A Computational Rediscovery of Neptune and other Applications in Numerical Astronomy

A thesis submitted in partial fulfilment of the requirements for the degree of

Bachelor of Sciences

in

Physics

By

Muhammad Haider Khan
22100217

Supervisor: Dr. Muhammad Sabieh Anwar

School of Science and Engineering
Lahore University of Management Sciences
LUMS

May 2022

Abstract

This paper serves to report on the computational reproduction of the research paper, ”A geometric method to locate Neptune”[1]. The computational program thus produced can be used to obtain the Mass, Sidereal Period, Semi-major axis, and Synodic Period (with respect to any planet) of Neptune. The paper also reports on the ephemeris data simulation for other exoplanets via pure geometric methods and consequently a complete computational algorithm thus established can be extended to the exoplanetary systems. Provided with the orbital parameters and experimental ephemeris of already discovered exoplanets in a system, the program can help hint at undiscovered exoplanets and ascertain their Mass and Period. The thesis also details on the computational implementation of the Transit Light Curve method of Exoplanet detection.

پروجیکٹ کا خلاصہ

حیدر خان

یہ تحقیقی پروجیکٹ "نیپچون کی تلاش کے ایک عددی طریقہ" کے عنوان کے حامل ایک مقالے کا جائزہ ہے۔اس عددی حساب کے ذریعے سیارہ نیپچون کے سیمی میجر خط، مداری مدت (پیریڈ) اور وزن کا اندازہ لگانے میں مدد ملتی ہے۔ میرے کمپیوٹر پروگرام کو ماورائے شمسی نظاموں میں نئے سیاروں کی دریافت تک بھی بڑھایا جاسکتا ہے۔ مزید برآں، ہم خالص ہندسی (نومیریکل) طریقوں کے ذریعے دوسرے ماورائے شمسی سیاروں[1] کے لیے ایفیمرس[2] ڈیٹا کی جنریشن کی تقلید کر سکتے ہیں اور اس طرح اس عددی ڈیٹا اور تجرباتی ڈیٹا کے درمیان کوئی موجود فرق ماورائے شمسی سیاروں کی موجودگی کی نشاندہی میں معاون ہو سکتا ہے۔ ان فلکی اجسام کے مداری خواص اور اوزان کی اس الگورتھم کا استعمال کرتے ہوئے پیش گوئی بھی کی جا سکتی ہے۔

نیز، ہم نے اس کام میں کئی فلکیات کے چند مسائل کو حل کرنے میں معاون پائتھن[3] کمپیوٹر کوڈ بھی بنائے ہیں جنھیں فلکیات کے میدان میں ایک نیا طالب علم فلکی طبیعیات کے مسائل اور ان کے حل کے بارے میں جاننے کے لیے استعمال کر سکتا ہے۔ یہ علم اور ہنر ان طلباء کے لیے نہایت مفید اور کارآمد ثابت ہو سکتا ہے۔

ہم نے ستاروں کے عین سامنے سے گزرنے والے سیاروں اور اجسام کی حرکت کو جانچنے کے لیے زمین پہ حاصل ہونے والی روشنی میں ردوبدل کے ذریعے سیاراتی میکانیات کو سمجھنے والا کمپیوٹر پروگرام بھی تیار کیا

---

[1] exoplanets
[2] ephemeris
[3] pyhton

ہے. اس طریقے کو روشنی کا تغیراتی طریقہ[4] کہا جاتا ہے اور اسے ماروائے شمسی سیاروں کا پتہ لگانے کے لیے ایک اور وسیع پیمانے پر استعمال ہونے والے طریقے کے طور پہ جانا جاتا ہے۔

ہمارے نظام شمسی میں نیپچون ایک ایسا سیارہ تھا جس کی عملی دریافت ہونے سے پہلے ہی ریاضیاتی طور پر پیش گوئی کردی گئی تھی۔ یورینس کے مدار کی سے بے قاعدگی کو دیکھتے ہوئے، اربین لی ویریئر نے اگست 1846ء میں یہ پیشن گوئی کی تھی کہ ایک غیر دریافت سیارہ یورینس کے مدار سے باہر موجود ہونا چاہیے. بعد ازاں ماہر فلکیات جوہان گوٹ فرائیڈ گیلے نے ستمبر 1846ء میں برلن آبزرویٹری میں بالآخر اس سیارے کو دریافت کر لیا۔ باب اول جس کا عنوان "گریویٹیشنل ماڈلنگ" ہے، میں ہم اپنے تحقیقی پروجیکٹ کے جیومیٹرک مواد کی وضاحت کرتے ہیں۔ ہم اپنے نظام شمسی میں تمام سیاروں کی پوزیشن ویکٹر اور رفتار کے دستیاب فلکیاتی ڈیٹا کا استعمال کرتے ہیں اور نیوٹن کے قوانین حرکت پر ویکٹر تجزیہ کے اطلاق کے ذریعے، نیپچون کے مداری پیرامیٹرز کا اندازہ کرتے ہیں۔ نیز، یہ بیان کرنے کے لیے کہ مختلف سیاروں اور ستاروں کے نظاموں کے مدار کیسے کام کرتے ہیں، ہمیں کچھ ہندسی تصورات اور ان کی ریاضیاتی تشکیل سے واقف ہونا ہوگا۔ پھر ہم دو جسموں کے مسئلے کو درست طریقے سے بیان کر سکتے ہیں: ایک ستارہ اور ایک سیارہ مدار میں یا دو ستارے ایک مداری نظام تشکیل دیتے ہیں۔ مہذب اندازوں کے ساتھ، ہم دو ستاروں اور ایک سیارے، یہاں تک کہ تین ستاروں کے ساتھ تین جسمانی نظام کا بھی ماڈل بنا سکتے ہیں۔ پہلا باب ان میکانیکی اور ریاضاتی مسائل پر مشتمل ہے۔

باب دوم میں، ہم پائتھن پروگرامنگ کو فلکی طبیعیات میں مسائل کے حل کے استعمال پر توجہ مرکوز کرتے ہیں۔ ہم پائتھن لائبریریوں کو متعارف کراتے ہیں جو فلکی طبیعیات میں فائدہ مند ہیں اور ساتھ ہی ان لائبریریوں کو مختلف پروگرام لکھنے کے لیے استعمال کرتے ہیں۔ اس میں ایسے پروگرام شامل ہیں جو کسی شے کے لیے قوس نہاری[5] کی لمبائی کا حساب لگاتے ہیں۔ یہ وہ راستہ ہے جس پر جسم کسی مخصوص جغرافیائی محل وقوع کے لیے آسمان پر چلتا ہے۔ اس سے یہ معلوم کرنے میں مدد ملتی ہے کہ ایک شے

[4] transit light curve method
[5] diurnal Arc

کسی خاص دن کے لیے اس مقام کے لیے آسمان میں کس وقت نمایاں رہتی ہے۔ اس پروگرام پہ مزید کام کرتے ہوئے ہم یہ ڈیٹا حاصل کر سکتے ہیں کہ سورج کسی مقام کے لیے سال بھر میں کتنی مدت کے لیے آسمان پر رہتا ہے جس سے ہم سال کے ہر ایک دن کی طوالت کا اندازہ لگا سکتے ہیں۔ دوسرے پروگراموں میں دو نجمی نظام[6] کے مداری راستوں کا سراغ لگانا شامل ہے۔ ہم نے ایک چھوٹے سیارے کے ساتھ دو نجمی نظام کا پروگرام بھی متعارف کرایا۔ ان اجسام کے سیمی میجر ایکسز کو تبدیل کرکے آپ مختلف قسم کے مدار حاصل کرسکتے ہیں۔ یہ تمام پروگرام متغیرات کی اجازت دیتے ہیں جن کے ساتھ طلباء کام کر سکتے ہیں اور نت نئے نتائج اخذ کر سکتے ہیں۔

ایک بار جب ہم اپنے ماڈل کی ہندسی بنیادیں قائم کر لیتے ہیں اور پائتھن پروگرامنگ میں مہارت حاصل کر لیتے ہیں، باب سوم بالآخر نیپچون کی دریافت کے لیے جیومیٹرک ماڈل کی کمپیوٹیشنل پروگرامنگ پر توجہ مرکوز کرتا ہے۔ اس فصل کا آغاز اس مفروضے کے ساتھ ہوتا ہے کہ ہم نیپچون کی موجودگی اور مداری پیرامیٹرز سے لاعلم ہیں۔ یورینس کے اسراع[7] کے عوامل کے لیے ایک گراف بنا کر، ہم اندازہ لگاتے ہیں کہ نظام شمسی میں ایک آٹھویں سیارے کی موجودگی کے بغیر اس کی وضاحت نہیں کی جا سکتی۔ اور ہاں یہ سیارہ نیپچون ہے! اس ڈیٹا کا استعمال کرتے ہوئے، ہم نیپچون کے مداری پیریڈ کا حساب لگاتے ہیں اور اس سے اس کا وزن بھی معلوم کرتے ہیں۔

آخری اور چوتھا باب ماورائے شمسی سیاروں کی دریافت کا ایک اور طریقہ بیان کرتا ہے جسے ٹرانزٹ لائٹ کرو طریقہ کہا جاتا ہے۔ یہ اس بات پر انحصار کرتا ہے کہ جب کوئی سیارہ ستارے کے سامنے سے گزرتا ہے (ستارے کو گرہن لگتا ہے) تو ستارے سے خارج ہونے والی روشنی جو مشاہدہ کرنے والے تک پہنچ رہی ہوتی ہے، اس کی شدت میں کمی آتی ہے۔ روشنی میں یہ کمی جب بار بار ہوتی ہے، تو یہ ایک ماورائے

[6] binary star systems
[7] acceleration

شمسی سیارے کی موجودگی کی نشاندہی کرتی ہے۔ اس طرح، ایک ستاری نظام کا ڈیٹا اکٹھا کر کے ، ہم اس سیارے کے مداری پیرامیٹرز کا حساب لگاتے ہیں جو اس ستارے کے گرد گھومتا ہے۔

Contents

1 Gravitational Modelling

1.1 Introduction

Neptune was the one planet in our solar system that was mathematically predicted before it was discovered. Looking at the irregularity of Uranus from its predicted orbit, Urbain Le Verrier postulated that an undiscovered planet should exist beyond Uranus's orbit and it was discovered by astronomer Johann Gottfried Galle in the Berlin Observatory in 1846 [4]. Here we make use of the astronomical data available of the position vectors and velocity of all the planets in our solar system and through the application of vector analysis on Newton's laws of motion, calculate the orbital parameters of Neptune.

1.2 Newton's Law of Gravitation

Starting with Newton's law of Gravitation written for the ith object in a system is given by:

$$\vec{F_i} = -\sum_{j\neq i}^{9} GM_iM_j\frac{\vec{x}_i - \vec{x}_j}{|x_i - x_j|^3}. \tag{1}$$

where G is the Gravitational Constant with a value of $6.671 \times 10^{-11}\text{Nm}^2\text{kg}^{-2}$.

This force provides the acceleration for the ith object during its motion which is given by Newton's second law of motion as follows:

$$\vec{F_i} = M_i\frac{d^2\vec{x}_i}{dt^2}. \tag{2}$$

Equating (1) and (2) gives us,

$$\frac{d^2\vec{x}_i}{dt^2} = -\sum_{j\neq i}^{9} GM_j\frac{\vec{x}_i - \vec{x}_j}{|x_i - x_j|^3}. \tag{3}$$

Eq. (1) is written for 9 objects: the Sun, 7 planets (up till Uranus) and the yet to be discovered Neptune.

1.3 The Vector V(t)

Knowing that Neptune is not yet discovered, one can separate all the terms in the expansion of this equation that are concerned with Neptune from the other terms. For this, one write Eq. (1) in terms of the Sun and Uranus and subtract them.

$$\frac{d^2\vec{x}_\odot}{dt^2} = -\sum_{j\neq\odot,N}^{9} GM_j\frac{\vec{x}_\odot - \vec{x}_j}{|x_\odot - x_j|^3} - GM_N\frac{\vec{x}_\odot - \vec{x}_N}{|x_\odot - x_N|^3}.$$
$$\frac{d^2\vec{x}_U}{dt^2} = -\sum_{j\neq U,N}^{9} GM_j\frac{\vec{x}_U - \vec{x}_j}{|x_U - x_j|^3} - GM_N\frac{\vec{x}_U - \vec{x}_N}{|x_U - x_N|^3}.$$

Subtracting the two equations, one gets

$$\frac{d^2(\vec{x}_U-\vec{x}_{\odot})}{dt^2} = -\sum_{j\neq\odot,U,N}^{9} GM_j\frac{\vec{x}_U-\vec{x}_j}{|x_U-x_j|^3} - GM_{\odot}\frac{\vec{x}_U-\vec{x}_{\odot}}{|x_U-x_{\odot}|^3} - \sum_{j\neq\odot,U,N}^{9} GM_j\frac{\vec{x}_{\odot}-\vec{x}_j}{|x_{\odot}-x_j|^3} - GM_U\frac{\vec{x}_{\odot}-\vec{x}_U}{|x_{\odot}-x_U|^3} - GM_N(\frac{\vec{x}_U-\vec{x}_N}{|x_U-x_N|^3} - \frac{\vec{x}_{\odot}-\vec{x}_N}{|x_{\odot}-x_N|^3}).$$

Define,

$$\vec{r}_i = \vec{x}_i - \vec{x}_{\odot}.$$
$$\implies \vec{r}_N - r_U = \vec{x}_N - \vec{x}_U.$$

So, the above equation becomes,

$$\frac{d^2\vec{r}_U}{dt^2} = GM_N(\frac{\vec{r}_N-\vec{r}_U}{|\vec{r}_N-\vec{r}_U|^3} - \frac{\vec{r}_N}{|r_N|^3}) - G(M_{\odot}+M_U)\frac{r_U}{|r_U|^3} + \sum_{j\neq\odot,U,N}^{9} GM_j(\frac{\vec{r}_j-\vec{r}_U}{|x_N-x_U|^3} - \frac{\vec{r}_j}{|r_j|^3}).$$

Finally, separating the terms involving Neptune's parameters from the others gets us,

$$\vec{V}(t) = GM_N(\frac{\vec{r}_N-\vec{r}_U}{|\vec{r}_N-\vec{r}_U|^3} - \frac{\vec{r}_N}{|r_N|^3}). \tag{4}$$

$$\vec{V}(t) = \frac{d^2\vec{r}_U}{dt^2} + G(M_{\odot}+M_U)\frac{r_U}{|r_U|^3} - \sum_{j\neq\odot,U,N}^{9} GM_j(\frac{\vec{r}_j-\vec{r}_U}{|\vec{r}_j-\vec{r}_U|^3} - \frac{\vec{r}_j}{|r_j|^3}). \tag{5}$$

The variable $\vec{V}(t)$ here is the part of Uranus's acceleration that cannot be explained by other planets. If Neptune did not exist then by definition in Eq. (2), $\vec{V}(t)$ will be 0. However, obtaining the data of the other planets' motions can help us plot the magnitude of $\vec{V}(t)$ and it comes out as follows in figure 1.1

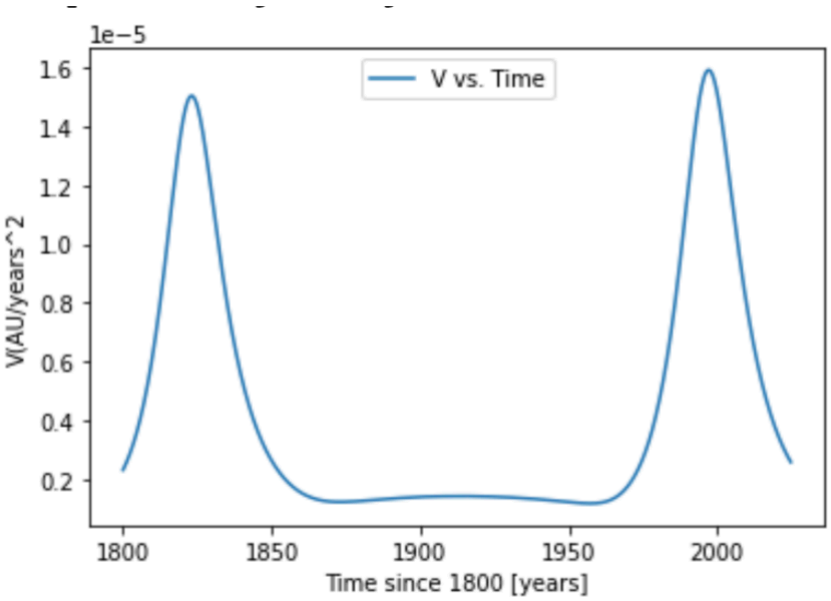

Figure 1.1 $|\vec{V}(t)|$ versus time since 1800.

This graph is obtained by plotting the magnitude of the vector $\vec{V}(t)$ from equation (5) by obtaining the ephemeris data of Uranus and the inner planets [2].This clearly is not zero with two very clear spikes in late-1822 and mid-1994. This ascertains our initial guess that an 8th planet should exist in our Solar System.

To understand the two spikes, we need to take into consideration the structure of $\vec{V}(t)$ from equation (4). Note that $\vec{V}(t)$ maximizes when the magnitude of the position vector of Neptune with respect to Uranus reduces to its minimum. This occurs when Neptune and Uranus form a conjunction.

The direction of vector $\vec{V}(t)$ is also of high significance as we will see in chapter 3. However, we do not need to worry about it at all instants as it has a complicated geometry, and we only need it at some points.

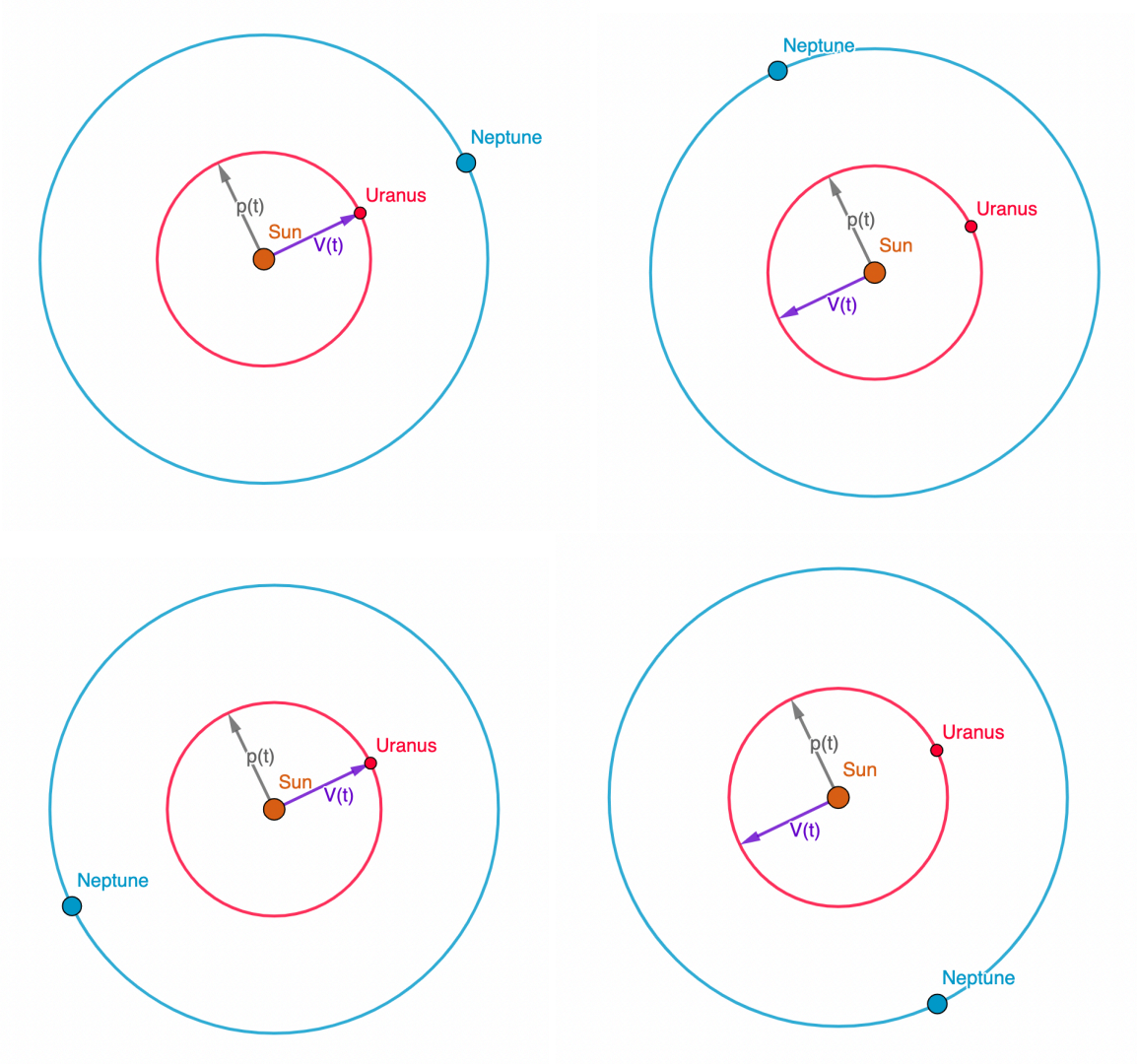

Figure 1.2: The trend of $\vec{V}(t)$ as Neptune makes a full circle but Uranus is held static.

The conjunction is the instant when the sun, Neptune, and Uranus lie in the same line on the same side of the sun. An opposition is the event when Neptune and Uranus are on the opposite sides of the sun while being in the same line.

Note that, whenever a conjunction occurs, all the vectors that make up $\vec{V}(t)$ point in the same direction:

towards Neptune and Uranus. Similarly, whenever there is an opposition, $\vec{V}(t)$ again points in the direction of Uranus.

To understand this, assume that a new coordinate system is defined where Sun is at the origin and the +x-axis is along the line joining Uranus to the origin. The negative x-axis would thus point towards Neptune. Thus, the new position vectors of Neptune and Uranus will have zero values for y and z coordinates. $|\vec{r}_{N_x}{}'|$ will be larger than $|\vec{r}_{U_x}{}'|$ so, $\frac{\vec{r}_{N_x}{}'-\vec{r}_{U_x}{}'}{|\vec{r}_{N_x}{}'-\vec{r}_{U_x}{}'|^3}$ will be pointed towards Neptune. However, it can be seen that the magnitude of $\frac{\vec{r}_{N_x}{}'-\vec{r}_{U_x}{}'}{|\vec{r}_{N_x}{}'-\vec{r}_{U_x}{}'|^3}$ is much smaller than that of $\vec{r}_{N_x}{}'$, so resultantly, $\frac{\vec{r}_{N_x}{}'-\vec{r}_{U_x}{}'}{|\vec{r}_{N_x}{}'-\vec{r}_{U_x}{}'|^3} - \frac{\vec{r}_{N_x}{}'}{|\vec{r}_{N_x}{}'|^3}$ will point towards Uranus. Thus, we can say that $\vec{V}(t)$ is pointed towards Uranus during both Conjunctions and Oppositions. This information will be of immense help while determining the time of conjunctions and oppositions.

1.4 The Period of Neptune

We define another vector $\vec{p}(t)$ that is perpendicular to the position vector of Uranus and is in the same plane as the three entities. Now you insert the information we deduced in the previous paragraph for the direction of V and notice that $\vec{p}(t)$ and $\vec{V}(t)$ are perpendicular to each other in the events of Conjunctions and Oppositions. Thus, if we define another quantity that is the projection of $\vec{V}(t)$ on $\vec{p}(t)$, we see that this quantity will be zero in case of both oppositions and conjunctions.

Keeping our goal in mind: calculating the sidereal period of Neptune that is the time it takes to complete one cycle around the sun and all the quantities we have so far defined will prove essential in obtaining the sidereal period of the planet.

Synodic period is another quantity that we need to evaluate for Neptune before we arrive at our goal. Synodic period of Neptune with respect to Uranus will be the time it would take to return to the same position around the sun as seen from Uranus. So, the time period between two conjunctions can be counted as the Synodic period for Neptune with respect to Uranus.

1.5 Kepler's Laws

Lastly, we need to acquaint ourselves with the Kepler's laws of planetary motion. There are three laws, the third of which is of use to us which states that, "The square of Sidereal period of a planet is directly proportional to the cube of its semi-major axis." [9]

$$a_N^3 = cT_N^2. \tag{6}$$

Now if you take the Sidereal Time Period in years and calculate the semi-major axis in Astronomical

Units, the value of c for our solar system comes out as 1 $(AU)^3/yr^2$.

1.6 Orbital Mechanics

To describe how the orbits of various planetary and stellar systems work, we have to be familiar with some geometric concepts and their mathematical formulation. Then we can accurately describe the two-body problem: a star and a planet in orbit or two stars (referred to as ä Binary") forming an orbital

system. With decent approximations, we can also model a three body system with two stars and a planet or even three stars.

Consider, as a first, that we have at hand, a two star system in mutual rotatory motion. The stars A and B have masses M_1 and M_2. If we take the frame where the center of mass of the system is the origin, we have the following position vectors for the two stars,

$$\vec{r_1} = -\frac{M_2}{M_1+M_2}\vec{d},$$

$$\vec{r_2} = \frac{M_1}{M_1+M_2}\vec{d} \tag{7}$$

Here $\vec{d} = \vec{r_2} - \vec{r_1}$, the relative displacement of the two stars. The center of mass is given by,

$$\vec{r}_{CM} = \frac{M_1\vec{r_1} + M_2\vec{r_2}}{M_1+M_2}, \tag{8}$$

and the force exerted on star B by A is given by,

$$\vec{F} = G\frac{M_1M_2}{d^3}\vec{d}. \tag{9}$$

This produces acceleration in the two bodies,

$$\vec{a}_1 = \frac{GM_2}{|r_2-r_1|^3}(\vec{r_2} - \vec{r_1}).$$

$$\vec{a}_2 = -\frac{GM_1}{|r_2-r_1|^3}(\vec{r_2} - \vec{r_1}). \tag{10}$$

We also have the $vis-viva$ equation that describes the relative velocity of stars in a binary system, and is given as [11],

$$v^2 = G(M_1+M_2)(\frac{2}{d} - \frac{1}{a}). \tag{11}$$

Here again, d is the magnitude of the relative displacement of the two stars and a is defined as the semi-major axis of the elliptical motion of the the vector $\vec{d}(t)$.

To define the initial values of our coupled differential equations problem, we take the periastron coordinates for the binary, i.e. when the two stars are in the closest proximity to each other. We take the planetary motion to be along the x-y axis. We also align our x-axis along the line joining the two stars at periastron. Let d_p be the distance separating the two stars at periastron, which the the instance of the closest proximity of the two stars, given by $d_p = a(1-e)$, then we have

$$\vec{r_1}(0) = (\frac{M_2}{M_1+M_2}d_p, 0, 0).$$
$$\vec{r_2}(0) = (-\frac{M_1}{M_1+M_2}d_p, 0, 0).$$

At periastron, the velocities of the two bodies are perpendicular to their position vectors and given by,

$$\vec{v}_1(0) = \frac{d}{dt}\vec{r}_1(0) = (0, -\frac{M_2}{M_1+M_2}d_p, 0)$$
$$\vec{v}_1(0) = \frac{d}{dt}\vec{r}_2(0) = (0, \frac{M_1}{M_1+M_2}d_p, 0).$$

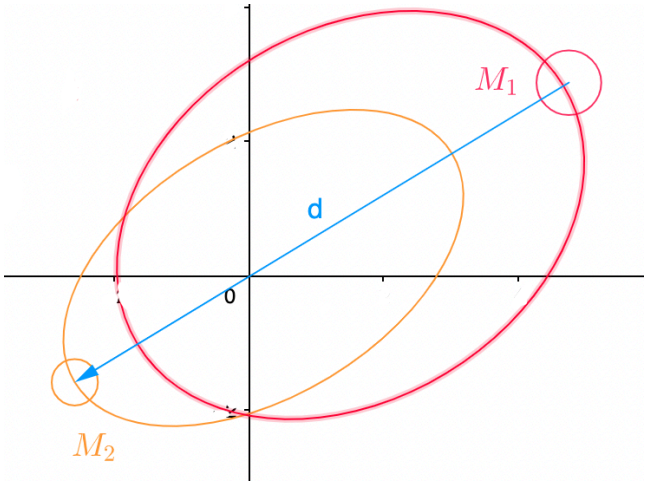

Figure 1.3: Elliptical orbits of the two stars in a binary.

Python programming for this problem will be described in Chapter 2.

2 Examples in Numerical Astronomy

This chapter assumes fundamental understanding of Python programming and focuses on the utilization of that understanding for problem solving in astrophysics and astronomy.

2.1 Libraries

We will be utilizing a number of that have functions defined for astrophysical problem solving which will prove beneficial for our programs. The standard libraries include `NumPy`, `Math`, `Pandas` and `Mathplotlib`. There are also libraries like `AstroPy` and `SciPy` [8].

`SciPy` helps us obtain the numerical values of various physical constants. The library `AstroPy` will mostly help us obtain the data of coordinates, positions and periods etc. of various astronomical objects.

2.2 Length of the Diurnal Arc

Let us begin with our first problem: Determining the length of a star's diurnal arc for a particular geographical location of the earth.

Diurnal arc is the path an object appears to follow in the sky as the earth completes its one rotation. The diurnal arc of the sun would be its trajectory as it (roughly) moves from the east to the west rising at the dawn and setting in the dusk. Similarly, for your given latitude, various stars can be seen to have a unique trajectory with respect to your position. This trajectory is termed as the diurnal arc and we will be utilizing this information to obtain the time period for which a particular star stays in the sky and also the maximum altitude it appears at for that location.

The length of the arc is given by,

$$h = \frac{1}{2}\cos^{-1}[\tan(\delta)\tan(\phi)]. \tag{12}$$

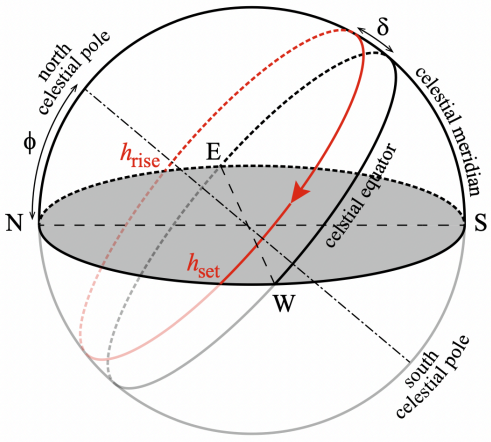

Figure 2.1: Diurnal Arc of an object moving around the celestial sphere (Schmidt, fig. 2.3, p28 [7]).

Here ϕ is the latitude of earth location that you are viewing the star from. The red circle in the figure above represents the diurnal arc of an object and δ is the declination of the star. Declination of an object is defined as its angular distance from the equatorial plane of the earth. Similarly, there is another quantity that we use to completely describe the location of a star in the sky: right ascension of a star is the angular distance measured eastward along the celestial equator at the time of March equinox.

Now we have sufficient knowledge of the theory to write a program to determine the length of the Diurnal Arc.

Let us do it for the brightest star of the constellation Lyra, Vega. We can use the library `AstroPy` to obtain its coordinates as follows:

```
from astropy.coordinates import SkyCoord, EarthLocation, AltAz, get_sun
Vega = SkyCoord.from_name("Vega")
```

We have now created the variable "Vega" that is coordinate class containing the Declination and Right Ascension of Vega. Use the print(Vega) command to obtain these values:

```
SkyCoord (ICRS): (ra, dec) in deg
(279.23473479, 38.78368896)
```

We can extract the value of the declination to our use as follows:

```
delta=Vega.dec
```

Next, we need to create a variable that contains the coordinates of Earth where we need to observe the diurnal arc of Vega from. This can be done as follows:

```
import astropy.units as u
LUMS = EarthLocation(lat=31*u.deg+28*u.arcmin+20*u.arcsec,
                                lon=74*u.deg+24*u.arcmin+40*u.arcsec)
phi=LUMS.lat
```

We now have both the necessary components required for our calculation and we can proceed ahead.

```
import math
x=-math.tan(delta.radian)*math.tan(phi.radian)
```

We have to keep one thing in mind that $-90^\circ \leq \Delta + \Phi \leq 90^\circ$. Otherwise, the stars have such low/high altitudes Δ that they cannot be seen from the sky of a location with latitude ϕ. Moreover, this will also trouble our arccos function obtained from the library math. In case of Vega and LUMS, it is not a problem since $\Delta + \Phi \approx 70^\circ$.

Finally, we obtain the length of the Diurnal Arc as follows:

```
h=math.acos(x)
T = (math.degrees(2*h)/360) #Converts radians to degrees
T=T*u.sday #converts degrees to hour format out of 24
print("T = {:.2f}".format(T.to(u.h)))
T = 15.89 h
```

Thus, Vega stays in the sky as seen from LUMS for 15 hours and 54 minutes.

2.3 Length of a day through the year

Let us utilize our knowledge of computational Python and astronomy to create a chart that details on how the days change in length over the course of a year for any geographical location on the earth.

Understanding the theory of altitude of the sun requires the knowledge of how the altitude of the sun varies for any particular location on the earth. If you are in the northern hemisphere, the length of a day increases during the summer season that lasts from late March to late September having peaked on the summer solstice (June 21). Then there is a dip in the length of a day during winters with the winter solstice (December 21) being the shortest day of the year. The very opposite is true the southern hemisphere where the seasons are reverted with the summers lasting between September and March and vice versa.

The obliquity ($\epsilon_0 = 23.44°$) of the earth's elliptic, that is, the inclination of earth's rotation plane with respect to its spin plane is responsible for the changing of seasons throughout the year. We do not need understand the exact details of earth's motion to make sense of this trend.

The annual variation in sun's declination is given by the following relation[10]:

$$\delta_\odot = -\sin^{-1}[\sin(\epsilon_0)\cos\left(\frac{360°}{365.24}(N+10)\right)]. \tag{13}$$

Here N represents the number of day for a year (January 10 being N=10) for which we are obtaining the declination.

Before we proceed, the distinction between declination and altitudes of a Star needs to be understood. The declination, as defined above, is the angular displacement of a celestial object from the equator. This means that a star of declination, say, $31.47°$ should pass directly overhead once a day, if viewed from LUMS (latitude$= 31.47°$N). So, the declination is a standardized coordinate that remains unchanged wherever you are around the globe. And it is your position on the globe that determines the local height of that object in the sky. This "local height" or more precisely, angular distance from the horizon on an object for a particular latitude is called the altitude of that object. So, the same star that is visible directly overhead at the equator, must be visible at an altitude of $58.53°$ in the LUMS skies. This can be determined by the simple relation of

$$Alt = 90° - (\text{Lat} - \delta) \text{ (if Lat} > \delta)$$
$$Alt = 90° + (\text{Lat} - \delta) \text{ (if Lat} < \delta).$$

The same phenomenon governs the variation in the altitude of the Sun with the change in its declination. Next we write a program that creates an array of the altitude of the Sun for a given location on Earth. Then we would convert it to the number of hours and thus you obtain the variation of the length of a day throughout the year.

```
import numpy as np
N = np.arange(0,365) # array with elements 0,1,2,...,364
omega = 2*math.pi/365.24 # Earth's angular velocity in rad/day
ecl = math.radians(23.44) \# obliquity of the ecliptic
```

Next we use equation (13) to obtain an array with the declination of the Sun varying throughout the year.

```
delta = -np.arcsin(math.sin(ecl) * np.cos(omega*(N+10)))
```

So, to obtain the diurnal arc, we use the same equation,
$h = \frac{1}{2}\cos^{-1}[\tan(\delta)\tan(\phi)]$.

First, we declare a new variable and apply a couple of if conditions as follows:

```
x=-np.tan(delta) * math.tan(phi.radian)
```

```
for i in range(len(x)):
  if x[i] \< -1:
    x[i]=-1
  if x[i] \> 1:
    x[i]=1
```

To understand this, consider the 171st day of the year, that is, June 21, the day of summer solstice in the northern hemisphere.
$\delta = -\sin^{-1}[\sin(23.44^\circ \times \cos(\frac{2\pi}{365.24}(171+10))]$
$\delta = 23.40^\circ$
Now, for $\phi = 80^\circ$
$x = \tan(23.40^\circ) \times \tan(80^\circ)$
$x = 2.45$

Now, if you take $\cos^{-1}()$ of this number, you realize that you get an error. This is because, mathematically, this number is outside the range of the cosine function, or the domain of the cosine inverse function.

Physically, on June 21, the sun does not set for any location on the earth above the latitude of 66.56°, called the arctic circle. So, if you input a location with coordinates above this latitude, our program would crash. Thus, we apply the catch, that whenever such an event would occur, the program would give us a value of 1, that would correspond to an arccos value of 90° and thus our program would provide us with the day length of 24 hours.

```
h=np.arccos(x)\\
T = (np.degrees(2*h)/360) * u.sday.to(u.h)\\
```

Now, we have the array T which contains the length of each for a year for any given location. If we insert the coordinates of LUMS, and plot this array versus the days array, we should get the following graph,

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(N,T, label="LUMS (32˚N)")

print("Minimum day length = {:5.2f} h".format(T.min()))
print("Maximum day length = {:5.2f} h".format(T.max()))
Minimum day length =  9.92 h
Maximum day length = 14.01 h
```

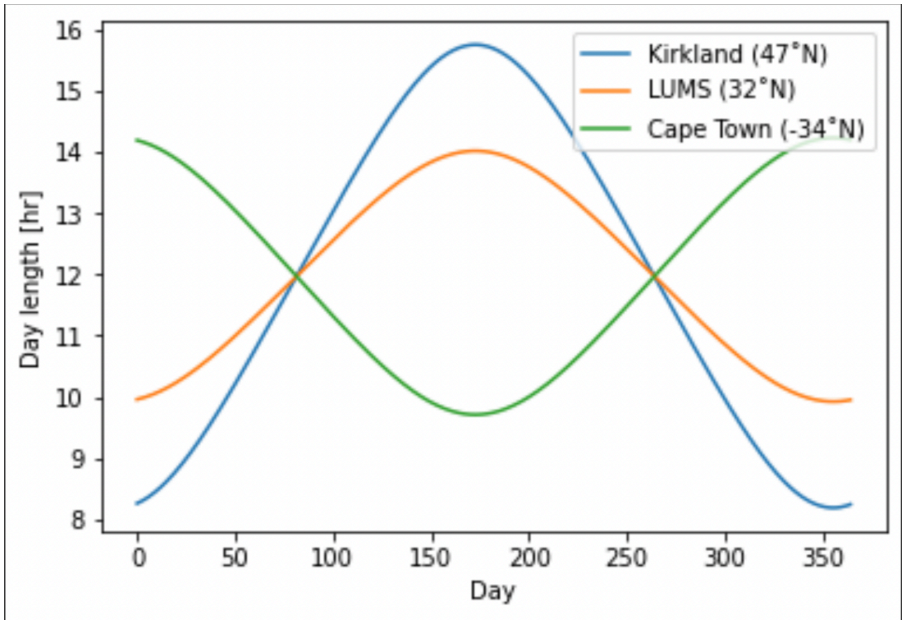

Figure 2.2: Altitude of the Sun throughout the year for LUMS (Lahore), Cape Town (34˚S) and Kirkland, WA (USA).

You can simultaneously plot multiple graphs (Figure 2.2) and compare how various latitudes have the length of their days vary.

Following are some interesting results:

1. All the locations in the northern hemisphere follow the same trend as LUMS: the length of the day peaking in mid-year (June 21) and having a minimum around late December.

2. All the locations in the southern hemisphere follow the opposite trend.

3. All locations on the equator have 12-hour long days all year long.

4. All locations above the arctic circle ($66.56°$) show a peculiar trend with days becoming longer than 24 hours in the summers and falling to 0 hours of sunlight during winters.

2.4 Trajectory of an object against the Sun

We realize that the stars change their altitude throughout the day for a given location as their azimuth angle also varies. Note that when the sun is out during the day, these stars are not visible. Therefore, we devise a program that can plot the varying altitude of these stars against the changing rising and setting sun so we have an idea of the best time for the observation of these stars.

We make use of the "AltAz" module of `astropy.coordinates` and use it to define our geographical position and time.

We start with defining a variable that encompasses the coordinates of our geographical location.

```
import astropy.units as u
from astropy.coordinates import SkyCoord, EarthLocation, AltAz, get_sun

# geographical position of the observer
LUMS=EarthLocation(lat=31*u.deg+28*u.arcmin+11*u.arcsec,
                                lon=- 74*u.deg+24*u.arcmin+42*u.arcsec)
phi=LUMS.lat
```

Next, we import a time format and adjust it to our time zone. Note that by default, the time is set to Coordinated Universal Time (GMT).

```
from astropy.time import Time

utc_shift = 5*u.hour # PST time zone (5h ahead)
local_time= Time("2022-03-25 00:00:00") + utc_shift
```

Next we will define our x-axis where we will have the 24 hours of time for a day. We take data points 5 minutes across for a smooth graph.

```
import numpy as np

# time array covering next 24 hours in steps of 5 min
elapsed = np.arange(0, 24*60, 5)*u.min
x_time = local_time + elapsed

frame_local_24h = AltAz(obstime=x_time, location=LUMS)
```

Next we import the coordinates of a star the we want to analyze and transform its data to our local frame (obtain its declination and right ascension). We do the same for the sun.

```
Vega = SkyCoord.from_name("Vega")
Vega_local = Vega.transform_to(frame_local_24h)

sun = get_sun(x_time)
sun_local = sun.transform_to(frame_local_24h)
```

Finally, we obtain the numerical data of our axes. We split the x-axis variable "elapsed" into 2 segments: when the sun is above the horizon (daytime) and when it is below the horizon (night time). Similarly, we make two variables `Vega_night` and `Vega_day` for the altitude of Vega during the night and day, respectively.

```
#Indicates the time when the Sun is below the horizon
elapsed_night = elapsed[np.where(sun_local.alt < 0)]

#Indicates the altitude of Vega while there is night
Vega_night = Vega_local.alt[np.where (sun_local.alt < 0)]
#Indicates the altitude of Vega while there is day
Vega_day = Vega_local.alt[np.where(sun_local.alt >= 0)]
```

Lastly, now that the data import is all complete, we import the `matplot` library to plot out data.

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(elapsed.to(u.h), sun_local.alt, color='Orange', label='Sun')

plt.plot(elapsed.to(u.h), Vega_local.alt, color='red', linestyle=':', label='Vega (dayligh

plt.scatter(elapsed_night.to(u.h), Vega_night, s=2, color='red', label='Vega (night)')

plt.xlabel('Time since midnight [h]')
plt.xlim(0, 24)
plt.xticks(np.arange(13)*2)
plt.ylim(0, 90)
plt.ylabel('Altitude [deg]')
plt.legend(loc='upper center')
```

Note that for plotting `Vega_night`, we have used scatter plot instead of continuous line plot. This is to ensure that in case the sun rises above the horizon while the star is still up in the sky, we avoid the disruption the continuous line graph makes in our plot by connecting the last point before sunrise to the first point after sunset with a straight line.

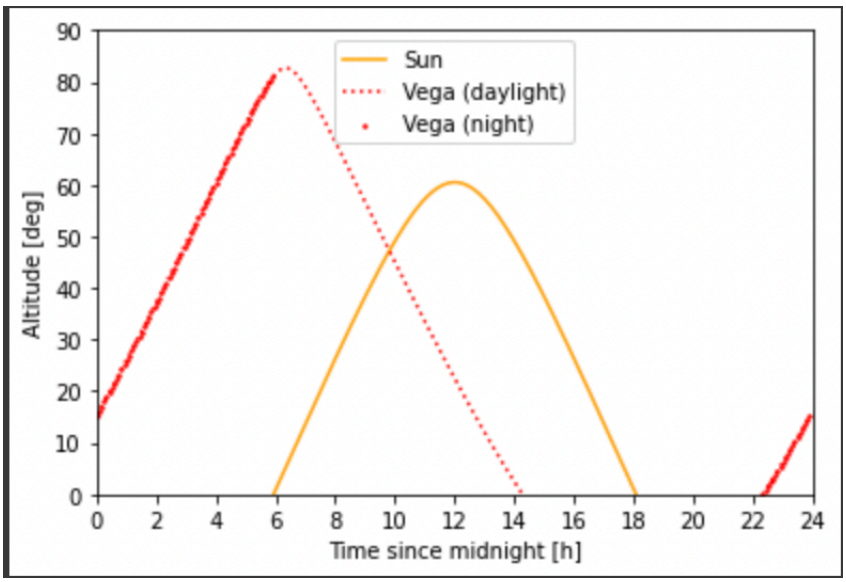

Figure 2.3(a): Vega's altitude against the sun as viewed from LUMS on March 25, 2022.

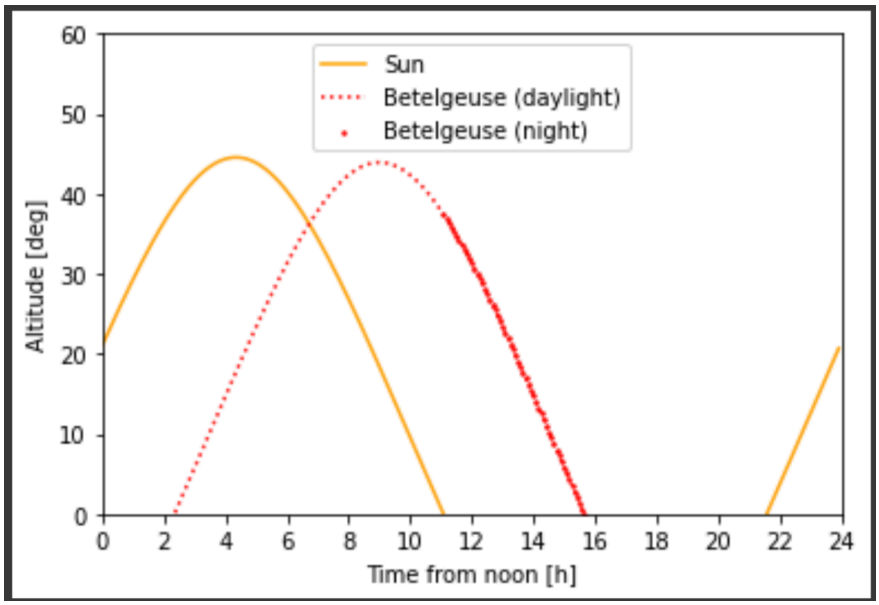

Figure 2.3(b): Betelgeuse's altitude against the sun as viewed from LUMS on April 10, 2022.

2.5 Orbital Mechanics

2.5.1 Two Star System

We start off with describing the python program that has the plot of the trajectory of two stars going in a mutual orbit as its output. We start off with importing the essential libraries and declaring the variables that contain the information of the stars' masses and the orbital parameters.

We start with the Sirius system that has planets A and B roughly twice and same the size of the sun, respectively.

```
import numpy as np
from scipy.constants import G,year,au
```

```
from astropy.constants import M_sun, M_earth
import matplotlib.pyplot as plt

M1 = 2.06*M_sun.value # mass of Sirius A
M2 = 1.02*M_sun.value # mass of Sirius B

a = 2.64*7.4957*au # semi-major axis of the 2 star system
e = 0.5914 # eccentricity of the two star system
```

The orbital time period for the system comes out as,

```
T = 2*np.pi * (G*(M1 + M2))**(-1/2) * a**(3/2)

print("Orbital period = {:.1f} yr".format(T/year))
```

We start off with describing the python program that has the plot of the trajectory of two stars going in a mutual orbit as its output. We start off with importing the essential libraries and declaring the variables that contain the information of the stars' masses and the orbital parameters.

We start with the Sirius system that has planets A and B roughly twice and same the size of the Sun, respectively.

```
import numpy as np
from scipy.constants import G,year,au
from astropy.constants import M_sun, M_earth
import matplotlib.pyplot as plt

M1 = 2.06*M_sun.value # mass of Sirius A
M2 = 1.02*M_sun.value # mass of Sirius B

a = 2.64*7.4957*au # semi-major axis of the 2 star system
e = 0.5914 # eccentricity of the two star system
```

Next we calculate the orbital period using Kepler's third law [9],

$$T = \frac{2\pi}{\sqrt{G(M_1 + M_2)}} a^{3/2}. \tag{14}$$

```
T = 2*np.pi * (G*(M1 + M2))**(-1/2) * a**(3/2)

print("Orbital period = {:.1f} yr".format(T/year))
```

One can play around with the orbital period by changing the defined semi-major axis and the masses of the stars.
Next, we are required to solve a system of coupled differential equations to obtain an array that can be regarded as the ephemeris of the two stars. As we had in equation (10),

$$\frac{d\vec{v}_1}{dt} = \frac{GM_2}{|r_2 - r_1|^3}(\vec{r}_2 - \vec{r}_1),\ \frac{d\vec{v}_2}{dt} = -\frac{GM_1}{|r_2 - r_1|^3}(\vec{r}_2 - \vec{r}_1).$$

Next we compute the integral of the whole orbital period of Neptune Euler method

```
n_rev = 3 # number of revolutions
n = n_rev*500 # number of time steps
dt = n_rev*T/n # time step
t = np.arange(0, (n+1)*dt,dt)
```

The number of revolutions can be changed and this would correspondingly, change the total number of time steps we would have over which the integration is being performed.

Next we start off by initializing the position vectors and the orbital velocity vectors. We keep them of the dimension of $n+1$ where n is the total number of steps. The zeroth index will have our initial values which we define as follows:

```
# periastron distance and relative velocity
d=a*(1+e)
v = np.sqrt(G*(M1 + M2)*(2/d - 1/a)) # vis-viva equation

#Initial values of x-y coordinates for each star
x1[0], y1[0] = d*M2/(M1 + M2), 0
x2[0], y2[0] = -d*M1/(M1 + M2), 0

#Initial values of vx-vy velocities for each star
vx1[0], vy1[0] = 0, -v*M2/(M1 + M2)
vx2[0], vy2[0] = 0, v*M1/(M1 + M2)

alpha = G*M1*M2
```

Proceeding, we define the distance and force vectors

```
for i in range(n):
  delta_x = x2[i] - x1[i]
  delta_y = y2[i] - y1[i]

  # third power of distance
  d3 = (delta_x**2 + delta_y**2)**(3/2)

  # force components
  Fx = alpha*delta_x/d3
  Fy = alpha*delta_y/d3
```

Finally we define our state vectors that help us calculate the derivative of our variables and the tools we required for integration are now complete.The vector notation is as follows:

$$\vec{s} = \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \end{pmatrix} \quad \vec{f}(t,s) = \begin{pmatrix} v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \\ F_{12x}/M_1 \\ F_{12y}/M_1 \\ -F_{12x}/M_2 \\ -F_{12y}/M_2 \end{pmatrix}$$

Here,

$$\frac{d\vec{s}}{dt} = \vec{f}(t, s) \tag{15}$$

Thus, we can now import the `solve_ivp()` function from `scipy.import`, define a function to determine the derivative of state vectors, and insert our variables,

```
from scipy.integrate import solve_ivp

init_state = np.array([ x1[0], y1[0], x2[0], y2[0],
                    vx1[0], vy1[0], vx2[0], vy2[0]])

def state_derv(t, state):
    alpha = G*M1*M2

    delta_x = state[2] - state[0] # x2 - x1
    delta_y = state[3] - state[1] # y2 - y1

    # third power of distance
    d3 = (delta_x**2 + delta_y**2)**(3/2)

    # force components
    Fx = alpha*delta_x/d3
    Fy = alpha*delta_y/d3

    return np.array([state[4], state[5], state[6], state[7],
                Fx/M1, Fy/M1, -Fx/M2, -Fy/M2])

tmp = solve_ivp(state_derv, (0,3*T), init_state,
dense_output=True)
data = tmp.sol(t)
```

The final ephemeris data for both our orbiting bodies is contained in the variable data in the sequence defined in the return argument of the `state_derv` function. Once we apply the `solve_ivp()` function on our data with our initial values as defined above by using the derivative function of the state vector, we will have the instantaneous values of the x-y position coordinates and the orbital velocity vectors of both the bodies. This can then be plotted and the figure 2.4 is obtained.

```
fig = plt.figure(figsize=(6, 6*25/35), dpi=150)

plt.plot([0], [0], '+k') # center of mass
plt.plot(data[0,:]/au, data[1,:]/au,
       color='red', label='Sirius A')

plt.plot(data[2,:]/au, data[3,:]/au,
        color='blue', label='Sirius B')
# plt.plot(data[0,0],data[0,0],"o")
# plt.plot(data[2,0],data[2,0],"x")

plt.xlabel("$x$ [AU]")
plt.xlim(-22.5,12.5)
```

```
plt.ylabel("$y$ [AU]")
plt.ylim(-12.5,12.5)
plt.legend(loc='upper left')
plt.savefig("sirius_scipy.png")
```

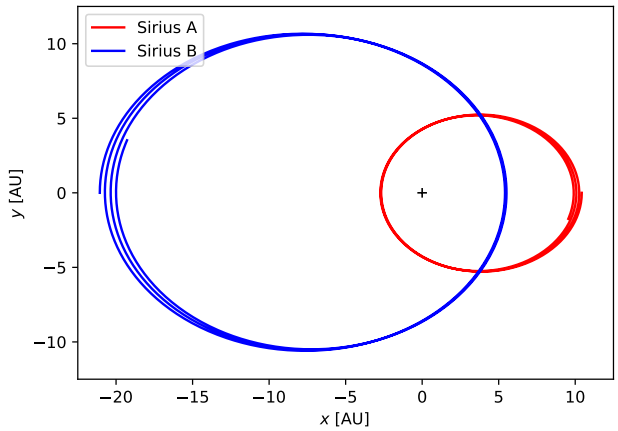

Figure 2.4: The orbital motion of Sirius A and B in x-y plane over three revolutions.

It can be realized that the orbits being obtained are not closed with the semi-major axis continuously shrinking. This has to do with the definition of the `solve_ivp()` function whereby it continuously adjusts the tolerances in time step to compute the integral efficiently. Besides that, this section provides us with a notion of how the orbital mechanics works for a binary star system with two stars with a comparable size. We can change the variable values and see how the change in stellar masses, the semi-major axis and the eccentricity affects our orbits.

2.5.2 Two Stars with a Hypothetical Planet

Next we introduce a hypothetical planet into our system whose mass is small enough to not cause any disturbances in the orbits of the two stars, but it has its path influences by the gravity of both these stars. The system is modelled in the same way as the two star system above, except that we define a third body of a small mass that has its own orbit of eccentricity e_3.

```
M1 = 2.06*M_sun.value # mass of Sirius A
M2 = 1.02*M_sun.value # mass of Sirius B
M3 = 10.2*M_earth.value # mass of planet

a = 2.64*7.4957*au # semi-major axis of the 2 star system
e = 0.5914 # eccentricity of the two star system

e3 = 0.3 #for the planet
```

The data variables remain the same for the 2-star systems and there is an introduction of four new arrays that will have the data of the x-y position coordinates and the orbital velocity vector of this new planet.

```
# data arrays for coordinates
x3 = np.zeros(n+1)
```

```
y3 = np.zeros(n+1)

# data arrays for velocity components
vx3 = np.zeros(n+1)
vy3 = np.zeros(n+1)
```

We can start off by setting the semi-major axis of this planet-stars system at 25 AU and keep on changing it to see how the orbit gets affected. We also define the initial position with the planet starting off its motion around the smaller star, Sirius B.

```
M12=M1+M2

a123 = 25*au # semi-major axis of the planets-stars system

d123=a123*(1-e3)
v123 = np.sqrt(G*(M12 + M3)*(2/d123 - 1/a123))

x3[0], y3[0] = -d123*M12/(M12 + M3), 0
vx3[0], vy3[0] = 0, v123*M12/(M12 + M3)
```

Finally, we make some changes to the state derivative computer function to account for this new planet, and our integration tools will be complete.

```
init_state2 = np.array([x1[0], y1[0], x2[0], y2[0], x3[0], y3[0],
                    vx1[0], vy1[0], vx2[0], vy2[0], vx3[0], vy3[0]])

def state_derv2(t, state):
  alpha = G*M1*M2
  beta = G*M1*M3
  gamma = G*M2*M3
  delta12_x = state[2] - state[0] # x2 - x1
  delta12_y = state[3] - state[1] # y2- y1

  delta13_x = state[4] - state[0] # x3 - x1
  delta13_y = state[5] - state[1] # y3- y1

  delta23_x = state[4] - state[2] # x3 - x2
  delta23_y = state[5] - state[3] # y3- y2

  # force components
  F12x = alpha*delta12_x/(delta12_x**2 + delta12_y**2)**(3/2)
  F12y = alpha*delta12_y/(delta12_x**2 + delta12_y**2)**(3/2)

  F13x = beta*delta13_x/(delta13_x**2 + delta13_y**2)**(3/2)
  F13y = beta*delta13_y/(delta13_x**2 + delta13_y**2)**(3/2)

  F23x = gamma*delta23_x/(delta23_x**2 + delta23_y**2)**(3/2)
  F23y = gamma*delta23_y/(delta23_x**2 + delta23_y**2)**(3/2)

  return np.array([state[6], state[7],
```

```
                state[8], state[9],
                state[10], state[11],
                ( F12x + F13x)/M1, ( F12y + F13y)/M1,
                (-F12x + F23x)/M2, (-F12y + F23y)/M2,
                (-F13x - F23x)/M3, (-F13y - F23y)/M3])
```

Finally we apply the `solve_ivp()` function and plot our data points.

```
tmp2 = solve_ivp(state_derv2, (0,3*T), init_state2,
             dense_output=True)
data2 = tmp2.sol(t)

fig = plt.figure(figsize=(6, 6*25/35), dpi=150)

plt.plot([0], [0], '+k') # center of mass
# plt.plot(data[4,0], data[5,0], '+k') #Starting point of the planet


plt.plot(data2[0,:]/au, data2[1,:]/au,
       color='red', label='Sirius A')

plt.plot(data2[2,:]/au, data2[3,:]/au,
        color='blue', label='Sirius B')

plt.plot(data2[4,:]/au, data2[5,:]/au,
         color='green', label='Planet')

plt.xlabel("$x$ [AU]")
plt.xlim(-30.5,15.5)
plt.ylabel("$y$ [AU]")
plt.ylim(-12.5,12.5)
plt.legend(loc='upper left')
plt.savefig("sirius planet.png")
```

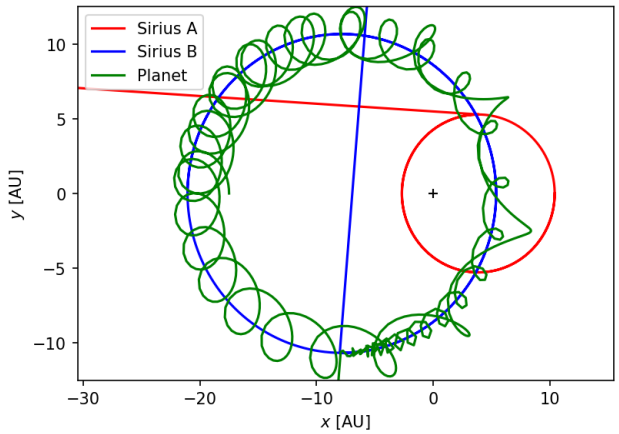

Figure 2.5(a): The orbital motion of Sirius star system with a planet of semi-major axis of 25 AU in x-y plane over three revolutions.

When the semi-major axis is changed to 200 AU, the following figure (2.5(b)) is obtained:

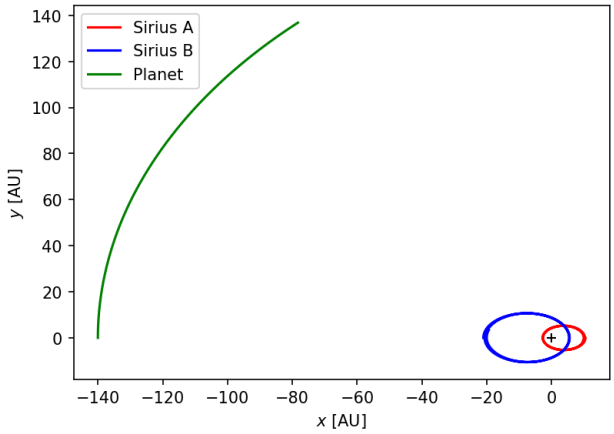

Figure 2.5(b): The orbital motion of Sirius star system with a planet of semi-major axis of 200 AU in x-y plane over three revolutions.

3 Geometric Discovery of Neptune

3.1 Methods

As previously mentioned in the Introduction, Neptune was first predicted through mathematical calculations before being actually sighted through a telescope. [1] devised a computational mathematical model that can be used for a rediscovery of Neptune.

As shown in equations (4) and (5) the vector $\vec{V}(t)$ is the component of Uranus's acceleration that can only be explained in terms of the gravitational force of Neptune. Had Neptune not existed, $|\vec{V}(t)|$ would have been zero. The magnitude of $\vec{V}(t)$ is maximum when the sun, Uranus and Neptune form a conjunction because it maximizes $\frac{1}{|\vec{r}_N - \vec{r}_U|^3}$.

From the data it can be ascertained that $|\vec{V}|$ peaked in late 1822 and mid 1994 (Figure 1.1) which can indicate the conjunctions of the Uranus, Neptune and sun system.

Note that while plotting the data, one is making the assumption that the planets are all moving in co-planar circles (Figure 1.2).

The first term in Eq. (3) demands the acceleration of Uranus. It is to be noted that the data available provides the numerical values of the velocity of Uranus with a resolution of data spaced 1 day apart. We differentiate this data to obtain the acceleration using the 5 point differentiation method:

$$\frac{d^2\vec{r}_U}{dt^2} = \frac{1}{12h}(v_U(i-2) - 8v_U(i-1) + 8v_U(i+1) - v_U(i+2)). \tag{16}$$

v_u is the python vector with the values of velocity of Uranus at different instants. h is the time difference between two consecutive values. If you take data 30 days apart, h comes out as $30/365.24$ years.
The value of G needs to be obtained in the units of kg $(\text{AU})^3/(\text{year})^2$ given by,

$$G = 2.95 \times 10^{-18} \frac{\text{kg } (\text{AU})^3}{(\text{year})^2}.$$

3.2 Synodic Period

To calculate the orbital period of Neptune, one introduces another quantity $\vec{p}(t)$ that is in the x-y plane and always perpendicular to $\vec{r}_U$.

$$\vec{p}(t) = \hat{z} \times \vec{r}_u(t). \tag{17}$$

One makes the following two observations:

1. $\vec{V}(t)$ points in the direction of Uranus in case of both the conjunction and opposition of Uranus and Neptune.
2. $\vec{p}(t)$ is perpendicular to $\vec{r}_U$ and will thus be perpendicular to $\vec{V}(t)$ too in case of opposition and conjunction.

Therefore, one introduces a scalar quantity $\chi(t)$,

$$\chi(t) = \vec{V}(t).\vec{p}(t). \tag{18}$$

From the second observation, one can see that $\chi(t)$ will be 0 in case of conjunctions and oppositions of Uranus and Neptune. This can thus help us obtain the synodic period of Neptune with respect to Uranus as it will provided by the time difference between any two conjunctions.

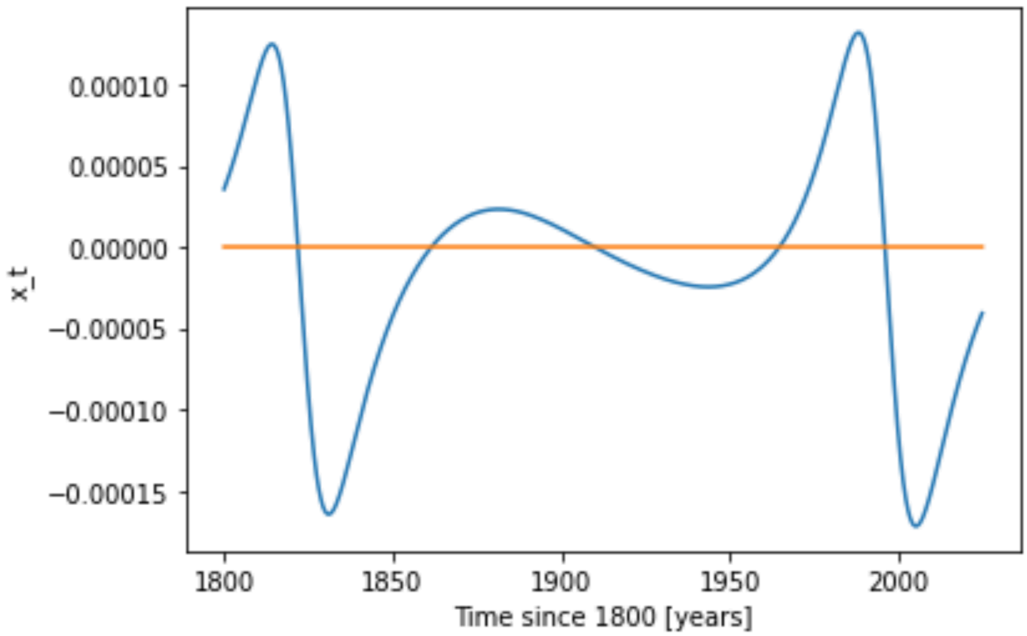

Figure 3.1: $\chi(t)$ versus time since 1800.

Upon plotting the figure, one can see that one has obtained five roots for $\chi(t)$. First, third, and fifth roots correspond to the first conjunction, opposition, and second conjunction, respectively (Figure 3.1). The roots in 1861 and 1964 correspond to the projections of $\vec{p}(t)$ and $\vec{V}(t)$ cancelling each other.

- First Conjunction (C_1): 1821
- Opposition: 1909
- Second Conjunction (C_2): 1995

The calculation of these events on python assumes each year to be of exactly 360 days. Thus to obtain an accurate value of the synodic period, one do the following calculations:

$$C = C_2 - C_1$$
$$T_{sy} = C(1 - \tfrac{5}{365.24}).$$

Thus, the synodic period of Neptune with respect to Uranus comes out as 171 years.

3.3 Sidereal Period and Semi-major axis

Sidereal period is the time taken by a planet to complete one cycle around the Sun. If Neptune and Uranus had the same angular velocities while going around the Sun, the Synodic period would be the same as the Sidereal period for Neptune. But since that is not the case, and Uranus, being in a smaller orbit, has a shorter sidereal period, the sidereal period of Neptune comes out as slightly shorter than the synodic period calculated above. one define a new quantity $\xi(t)$. First define the normal vector to the orbital plane of Uranus,

$$\vec{n}_U = \vec{r}_U \times \frac{d\vec{r}_U}{dt}$$

$$\xi(t) = \vec{V}(t).\vec{n}_U \tag{19}$$

The variable $\xi(t)$ gives the projection of $\vec{V}(t)$ on $\vec{n}_U$. $\xi(t)$ allows us to move beyond the assumption of the planets moving in co-planar orbits. Since for co-planar orbits, $\xi(t)$ would come out as zero.

$\xi(t)$ will be zero at conjunctions and oppositions alongside having 2 other roots like $\chi(t)$.

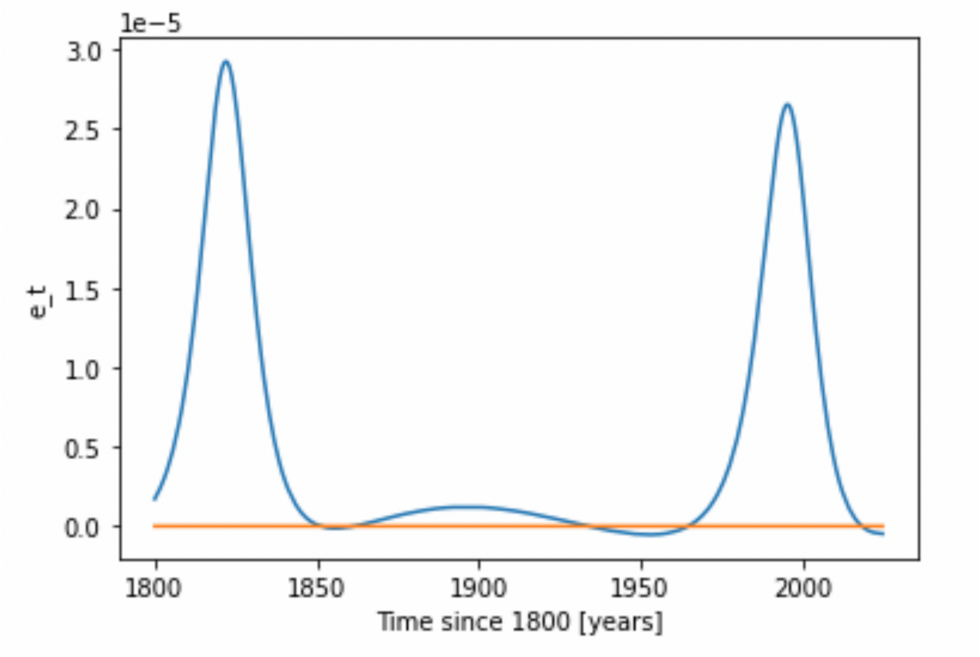

Figure 3.2: $\xi(t)$ versus time since 1800.

From the nth and (n+4)th roots of $\xi(t)$ in the same manner as for $\chi(t)$, one can obtain the Sidereal period of Neptune that comes out as 165 years.

Next, to obtain the semi-major axis of Neptune, one can put to use Kepler's third law of planetary motion [9].

$$a_N^3 = cT_N^2. \tag{20}$$

Since, the sidereal period is given in years, c can be taken as 1 to obtain the semi-major axis in AU. a_N comes out as 30.05 AU.

3.4 Mass

Employing equation 2 can help obtain the mass of Neptune.

$$M_N = \frac{1}{G}\left(\frac{\vec{V}}{\frac{\vec{r}_N - \vec{r}_U}{|\vec{r}_N - \vec{r}_U|^3} - \frac{\vec{r}_N}{|r_N|^3}}\right). \tag{21}$$

A python program would thus provide you with an array of values for M_N. Thus, upon averaging it, you have the mass of Neptune come out as 1.024×10^{26} kg.

3.5 Python Program

To begin programming in Python, we begin with importing our data first. The ephemeris data we are emloying in our model is obtained from NASA's Jet Propulsion Laboratory (JPL) at Caltech. The data can be downloaded from the link shared in reference [2]. The modified data in xlsx file available at [3] has the x, y and z coordinates for the respective planet as well as the x, y and z components of their velocities.

```
import pandas as pd
import numpy as np

#Importing the position vectors of the 4 planets.
Uranus=pd.read_excel("Uranus ephimeres.xlsx", header=0, index_col=0, skiprows=[0],
    usecols=[1,2,3,4])
Saturn=pd.read_excel("Saturn ephimeres.xlsx", header=0, index_col=0, skiprows=[0],
    usecols=[1,2,3,4])
Jupiter=pd.read_excel("Jupiter ephimeres.xlsx", header=0, index_col=0, skiprows=[0],
    usecols=[1,2,3,4])

Uv=pd.read_excel("Uranus ephimeres.xlsx", header=0, index_col=0, skiprows=[0],
    usecols=[4,5,6,7])
```

The variables thus declared are essentially the position vectors of Uranus, Saturn and Jupiter. While the variable "Uv" is the velocity vector of Uranus. This last variable is essential for the determination of the acceleration of Uranus which is the first term in equation (5). The position vectors have the units of AU while the velocities are in AU/day.
Next we convert these variables into NumPy arrays to apply further operations on them.

```
#Converting it to an array here
Neptune=Neptune.to_numpy()
Uranus=Uranus.to_numpy()
Saturn=Saturn.to_numpy()
Jupiter=Jupiter.to_numpy()
Uv=Uv.to_numpy()
Uv=Uv*365.24218750 #To convert from au/day to au/year
```

Here the last term takes the Uranus velocity given in AU/day to AU/year.
We begin with the calculations of the second term from equation (5).

$$\vec{V}(t) = \frac{d^2\vec{r}_U}{dt^2} + G(M_{\odot} + M_U)\frac{r_U}{|r_U|^3} - \sum_{j\neq\odot,U,N}^{9} GM_j\left(\frac{\vec{r}_j - \vec{r}_U}{|\vec{r}_N - \vec{r}_U|^3} - \frac{\vec{r}_j}{|r_j|^3}\right).$$

```
M_S=1.989e30 #Mass of the Sun
M_U=8.681e25 #Mass of Uranus
M_St=5.683e26
M_J=1.898e27

#############
#Second Term
#############
Vv23=((Ux)**2+(Uy)**2+(Uz)**2)**(1/2)
```

```
V2_2_3= np.tile(Vv23, (3,1))
V2_2_3=np.transpose(V2_2_3)

V2_2=G*(M_S+M_U)*Uranus/(V2_2_3**3)
```

The function `np.title()` duplicates the vector one provides as input and creates a new array with each of the columns that are the same as the argument vector. This helps us easily divide each column of the position vector of Uranus by the magnitude cube of the same vector as required in the equation.

Next we evaluate the third term which includes the acceleration part of Neptune due to the inner planets. Although the sum is over all the planets except Uranus, we perform it only for Jupiter and Saturn. For a greater precision one can also include the inner planets, but these are far away from Neptune and have such small masses that they have no practical gravitational pull on Neptune.

```
#Saturn
Sx,Sy,Sz=Saturn.T

Sv2=((Sx-Ux)**2+(Sy-Uy)**2+(Sz-Uz)**2)**(3/2)
S2= np.tile(Sv2, (3,1))
S2=np.transpose(S2)

Sv3=((Sx)**2+(Sy)**2+(Sz)**2)**(3/2)
S3= np.tile(Sv3, (3,1))
S3=np.transpose(S3)

S=G*M_St*((Saturn-Uranus)/S2-Saturn/S3)

#Jupiter
Jx,Jy,Jz=Jupiter.T

Jv2=((Jx-Ux)**2+(Jy-Uy)**2+(Jz-Uz)**2)**(3/2)
J2= np.tile(Jv2, (3,1))
J2=np.transpose(J2)

Jv3=((Jx)**2+(Jy)**2+(Jz)**2)**(3/2)
J3= np.tile(Jv3, (3,1))
J3=np.transpose(J3)

J=G*M_J*((Jupiter-Uranus)/J2-Jupiter/J3)

V2_3=J+S
```

Next we obtain the acceleration term for Uranus. This has been the most tricky part of this paper and we have to achieve extreme preciseness in this calculation to obtain the required results.

Different methods for this calculation can be tried. We use the 5 point differentiation method that makes use of two neighbouring points on each side of a given point and calculate the change in the function at that point.

$$\frac{\partial f}{\partial x} = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h}, \tag{22}$$

where h is the difference between two consecutive point coordinates relative to which the derivative is being calculated.

```
h=30/365.24218750 #We divide it by 365.24 to convert it to years

XX=np.zeros(2698)
V2_1=np.tile(XX,(3,1))
#Print("Datatype:",V2_1.dtype)
V2_1=np.transpose(V2_1)
for i in range(2696):
    if i==0:
      continue
    if i==1:
      continue
    V2_1[i]=1/(12*h)*(Uv[i-2]-8*Uv[i-1]+8*Uv[i+1]-Uv[i+2])

#Hard coding the first and last 2 values in the array
#V2_1=V2_1
V2_1[0]=V2_1[2]
V2_1[1]=V2_1[2]
V2_1[2697]=V2_1[2695]
V2_1[2696]=V2_1[2695]
```

Now that we have calculated the individual terms, we add them all to obtain the array for $\vec{V}(t)$. Lastly, we obtain its amplitude and plot it vs. time.

```
import matplotlib.pyplot as plt

V_RHS=V2_1+V2_2-V2_3
Vrx,Vry,Vrz=V_RHS.T
Vr_mag=((Vrx)**2+(Vry)**2+(Vrz)**2)**(1/2)
plt.plot(Time,Vr_mag,label="V versus Time")

plt.xlabel("Time since 1800 [months]")
plt.ylabel("V")
plt.legend(loc="upper center")
```

Next we use the array of $\vec{V}(t)$ and obtain the arrays for $\chi(t)$ and $\xi(t)$

```
Z=np.zeros(2698) #z-axis
Z=np.tile(Z,(3,1))
Z=np.transpose(Z)
for i in range(2698):
  Z[i,2]=1

p_t=np.cross(Z,Uranus)
px,py,pz=p_t.T
x_t=Vx*px+Vy*py+Vz*pz
```

```
plt.plot(Time,x_t)

Zero=np.zeros(2698)
Zero=np.array(Zero)
plt.plot(Time,Zero)
plt.xlabel("Time since 1800 [years]")
plt.ylabel("$x_t$")
```

To obtain the roots of $\chi(t)$ we must apply a number of if-else statements keeping in mind that at no point our function may exactly be zero, but we can indicate the point at which the graph crossed the $\chi(t) = 0$ line.

```
#Finding the roots of x_t
C1=0
a=0
O1=0
b=0
C2=0
t=0
for i in range(2695):

  if x_t[i]<0 and x_t[i+1]>0:
    if t==0:
      C1=Time[i]
      t=t+1
    elif t==1:
      a=Time[i]
      t=t+1
    if t==2:
      O1=Time[i]
      t=t+1
    elif t==3:
      b=Time[i]
      t=t+1
    elif t==5:
      C2=Time[i]
      t=t+1


  if x_t[i]>0 and x_t[i+1]<0:
    if t==0:
      C1=Time[i]
      t=t+1
    elif t==1:
      a=Time[i]
      t=t+1
    if t==2:
      O1=Time[i]
      t=t+1
```

```
    elif t==3:
      b=Time[i]
      t=t+1
    elif t==4:
      C2=Time[i]
      t=t+1

print("First Conjunction: ",C1)
#print(a)
print("First opposition: ",b)
#print(b)
print("Second Conunction: ",C2)

C=C2-C1
C=(1-5/365.24218750)*C

print("The Synodic Period of Neptune with respect to Uranus is :",C," years")
```

In a similar manner, we declare $\xi(t)$ and calculate its roots.

```
Nu=np.cross(Uranus,Uv)

Nux,Nuy,Nuz=Nu.T
E_t=Vx*Nux+Vy*Nuy+Vz*Nuz
plt.plot(Time,E_t)
plt.plot(Time,Zero)
plt.xlabel("Time since 1800 [years]")
plt.ylabel("e_t")
```

The roots are calculated as follows:

```
#Finding the roots of E_t
D1=0
c=0
E1=0
d=0
D2=0
t=0
for i in range(2695):

  if E_t[i]<0 and E_t[i+1]>0:
    if t==0:
      D1=Time[i]
      t=t+1
    elif t==1:
      c=Time[i]
      t=t+1
    if t==2:
      E1=Time[i]
      t=t+1
```

```
    elif t==3:
      d=Time[i]
      t=t+1
    elif t==5:
      D2=Time[i]
      t=t+1


  if E_t[i]>0 and E_t[i+1]<0:
    if t==0:
      D1=Time[i]
      t=t+1
    elif t==1:
      c=Time[i]
      t=t+1
    if t==2:
      E1=Time[i]
      t=t+1
    elif t==3:
      d=Time[i]
      t=t+1
    elif t==4:
      D2=Time[i]
      t=t+1

D=D2-D1
D=(1-5/365.24218750)*D
print("The sidereal period for Uranus is: ",D)
```

Now that we have the sidereal period, we can calculate the semi-major axis of Neptune using Kepler's third law.

```
a_u=D**(2/3)
print("Uranus's semi-major axis in AU is:",a_u)
```

The semi-major axis is in AU.

Finally, the mass term is calculated as given by equation (17),

```
#Finding the Mass of Neptune:
VVV=V1/V2-Neptune/V3
VVVx,VVVy,VVVz=VVV.T
VVV_mag=((VVVx)**2+(VVVy)**2+(VVVz)**2)**(1/2)

#This won't be a single value but an array of values
Mass_N=V_mag/(VVV_mag*G)

#We print the average of these values
print("The Mass of Neptune is: ",np.sum(Mass_N)/2698)
```

3.6 Conclusion

The orbital parameters of Neptune can be obtained through this computational program that utilizes the given data of position vectors and velocities of the other solar system planets. The Sidereal period can be very accurately determined through the information of conjunctions of the Uranus, Neptune, and Sun system which can thus lead to the accurate determination of the semi-major axis of the planet.

For Neptune the sidereal period was calculated to be **165 years**. Kepler's second law of planetary motion helped us determine its semi-major axis to be **30.05 AU**. The mass was came out as **1.024 × 1026 kg**.

This model can be extended to the discovery of exoplanets provided that you have information regarding the masses and orbital parameters of the other major planets in the stellar system.

3.7 Prospects

The goal of the computational reproduction of the research paper in discussion has been the prospective applications of the concerned model on other exosolar planetary systems. The paper provides us with a model that we can apply to any given exoplanet system and see if the stellar system has any more planets that are yet to be observed/discovered. By using the ephemeris data of the currently discovered planet, we can posit the existence of yet undiscovered planets in the system.

Once the experimental ephemeris data is obtained for the currently discovered planets of a stellar system, one can compare it with the simulated ephemeris generated data in section 2.5, and if there is any deviation, the geometric model here established can help determine the exoplanetary cause of this deviation.

Moreover, due to the analogous symmetry of the problem, we can also extend the geometric model to the discovery of planetary moons as well. Since planets with their moons geometrically act in the same manner as planets moving around a star, if we apply our model to these planet-moon systems, we can expect to discover new satellites that are orbiting these planets.

3.8 Limitations

- For the discovery and accurate determination of the orbital parameters of a planet, it is necessary to have information regarding the parameters of the other major planets in the stellar system.
- The model assumes that the system has at least two planets.
- The planet must have completed at least one circle around the star or its sidereal period and correspondingly, semi-major axis cannot be obtained.

4 Transit Light Curves

In this chapter we describe what is one of the most dominant methods for exoplanet discovery. We also develop the geometric tools that help us ascertain the orbital and system parameters and finally write the Python program to help us extract these parameters from observational data.

4.1 Method

Transit Light Curve method for explanatory detection is a high precision method that has so far helped discover over 4,000 exoplanets across our galaxy. The method relies on the fact that when a planet passes from in front of the star (eclipses the star) the light emitted from the star that is reaching the observer, reduces in intensity. The system is consequently referred to as an ”Eclipsing Binary”. This reduction in light when reoccurs over a period of time, indicates the presence of an exoplanet in the system under observation.

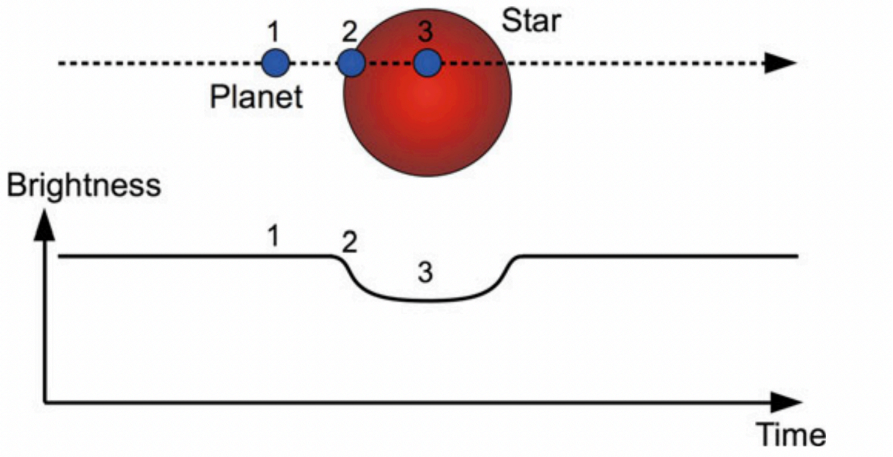

Figure 4.1:How a transiting exoplanet reduces the intensity of the star’s light.
(Schmidt, fig. 5.23, p189 [7])

Let F_0 be the flux of light reaching the Earth before the planet transits. Assuming a uniform surface brightness, the flux is proportional to the surface area of the star,

$$F_0 = cR_s^2.$$

Here R_s is the radius of the star.

Now, imagine a planet of radius R_p transits this star. The flux reaching the earth will now be reduced to,

$$F = c(R_s^2 - R_p^2).$$

The fraction reduction in the flux is thus given by,

$$\frac{\Delta F}{F_0} = \frac{F - F_0}{F_0}.$$

$$\frac{\Delta F}{F_0} = (\frac{R_p}{R_s})^2. \tag{23}$$

4.2 Size of the exoplanet

The orbital period of the star can very easily be calculated by timing the time difference between two consecutive transits. There are a number of eclipsing binaries visible in our night sky that have an orbital period of 2 to 5 days.

Once the orbital period P is experimentally determined, the semi-major axis can be obtained from Kepler's third law.

$$a = \sqrt[3]{\frac{P^2(M_s+M_p)}{k}}.$$

Since the planet's mass is very small as compared to that of the star, Kepler's law takes the form,

$$a = \sqrt[3]{\frac{M_s}{M_\odot}} P^{2/3}. \tag{24}$$

Once we have the semi-major axis and the period, we can use the following relation to determine the planet's size,

$$\sin\left(\frac{\pi T_{trans}}{P}\right) = \frac{R_s + R_p}{a}. \tag{25}$$

T_{trans} is the time the exoplanet takes to complete its transit of the star.

$$R_p(1 + \tfrac{R_s}{R_p}) = a \sin\big(\tfrac{\pi T_{trans}}{P}\big).$$

Since the transit time is much smaller than the orbital period of the exoplanet, we can use the small angle approximation on the sine function. Also, from equation (22),

$$R_p(1 + \sqrt{\tfrac{F_0}{\Delta F}}) \approx a(\tfrac{\pi T_{trans}}{P}).$$

$$R_p \approx a\frac{\pi T_{trans}}{P(1 + \sqrt{\frac{F_0}{\Delta F}})}. \tag{26}$$

Next, we will proceed with the python programming.

4.3 Python Programming

We will work with the transit light curve data available for TrES-2 system in reference [6] (in chapter 5). 702 light years away from the Earth, this planet with a mass of roughly 1.5 Jupiter orbiting the star GSC 03549-02811, is the darkest exoplanet ever discovered that absorbs more than 99% of light that falls on its surface [5].

We download the txt file and load it in our Python program,

```
data = np.loadtxt("tres2_data.dat")

time = data[:,0]
flux = data[:,1]
err = data[:,2]
```

The variable ”time” will be the time parameter along our x-axis while plotting the flux. Its units are in days and calculated over a span of a few days. The variable ërr” has the data for the error values stored.

Next we use the ërrorbar()” function from `matplotlib.pyplot()`, to plot our data while also estimating the error in each value.

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.errorbar(time, flux, yerr=err, ecolor='steelblue',
             linestyle='none', marker='o', color='navy')

plt.xlabel("Time")
plt.ylabel("Flux")
```

We obtain the following figure 4.2,

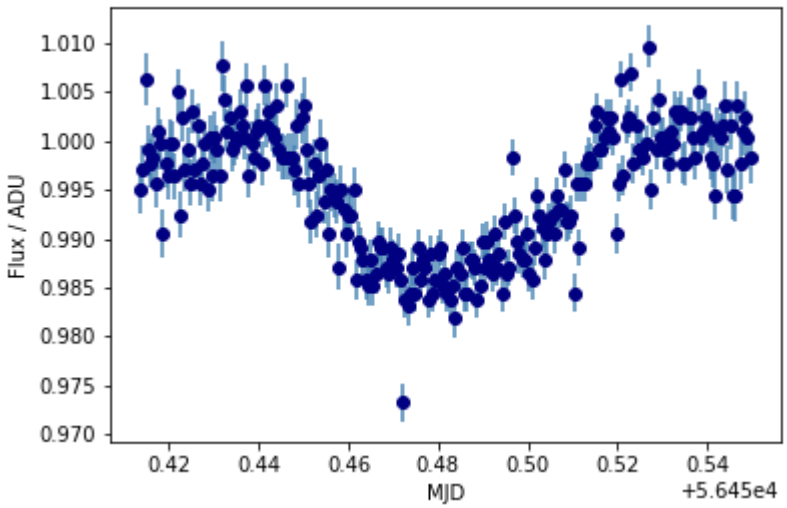

Figure 4.2: Dimming of Light from the star due to the transit of an exoplanet.

Looking at this plot, we try and ascertain the starting and ending times of the transit,

```
T_start=0.445+5.645e4
T_end=0.52+5.645e4
```

Next we proceed to smooth the plot in order to obtain the minimum flux at the peak time of the transit so that we can compute ΔF. This can be done by averaging over neighbouring data points

[7],

$$F_i^{(n)} = \frac{1}{N+1} \sum_{n=-N/2}^{N/2} F_{i+n}. \tag{27}$$

Here you sum over N neighbouring points with $N/2$ on either side. The python program works as follows:

```
N=14

# compute moving average
flux_smoothed = np.ones(flux.size - N)
for i,val in enumerate(flux_smoothed):
  flux_smoothed[i] = np.sum(flux[i:i+N+1])/(N+1)

flux_min = np.min(flux_smoothed)
print(f"Minimum flux: {flux_min:.3f}")
```

The program prints the following:
`Minimum flux: 1.487`

The maximum flux can be obtained as:

```
flux_max = np.max(flux_smoothed)
print(f"Maximum flux: {flux_max:.3f}")
```

`Maximum flux: 1.512`
This means we can use equation (22) to obtain the size of our planet.

$F_0/\Delta F$ is calculated as,
$\frac{F_0}{\Delta F} = \frac{1.512}{1.512-1.487} = 60.48$

Equation (22) gives us,
$R_p \approx a\frac{\pi T_{trans}}{P(1+\sqrt{\frac{F_0}{\Delta F}})}$.

Here P is the experimentally determined orbital period of the planet. In case of TrES-2, it comes out to be 2.47063 days and the semi-major axis as provided in equation (26) is estimated to be 0.03563 AU[5].

The Python runs as follows:

```
a=0.03563*1.496e8 #converting AU to km
P=2.47063 #days
T_trans=(T4-T1) #days
x=math.sqrt(flux_max/(flux_max-flux_min))
Radius=a*np.pi*T_trans/(P*(1+x))

print(f"Radius of the planet: {Radius:.3f} km")
```

```
Radius of the planet: 57719.674 km
```

5 Conclusion

Over the course of these four chapters, we have established an understanding of various geometric and computational concepts. We learned the modelling of orbital motion of various star systems and learned how these systems with varying masses, eccentricities and semi-major axes behave. Following on, the Python program we implemented helped us obtain the ephemeris data for these systems.

The chapter 2 focused on solving various astrophysical problems computationally. This would help a student newly stepping into the world of Python to learn computational astrophysics problem solving. Moreover, the various programs thus produced can be used by new learners to understand a myriad of astronomy concepts.

Finally, the main essence of the thesis, that is, the computational reproduction of the paper ”A geometric method to locate Neptune” is realized in chapter 3. Here, we learned how the model of geometrically predicting a new undiscovered planet works and how it can be implemented computationally. The uncertainties and lack of precision in the calculation of derivatives marred our progress, and consequently, the final results could not accurately determine the orbital parameters for Neptune, but once the correction is made, the model assures that you obtain the mass and semi-major axis of Neptune and from that, the orbital period. The prospects of this model are huge and can be applied to various extra-solar planetary systems and can lead to a prediction of undiscovered planets.

Finally, we describe another exoplanetary detection techniques that is currently very efficiently used in discovery of new exoplanets: Transit light curve method. We describe the algorithm that can help use this flux data from the eclipsing binaries and obtain the size of the exoplanet.

6 References

References

[1] Siddharth Bhatnagar, Jayanth P Vyasanakere, and Jayant Murthy. “A geometric method to locate Neptune”. In: *American Journal of Physics* 89.5 (2021), pp. 454–458.

[2] D. Giorgini and JPL Solar System Dynamics Group. *NASA/JPL Horizons On-Line Ephemeris System*. 2020. url: https://ssd.jpl.nasa.gov/horizons.cgi (visited on 04/04/2020).

[3] Muhammad Haider Khan. *Ephemeris Data.xlsx*. url: https://drive.google.com/drive/folders/1jTIP4iSCrfoqdKXt4sUcwocLxYqIx8uG?usp=sharing (visited on 05/20/2022).

[4] Nick Kollerstrom. “Neptune’s Discovery. The British Case for Co-Prediction”. In: *University College London. Archived from the original on* 11 (2005).

[5] NASA. *Exoplanet Catalog-Exoplanet Exploration*. url: https://exoplanets.nasa.gov/exoplanet-catalog/1716/tres-2-b/ (visited on 05/20/2022).

[6] Hamburg Observatory. *TrES-2 Data*. url: https://www.physik.uni-hamburg.de/en/hs/group-schmidt/python-book.html (visited on 05/20/2022).

[7] Wolfram Schmidt and Marcel Völschow. “Numerical Python in Astronomy and Astrophysics”. In: (2021).

[8] *Scipy*. url: https://scipy.org/about/ (visited on 05/20/2022).

[9] Wikipedia. *Kepler’s Laws of Planetary Motion*. url: https://en.wikipedia.org/wiki/Kepler%5C%27s_laws_of_planetary_motion (visited on 05/20/2022).

[10] Wikipedia. *Position of the Sun*. url: https://en.wikipedia.org/wiki/Position_of_the_Sun (visited on 05/20/2022).

[11] Wikipedia. *Vis-Viva Equation*. url: https://en.wikipedia.org/wiki/Vis-viva_equation#:~:text=In%5C%20astrodynamics%5C%2C%5C%20the%5C%20vis%5C%2Dviva,object%5C%20is%5C%20its%5C%20own%5C%20weight. (visited on 05/20/2022).

7 Appendix

My Google Colab notebooks for the solved python programs from the project are available as follows:

Discovering Neptune: https://colab.research.google.com/drive/1BghYEpM9IZEcbzNevBOO-6hvLeNHbztpT?usp=sharing

Orbital Mechanics: https://colab.research.google.com/drive/1Mcyhbco1Q8cyYtzHMTGEvU-Ybop6QLl7?usp=sharing

Transit Light Curves:
https://colab.research.google.com/drive/1hu49Nxs20F7kBlswFYdxvGl8-Z9yBpix8?usp=sharing

Diurnal Arc: https://colab.research.google.com/drive/1xomVv0YrdCPnwLd2a9UOyh-YtK4KwPpfC?usp=sharing

Star Latitude against the Sun:
https://colab.research.google.com/drive/13oSq6LANZSl-f1bI8pALJ2K6U- f6iC4-9?usp=sharing