# IPASS Documentation; NKepler

Casper Smet; 1740426

June 2019

## Contents

# 1 Introduction

## 1.1 The name

Three weeks ago, I ventured to create a library for simulating orbits. The library was to be used as an educational tool for those interested in astrophysics.

I decided on simulating undisturbed[1] Keplerian[2] orbits. After spending a week forging an understanding of Kepler's laws, and about two days of attempting to implement these in Python, I realised that Keplerian orbits were too complex to implement in such a short period. Instead, I turned to circular orbits.

Thus, the name NKepler sprung into existance; - Not Kepler!

Compared to Keplerian orbits, circular orbits are very simple. While I am saddened by the fact that I was unable to simulate Keplerian orbits, I still highly enjoyed making NKepler.

## 1.2 Defining the product

For the purposes of this assignment, I contacted astrophysics student and long time friend, Christian Holbaum. For the purposes of this report, Mr. Holbaum will be referred to as Product-Owner, or PO for short.

In our first conversation on the topic of this assignment, both the PO and myself had a clear vision of the base product: a simple library capable of simulating orbits. It also needed to function as an educational program.

The PO noted that the current educational programs, while accurate, were inadequate for modern use. They lacked intuitiveness and required the user to calculate far too much data themselves. During this and many other conversations, I set up a list of requirements for the library:

- The user has to be able enter as little data themselves
- The user must be able to enter as much data as they wish
- The library must be able to calculate the missing data

Along with a list of requirements for the program

- The program must be able to simulate orbits visually
- The program shouldn't have too grand of a GUI, as most current programs utilise Unix Command Line plotting
- The program must contain a simulation of our solar system

## 1.3 Defining the extra's

After the requirements above were implemented, the extra's defined below were implemented. For the library:

- The ability to calculate position for a certain date

[1]meaning; not being subjected to gravitational forces of passing objects; unchanging
[2]meaning; defined as per Kepler's Laws; elliptical

- The ability to simulate a satellite orbiting another satellite (E.G. the Moon around the Earth around the Sun)

For the program:

- The ability to save simulations as GIFs
- Simulation showing the Moon around Earth
- Simulation showing the moons of Jupiter around Jupiter
- Simulation showing the Moon around the Earth as the Earth orbits the sun
- Simulation of the Moon around the Earth, with adjustable mass and radius
- Simulation of Jupiter's moons at certain dates.

The simulation of the moons around Jupiter being a special request from the PO, as Io is his favourite moon.

# 2 Implementing circular orbits

In this chapter, I detail and defend certain design decisions.

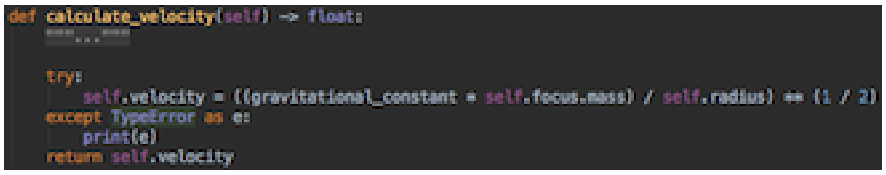
## 2.1 An appeal to Reusability

### 2.1.1 Basic design philosophy

By design, all IPASS libraries must be highly reusable. I strove to make each function, variable and method as reusable as possible. In order to do so, I attempted to uphold the following rule-of-thumb:

> A function may only include one equation

For example, the function used to calculate velocity[3]:



```python
def calculate_velocity(self) -> float:
    """..."""

    try:
        self.velocity = ((gravitational_constant * self.focus.mass) / self.radius) ** (1 / 2)
    except TypeError as e:
        print(e)
    return self.velocity
```

[3]Appendix A.4

### 2.1.2 Conglomerate functions

Some equations tend to only get called in combination with each other, for example angle_to_x and angle_to_y[4]. They also both take the same variables: angular displacement and radius.

To enlarge the ease of use for these two functions, they were combined into a conglomerate function[5]: angular_displacement_to_coordinates.

### 2.1.3 Increasing readability

The process of creating one's library in this manner usually does result in lessened readability. To counteract this, I attempted to uphold a second rule-of-thumb:

> The functionality of each function or somewhat complex line must be documented carefully via commenting or DOCSTRINGS

## 2.2 Notes on optimisation

### 2.2.1 Identifying the problem

In its standard form, angular displacement, and thus orbital position, is calculated per second. Our moon has an orbital period of 27 days, which equals roughly 2 million seconds. Translated to function calls, this is significantly more:

| | |
|---|---|
| angular_displacement_at_t: | $2.33E6 \cdot 1$: $2.33E6$ |
| angular_displacement_to_coordinates: | $2.33E6 \cdot 1$: $2.33E6$ |
| angle_to_x: | $2.33E6 \cdot 1$: $2.33E6$ |
| angle_to_y: | $2.33E6 \cdot 1$: $2.33E6$ |
| total amount of calls: | : $9.32E6$ |

This indicates a problem, the Moon's orbital period is relatively short. The Earth's orbit is ten times larger than the Moon's, Jupiter's is roughly ten times that of Earth. -and Neptune's is more than ten times that of Jupiter. Without any optimisation, calculating the Moon's orbit took roughly one minute. Earth, Jupiter and Neptune would take factors of ten longer.

To counteract this issue, two steps were taken:

1. Implementing a 'time interval'
2. Implementing memoization

[4]Appendix A.10 & 11

[5]A function which purpose is to call other functions

### 2.2.2 Time interval

The first method to optimise this process, is to decreasing the amount of function calls. This is accomplished by implementing a 'time interval' functionality. Calculating the orbit of a planet per second is usually unnecessary. Thus, the option to calculate position per minute, hour, or any other time interval was implemented. Time interval's standard value is 3600 seconds[6].

| | | |
|---|---|---|
| angular_displacement_at_t: | $\frac{2.33E6}{3600}$ | : 647 |
| angular_displacement_to_coordinates: | $\frac{2.33E6}{3600}$ | : 647 |
| angle_to_x: | $\frac{2.33E6}{3600}$ | : 647 |
| angle_to_y: | $\frac{2.33E6}{3600}$ | : 647 |
| total amount of calls: | | : 2589 |

Time interval is a static variable, it can be altered by the user simply by calling:

Satellite.time_interval $= t$

### 2.2.3 Memoization

The second optimisation method is called memoization. Through memoization, you can save the results of function calls and return the cached results when the same variables are inputted again. This is best applied at angular_displacement_to_coordinates. This function gets called for each value for angular displacement. Due to rounding[7] at two decimals, the same value of angular displacement occurs many times.

Memoization is especially useful in cases where more than one full orbit is calculated, as in a circular orbit all values are repeated after the first cycle. Due to the nature of angular displacement, its value reaches beyond 6.28 radians[8]. To be able to use memoization more effectively, the function range_setter was written. This function alters the given variable to be within the range of 0 and the given maximum. This ensures that all values of angular displacement fall between 0 and 6.28 radians.

Memoization was implemented in angular_displacement_to_coordinates, using the Python module lru_cache.

### 2.2.4 Results

Now, on an early 2015 model MacBook being used to write this report, 30,171,786 function calls are made to calculate 30 full orbits[9]. Two full orbits per inner

[6]One hour
[7]Accuracy can be set by altering Satellite.accuracy
[8]One circle
[9]Assuming a time interval of 60 seconds

planet in our solar system, also including the Moon around the Earth. It took 17.139 seconds to simulate these orbits. A sharp decline compared to the one minute it took to calculate the Moon before.

## 3 Summary

In short, NKepler has fulfilled all of its initial requirements. The library is short, relatively simple to use and fast. The program, while simple, effectively showcases all of NKeplers functionality. Parts of the program could be reused to effectively plot orbits of new satellites, if the user were to add them. The program simply serves its purposes.

## 4 A brief thanks

In this chapter, I would like to thank all those who helped me throughout this year. Firstly, I want to thank Christian Holbaum, who helped me forge an understanding of basic astrophysics. Secondly, a thank you to Arno Kamphuis. He was the one that first gave me the idea for this project. Thirdly, a thank you to my IPASS-coordinators Diptish Dey, Huib Aldewereld and Nick Roumimper, for assisting me with various issues. Fourthly, a thank you to my peers, who challenged and assisted me in various fields. Last, but certainly not least,

> a thank you to the AAI-division at the HU. It has been a uniquely challenging and, most importantly, fun year.

I look forward to my second year.

## 5 Appendices

### 5.1 Appendix A: Formula's used in NKepler

Appendix A can be found in the .pdf file report/formulas_NKepler.pdf

### 5.2 Appendix B: Documentation

Appendix B can be found in the folder docs/_build/html If an alternative file format is required for the tutorial, that can be found here: docs/tutorial.rst

### 5.3 Appendix C: README

Appendix C can be found at /README.md

### 5.4 Appendix D: License

Appendix D can be found at /LICENSE

## 5.5 Appendix E: Poster

Appendix E can be found at report/IPASS-poster 1.0.pdf