and EP systems.

7.1 LLM - ChatGPT

This was all ultimately done with the aid of an LLM, ChatGPT, which provided a structured guide developed from my idea and expectations. Using LLMs to create types of software like these is extremely useful, especially at an undergrad level where it explicitly describes the steps and approaches it's taking as it is helping you along the way. LLMs allow for the rapid execution of ideas, especially in software development where beginners suffer from syntax and coding language limitations. Using an LLM for this assignment proved to be extremely beneficial.

In order to ensure that ChatGPT gave me coherent and reasonable solutions, I relied on its ability to cross reference other conversations where I had fed it ASTE 580 and ASTE 475 knowledge. With this already set as a baseline, I then fed it the ASTE 404 MiniProject Rubric, along with the following prompt:

> Use the attached rubric to help me create my mini project.
>
> I am working on a numerical methods mini-project for an undergraduate astronautical engineering course. The idea is to implement a python tool that compares chemical and electrical propulsion systems for orbital transfers. Here, we will assume that the chemical method is impulsive, whereas the EP method is low thrust. Furthermore, we're simplifying the problem by assuming two circular Earth orbits with the ability to change inclination.
>
> What I want this program to do:
>
> 1. Accept user inputs for initial and final orbital altitude, inclination change, spacecraft mass, thrust level, Isp, and desired transfer time.
> 2. Compute an analytical chemical Hohmann transfer and plane change.
> 3. Use a numerical solution like RK4 to solve ODEs, which will help simulate EP low thrust. Keep in mind this needs a stop event for when the final orbit is reached.
> 4. Help find if using EP is feasible given a time and thrust constraint.
> 5. Perform a convergence study to serve as a verification of the numerical method. This convergence method can be simple, like a time-step based flow.
> 6. Plot a trade sweep curve to show how propellant and time are intertwined.
> 7. Allow user to interact with the program smoothly.
>
> I need your help guiding me through:
> 1. Using fundamental orbital mechanics equations from ASTE 580 and 475
> 2. Choosing the appropriate numerical method
> 3. Having a strong and clean package structure
> 4. Designing the convergence test
>
> This is an engineering design problem. As such, this model must be physically reasonable, relying on nothing but current numerical methods and orbital mechanic theories.

I've attached the transcript with the LLM to the package folder, named ASTE404_LLMTranscript.pdf. Please note that referencing every prompt with it would be unreasonable, as the comment extends for more than +100 pages.

It’s important to note that in order to follow along with ChatGPTs reasoning, I didn’t simply copy and paste code into my environment and scripts. I typed every single line, catching errors, syntax discrepancies, and naming the variables differently in order to ensure I absorbed how the software worked. Working this way allowed me to press the LLM about questionable choices, creating back and forth discussion where I was able to learn about different Python syntax and techniques.

8. LLM Transcripts

Please refer to the file: ASTE404_LLMTranscript.pdf