МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт **«Компьютерные науки и прикладная математика» Кафедра** **802**

Группа М8О-409Б-18 **Направление подготовки** 01.03.04 «Прикладная математика»

Профиль **«Математическое моделирование динамических систем»**

Квалификация **бакалавр**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

На тему: «Построение маршрута космического корабля с использованием эффекта Оберта»

Автор ВКРБ Михеев Кирилл Вячеславович *подпись*
(фамилия, имя, отчество полностью)

Руководитель Беличенко Михаил Валериевич *подпись*
(фамилия, имя, отчество полностью)

К защите допустить

Заведующий кафедрой 802 Бардин Борис Сабирович
(№ каф) (фамилия, имя, отчество полностью)

24 мая 2022г.

Москва 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт «Компьютерные науки и прикладная математика» **Кафедра** 802

Группа М8О-409Б-18 **Направление подготовки** 01.03.04 «Прикладная математика»

Профиль «Математическое моделирование динамических систем»

Квалификация бакалавр

УТВЕРЖДАЮ

Заведующий кафедрой 802 ________________ Б.С.Бардин
(№ каф.) (подпись) (инициалы, фамилия)
09 февраля 2022 г.

ЗАДАНИЕ
на выпускную квалификационную работу бакалавра

Обучающийся Михеев Кирилл Вячеславович
(фамилия, имя, отчество полностью)

Руководитель Беличенко Михаил Валериевич
(фамилия, имя, отчество полностью
к.ф.-м.н., доцент, доцент каф. 802 МАИ
ученая степень, ученое звание, должность и место работы)

1. Наименование темы «Построение маршрута космического корабля с использованием эффекта Оберта»

2. Срок сдачи обучающимся законченной работы 24.05.2022

3. Задание и исходные данные к работе
Реализовать алгоритм построения маршрута космического корабля с использованием эффекта Оберта. Смоделировать другие маневры спутника для выхода в открытый космос для оценки эффективности использования эффекта Оберта.

Перечень иллюстративно-графических материалов*при наличии**:**

№ п/п	Наименование	Количество листов
1	Раздаточный материал	

4. Перечень подлежащих разработке разделов и этапы выполнения работы

№ п/п	Наименование раздела или этапа	Трудоёмкость в % от полной трудоёмкости ВКРБ	Срок выполнения	Примечание
1	Постановка задачи	5 %	09.02.2022-20.02.2022	
2	Изучение литературы	20%	21.02.2022-01.03.2022	
3	Разработка программы и алгоритма	20%	02.03.2022-20.03.2022	
4	Моделирование космических систем	20%	21.03.2022-20.04.2022	
5	Сравнение результатов	10%	21.04.2022 -13.05.2022	
6	Оформление текста ВКР	20%	14.05.2022-20.05.2022	
7	Подготовка презентации и раздаточного материала	5%	21.05.2022-24.05.2022	

5. Исходные материалы и пособия____

6. Дата выдачи задания 09.02.2022

Руководитель______________________________ Беличенко М.В

(подпись)

Задание принял к исполнению____________________ Михеев К.В.

(подпись)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

ОТЗЫВ
РУКОВОДИТЕЛЯ

Обучающийся Михеев Кирилл Вячеславович
(фамилия, имя, отчество полностью)

Институт «Компьютерные науки и прикладная математика» Кафедра 802

Группа М8О-409Б-18 **Направление подготовки** 01.03.04 «Прикладная математика»

Профиль «Математическое моделирование динамических систем»

Квалификация бакалавр

Наименование темы «Построение маршрута космического корабля с использованием эффекта Оберта»

Руководитель Беличенко Михаил Валериевич
(фамилия, имя, отчество полностью, ученая степень, ученое звание, должность и место работы)
к.ф.-м.н., доцент, доцент каф. 802 МАИ

Работа проверена на объем заимствования. % заимствования – 20%.

Заключение: Михеев Кирилл Вячеславович за выполнение выпускной квалификационной работы заслуживает оценку 4 (хорошо) с присуждением степени бакалавра по направлению «Прикладная математика».

24 мая 2022 г. Руководитель ______________________ М. В. Беличенко
(подпись)

Титульник (=бланк)

РЕФЕРАТ

Выпускная квалификационная работа содержит 33 страниц, 21 рисунков, 1 таблицы и 10 использованных источников.

ЭФФЕКТ ОБЕРТА. ГРАВИТАЦИОННЫЙ МАНЕВР. УРАВНЕНИЯ ДВИЖЕНИЯ. ГЕОСТАЦИОНАРНАЯ ОРБИТА. ОБЕЗРАЗМЕРИВАНИЕ. МЕТОД РУНГЕ-КУТТЫ.

Было исследовано действие эффекта Оберта на гравитационном маневре космического аппарата, проходящего вблизи Луны, для выхода на определпённое расстояние от Земли. Кроме того, для сравнения результатов были промоделированы два других этюда – использование гравитационного маневра с помощью Луны без эффекта Оберта и уход с орбиты Земли посредством использования только двигателей космического аппарата без участия гравитационного поля Луны. Были выведены уравнения движения в космической системе и проведено численное решение этих уравнений с помощью метода Рунге-Кутты. Построены траектории движения всех небесных тел в текущей системе. Спроектировано и разработано программное обеспечение для моделирования.

В ходе моделирования космических систем, экспериментальным путем были получены оптимальные параметры для каждого из объектов, чтобы визуализировать эффект Оберта и получить данные для сравнения. Посредством максимизации или минимизации некоторых признаков были подобраны такие данные, как: фазы старта движения Луны и космического аппарата, время и сила действия двигателя спутника для выхода с орбиты Земли, а также старт и конец действия двигателя при гравитационном маневре вблизи Луны для использования эффекта Оберта.

По полученным данным были построены графики финальных скоростей и затраченного топлива для каждого из этюдов, чтобы оценить эффективность использования эффекта Оберта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

До появления идеи использования гравитационного маневра с эффектом Оберта существовала очень важная проблема – невозможность исследования дальних космических тел. Проблема вытекала из того, что на тот момент не было создано тех двигателей, которые позволили бы выйти на орбиты отдаленных планет для дальнейшего исследования. Перед учеными стояла задача, как разработать более эффективные реактивные двигатели. Например, использовать ядерные или электрические ракетные двигатели.

Существует более эффективный способ достижения цели вывода космических аппаратов на орбиты дальних планет. Этим способом является гравитационный маневр около массивного движущегося небесного тела или около естественного спутника планеты. В основе этого маневра лежит идея перераспределения кинетической энергии двух тел - Луны и космического аппарата. Учитывая, что разница массы наших тел очень значительная, то получаем, что полученный разгон для космического аппарата является более эффективным способом разгона, изменения направления движения или торможения.

Эффект Оберта позволяет несколько увеличить эффективность гравитационного маневра. Сутью данного эффекта является включение двигателей спутника по направлению к небесному телу, по орбите которого происходит движение. Это позволяет получить дополнительную энергию, которая дает нам вспомогательный разгон для нашего космического аппарата.

Гравитационный маневр с использованием эффекта Оберта является более эффективным способом при нынешних двигателях, чтобы достигать точек далеко отдалённых от Земли, исследовать Солнечную систему или выходить за ее пределы.

ОСНОВНАЯ ЧАСТЬ

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Постановка задачи

Необходимо разработать программное обеспечение для построения траекторий движений и настройки параметров начальных данных небесных тел.

Рассмотрим движение космической системы, состоящей из 3-х небесных тел - Земля, Луна, спутник. В этой системе для построения и анализа эффективности использования гравитационного маневра с использованием эффекта Оберта смоделируем 3 этюда:

1. выход на определенную отдаленность от Земли с помощью реактивного двигателя без использования гравитационного поля Луны;
2. использование гравитационного маневра около Луны для получения дополнительного разгона;
3. использование гравитационного маневра с эффектом Оберта около Луны, для получения более эффективного дополнительного разгона.

Для всех этюдов нам нужно подобрать начальные данные 3-х небесных тел, тем самым смоделировав космическую систему, где они действуют на друг друга с определенной силой притяжения и имеют первоначальные координаты и скорости.

Для первого этюда нам необходимо запустить двигатели на определенном участке траектории по направлению движения космического аппарата, чтобы уйти с орбиты Земли.

Для второго этюда нужно подобрать параметр фазы начала движения Луны, чтобы “попасть” космическим аппаратом в зону действия гравитационного поля. Должны быть соблюдены все условия для того, чтобы гравитационный маневр был явно выражен и при этом не нарушал логику движения внутри космической системы. Например, чтобы координаты спутника не пересекались с координатами Луны.

Для третьего этюда с помощью экспериментальных исследований подобрать наилучшие точки включения и выключения двигателя для того, чтобы получить максимальный разгон для нашего космического аппарата. Точки подбираются благодаря перебору значений угла между векторами направления скоростей Луны и спутника.

Выведем результаты моделирования этюдов в виде графиков и таблиц, а также сделаем выводы об эффективности использования гравитационного маневра с использованием Эффекта Оберта.

1.2. Уравнения движения

Рассмотрим уравнения движения всех космических объектов, участвующих в системе. В рассматриваемом случае присутствуют несколько небесных тел, которые действуют друг на друга с некоторой силой притяжения. Необходимо вычислить сумму всех сил, действующих на каждое из тел в системе в разрезе координат. Проанализируем пример с 3-мя объектами (Земля, Луна, спутник).

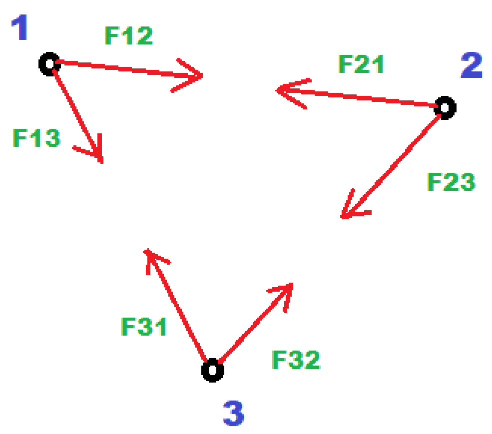


Рис. 1.1. Космическая система с направлениями действий сил приятжения

На рисунке 1.1 изображена космическая система с объектами, которые по закону всемирного тяготения Ньютона притягиваются друг к другу с некоторой силой, вызванной силой гравитации, а также зависящей от массы каждого тела и от расстояния между телами.

Сила притяжения каждого тела к друг другу определяется следующей формулой:

$$\vec{F}_{ij} = \frac{\gamma m_i m_j (\vec{r}_j - \vec{r}_i)}{\left|-\vec{r}_j + \vec{r}_i\right|^3} \tag{1.1}$$

Где γ – гравитационная постоянная, m – масса тела, r - радиус вектор.
Разложим (1.1) это в виде вектора по компонентам X и Y:

$$\vec{F}_{ij} = \left| \begin{matrix} \frac{\gamma m_i m_j (x_j - x_i)}{\left((x_j - x_i)^2 + (y_j - y_i)^2\right)^{3/2}} \\ \frac{\gamma m_i m_j (y_j - y_i)}{\left((x_j - x_i)^2 + (y_j - y_i)^2\right)^{3/2}} \\ \\ \end{matrix} \right| \tag{1.2}$$

Чтобы в дальнейшем смоделировать движение космических объектов, необходимо задать вектор состояния системы:

$$(x_1, x_2\,, x_3, y_1, y_2, y_3, \dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{y}_1, \dot{y}_2, \dot{y}_3) \tag{1.3}$$

Где x_i, y_i– координаты объектов, $\dot{x}_i$, $\dot{y}_i$- скорость объектов.
Для моделирования движения объектов нам необходимо получить новый вектор состояния, продифференцировав все предыдущие компоненты вектора состояния системы. Получим:

$$(\,\dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{y}_1, \dot{y}_2, \dot{y}_3, \ddot{x}_1, \ddot{x}_2, \ddot{x}_3, \ddot{y}_1, \ddot{y}_2, \ddot{y}_3) \tag{1.4}$$

где $\ddot{x}_i, \ddot{y}_i$ – ускорения объектов.

Получаем, что в новым векторе появились ускорения, которые находим по 2 закону Ньютона:

$$m_i \bar{w}_i = \sum_j \frac{\gamma m_i m_j * (r_j - r_i)}{|-r_j + r_i|^3} \quad (1.5)$$

Разложим полученную формулу по 2-м компонентам X и Y:

$$\frac{d^2}{dt^2} x_i(t) = \sum_{i=1}^{n} \frac{\gamma m_j (x_j - x_i)}{\left((x_j - x_i)^2 + (y_j - y_i)^2\right)^{3/2}} \quad (1.6)$$

$$\frac{d^2}{dt^2} y_i(t) = \sum_{i=1}^{n} \frac{\gamma m_j (y_j - y_i)}{\left((x_j - x_i)^2 + (y_j - y_i)^2\right)^{3/2}} \quad (1.7)$$

Получаем все необходимые компоненты векторов состояний системы для дальнейшего моделирования.

1.3. Геостационарная орбита

Для определенности будем полагать, что спутник стартует с геостационарной орбиты. Геостационарная орбита – это орбита, которая расположена над экватором Земли. Особенность ее заключается в том, что спутник вращается на ней с угловой скоростью, равной угловой скорости вращения Земли вокруг своей оси.

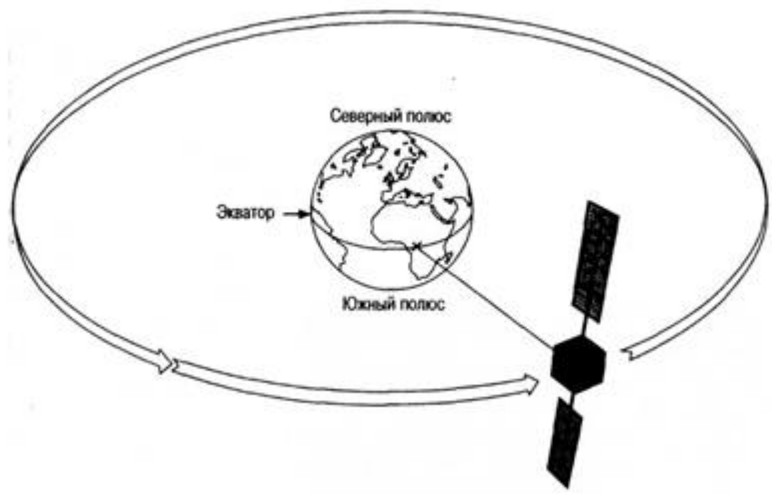


Рис. 1.2. Расположение спутника на геостационарной орбите

Вычислим радиус орбиты. Важным признаком является то, что действующие на спутник силы гравитации и центробежная сила должны уравновешивать друг друга. Чтобы вычислить высоту геостационарной орбиты воспользуемся равенством:

$$F_{и} = F_{г} \quad (1.8)$$

Где $F_{и}$ – сила инерции, а в данном случае, центробежная сила; $F_{г}$ – гравитационная сила.

По закону всемирного тяготения можем получить что:

$$F_{г} = G * \frac{M_{З} * m_{c}}{R^2} \quad (1.9)$$

Где m_c – масса спутника, $M_З$ – масса Земли, G – гравитационная постоянная, R – расстояние от спутника до Земли.

А величина центробежной силы равна:

$$F_{и} = m_c * a \quad (1.10)$$

Где $a = w^2 * R$ – центростремительное ускорение (w – угловая скорость вращения спутника).

Подставляя (1.9) и (1.10) в (1.8) получаем:

$$m_c * w^2 * R = G * \frac{M_{З} * m_c}{R^2} \quad (1.11)$$

Из (1.11) следует:

$$R = \sqrt[3]{\frac{G * M_{З}}{w}} \quad (1.12)$$

Угловая скорость равна делению угла, пройденного за один оборот на период обращения. Имеем:

$$w = \frac{2\pi}{86164} = 7.29 * 10^{-5}$$

Получаем, что $R_O -$ радиус орбиты равен 42 164 километрам.

1.4. Безразмерные параметры, переменные и уравнения

При использовании данных в моделировании космической системы кроме реальных расстояний и скоростей необходимо учитывать такое понятие, как время. Полный оборот Луны вокруг Земли будет длиться 24 часа, соответственно это не то, что необходимо. Для этого вводим обезразмерные данные относительно радиуса орбиты и расстояния не только для того, чтобы моделирование протекало быстрее, но и для получения скорости спутника относительно новых введенных величин. Для перевода метров в новые величины мы делаем следующее:

$$X = R_O * \xi \tag{1.13}$$

Где ξ – безразмерная величина.

Чтобы превратить время в обезразмерную величину нужно:

$$\tau = \Omega * T \tag{1.14}$$

Где $T = 23 * 60 * 60 + 56 * 60 + 4.091;$

$\Omega = \frac{2\pi}{T}$ - имеет размерность 1/с, поэтому чтобы скорость сделать обезразмерной, необходимо сделать следующие преобразования для обезразмеривания скорости:

$$\frac{\mathrm{dx}}{\mathrm{dt}} = \Omega\, R_O \frac{d\xi}{d\tau}$$

Тогда скорость спутника будет равна:

$$V = \Omega * R_O = 3074.655$$

Исходя из вышесказанного в конечном итоге получились обезразмерные величины для дальнейшего моделирования системы.

1.5. Стратегии достижения цели

Для оценки качества выбранного этюда необходимо определить скорость спутника на расстоянии от Земли, когда гравитационная сила изменится в 1000 раз. Также посчитать количество топлива, которое было затрачено двигателем при уходе с орбит Земли и Луны. Важным моментом, стоить отметить, что при случае, когда моделируется космическая система с помощью эффекта Оберта, то топливо затрачивается больше, чем в случае выхода с орбиты Земли посредством одного двигателя или гравитационного маневра. Для чистоты эксперимента необходимо искусственно добавлять разницу топлива для ускорения космических аппаратов в этих двух этюдах, чтобы получить равноценную по условиям финальную скорость.

Получим необходимое расстояние от Земли, где гравитационная сила уменьшится в 1000 раз и на котором будет зафиксирована финальная скорость:

$$r = \frac{\sqrt{R_O^2 * 1000}}{R_O} = 31.6 \tag{1.15}$$

Данная величина тоже была приведена к обезразмерной величине.

А) Рассмотрим случай, когда спутник будет уходить с орбиты посредством только одного двигателя до назначенной цели. Необходимо использовать запуск двигателя вдоль направления скорости спутника до тех пор, пока траектория космического аппарата не станет параболической и не закончится топливо. После выключения двигателей спутник будет иметь определённую скорость, которая будет постепенно уменьшаться приближаясь к финальной точке, где фиксируются результаты.

Б) Рассмотрим этюд с использованием гравитационного маневра. Гравитационный маневр – способ изменения скорости и направления движения спутника с помощью гравитационных полей небесных тел. При прохождении космического аппарата рядом с гравитационным полем небесного тела меняется траектория. Чем ближе к небесному телу спутник, тем сильнее изменяется его траектория.

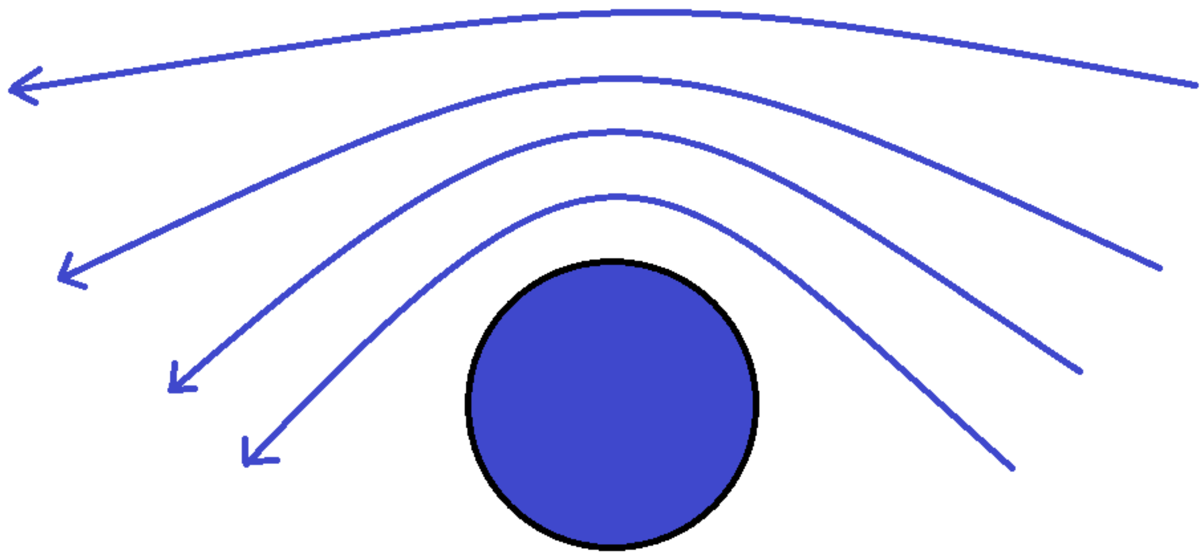


Рис. 1.3. Траектории полета спутника вблизи планеты

Космический аппарат пролетает в гравитационном колодце, который находится в движении также, как и планета, которая вращается вокруг Солнца. На вылете с орбиты планеты он получает часть орбитального импульса и ускоряется относительно Солнца.

Таким образом, мы приобретаем дополнительную скорость для космического аппарата, при всем этом не использовав двигатель, что позволило сохранить часть топлива, которое может пригодиться в дальнейшем путешествии.

В) Рассмотрим этюд с использованием гравитационного маневра с эффектом Оберта. Эффект Оберта подразумевает, что двигатель, прикрепленный к ракете и движущийся с большой скоростью, создает больше полезной энергии, чем такой же двигатель, движущийся медленно. То есть чем больше скорость у ракеты, тем больше она обладает кинетической энергией, что позволяет ее использовать для получения большей механической мощности.

Рассмотрим 3 случая с двумя объектами (планета и космический аппарат):

1) Первый случай, когда планета стоит на месте, а спутник пролетает рядом с ней определённой скоростью. Гравитация планеты воздействует на космический аппарат, что позволяет ему менять свое направление движения. Траектория получается гиперболическая.

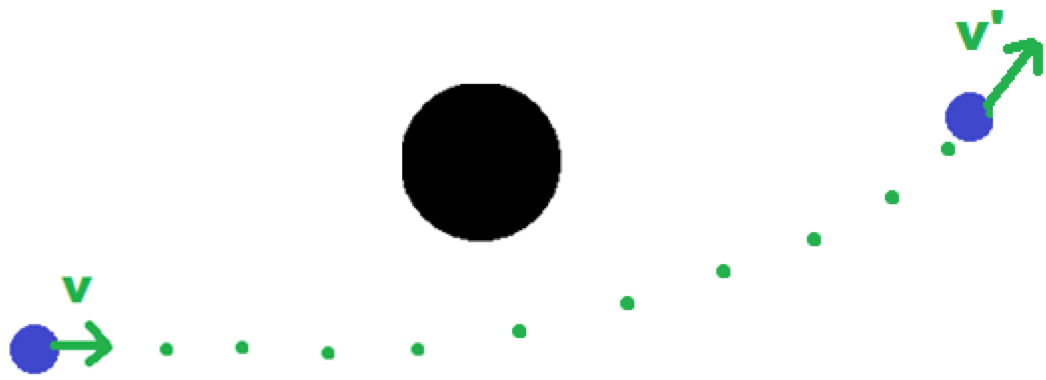


Рис 1. 4.

2) Второй случай, когда планета и космический аппарат двигаются. Благодаря действиям гравитационных полей планеты, при прохождении рядом мы получаем дополнительную скорость, которая позволяет космическому аппарату выйти на гиперболическую траекторию движения, только уже более измененной траекторий и большоей скоростью. Таким образом совершается гравитационный маневр.

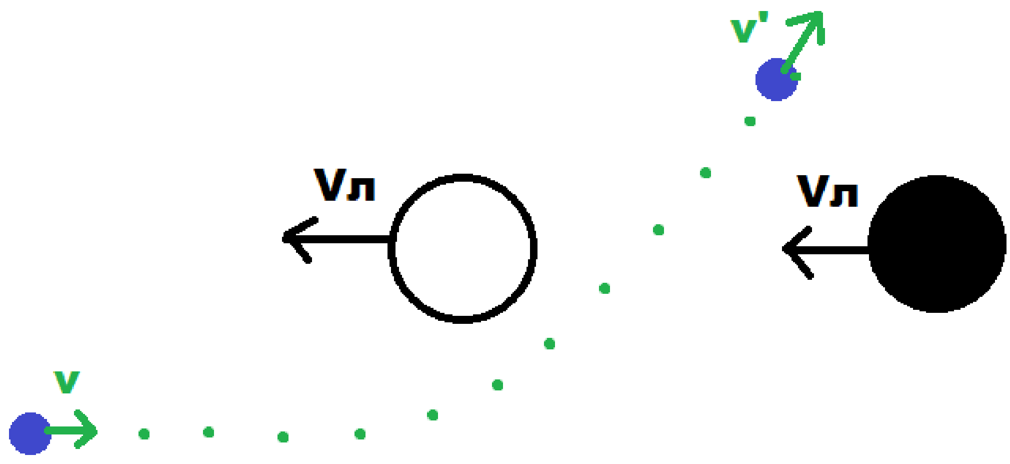


Рис 1.5.

3) Третий случай, когда в момент прохождения рядом с планетой, включаем двигатели сила которых направленна ортогонально Луне. Дойдя до определенной точки, отключаем двигатель и спутник приобретает дополнительную скорость. На этом этапе можно заметить, что запуск двигателя на высокой скорости вызывает больше изменений кинетической энергии, чем при запуске аналогичным образом на более низкой скорости. На высоких скоростях вызывается большее изменение механической энергии, чем при использовании на более низкой скорости.

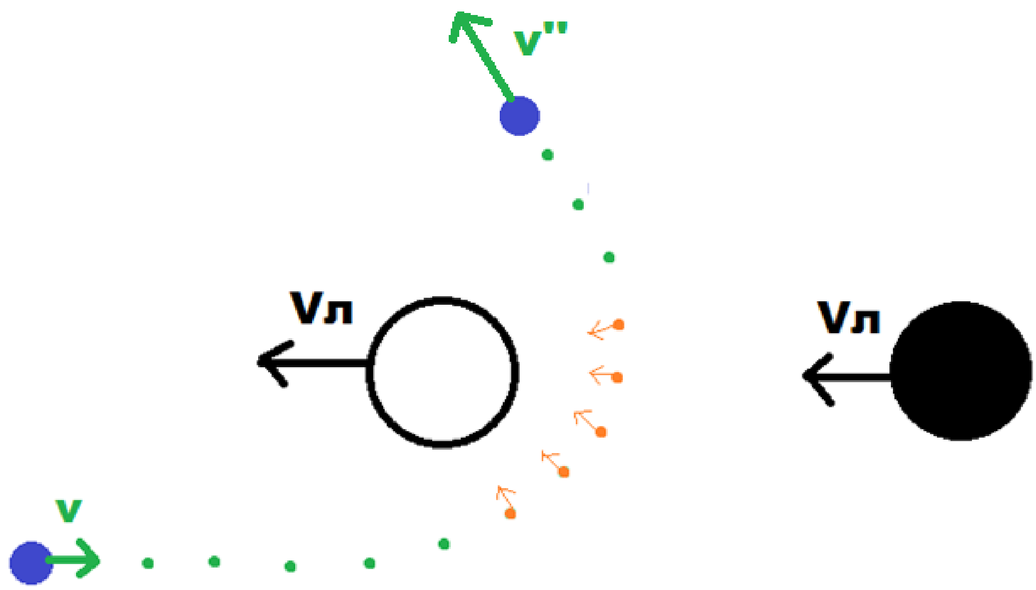


Рис 1.6.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Инструментарий

Для того чтобы смоделировать космические системы и отобразить результаты было разработано **GUI** приложение. Технологии которые были использованы:

1. Python – язык программирования
2. PyQt – набор расширений графического фреймворка Qt
3. NymPy – библиотека Python с поддержкой сложных математических структур.
4. SymPy – библиотека Python для использования символьных вычислений.
5. Matplotlib – визуализация данных двухмерной и трехмерной графики.

2.2. Создание моделей объектов системы

Было создано 3 класса объектов в программе, что ими можно было опперировать ими в самом приложении, это:

1. **PlanetSystem** – основной класс, где хранится информация обо всех объектах системы. В нем есть все методы для работы с принадлежайшему ему объектами. У него есть следующие атрибуты

    a. planets – список объектов планет в моделировании.

    b. spaceShip – объект космического аппарата.

    c. SpaceBodyMoveEquations – объект библиотеки sympy, который хранит в себе уравнения движения планет.

    d. SpaceShipMoveEquations – объект библиотеки sympy, который хранит в себе уравнения движения космического аппарата.

Методы класса:

a. add_new_planet – метод добавления новой планеты в список всех объектов.
b. add_spaceship – метод добавления космического аппарата в список всех объектов.
c. replace_system – метод изменения координат в системе у всех объектов.
d. draw – метод отрисовка всех объектов системы
e. get_move_equations – метод получения уравнения движений всех уравнения объектов.

2. **Planet** – класс планеты. Атрибуты:
   a. x, y, z – координаты в пространстве.
   b. Vx, Vy, Vz – вектора скорости.
   c. m – масса.
   d. r – радиус.

   Методы класса:
   a. replace – метод измения координат.
   b. draw – метод отрисовки в системе в начальный момент времени.
   c. re_draw – отрисовка планеты в новых координатах.

3. **SpaceShip** – класс космического аппарата. Атрибуты:
   a. x, y, z – координаты в пространстве.
   b. Vx, Vy, Vz – вектора скорости .
   c. m – масса.
   d. F_dv – сила тяги двигателя.

Методы класса:

d. replace – метод измения координат.

e. draw – метод отрисовки в начальный момент вермени в системе.

f. re_draw – отрисовка в новых координатах.

g. rot_2D – поворот ракеты.

2.3. Графический интерфейс.

Приложение для работы с моделированием системы представляет из себя окно (рис 2.1)

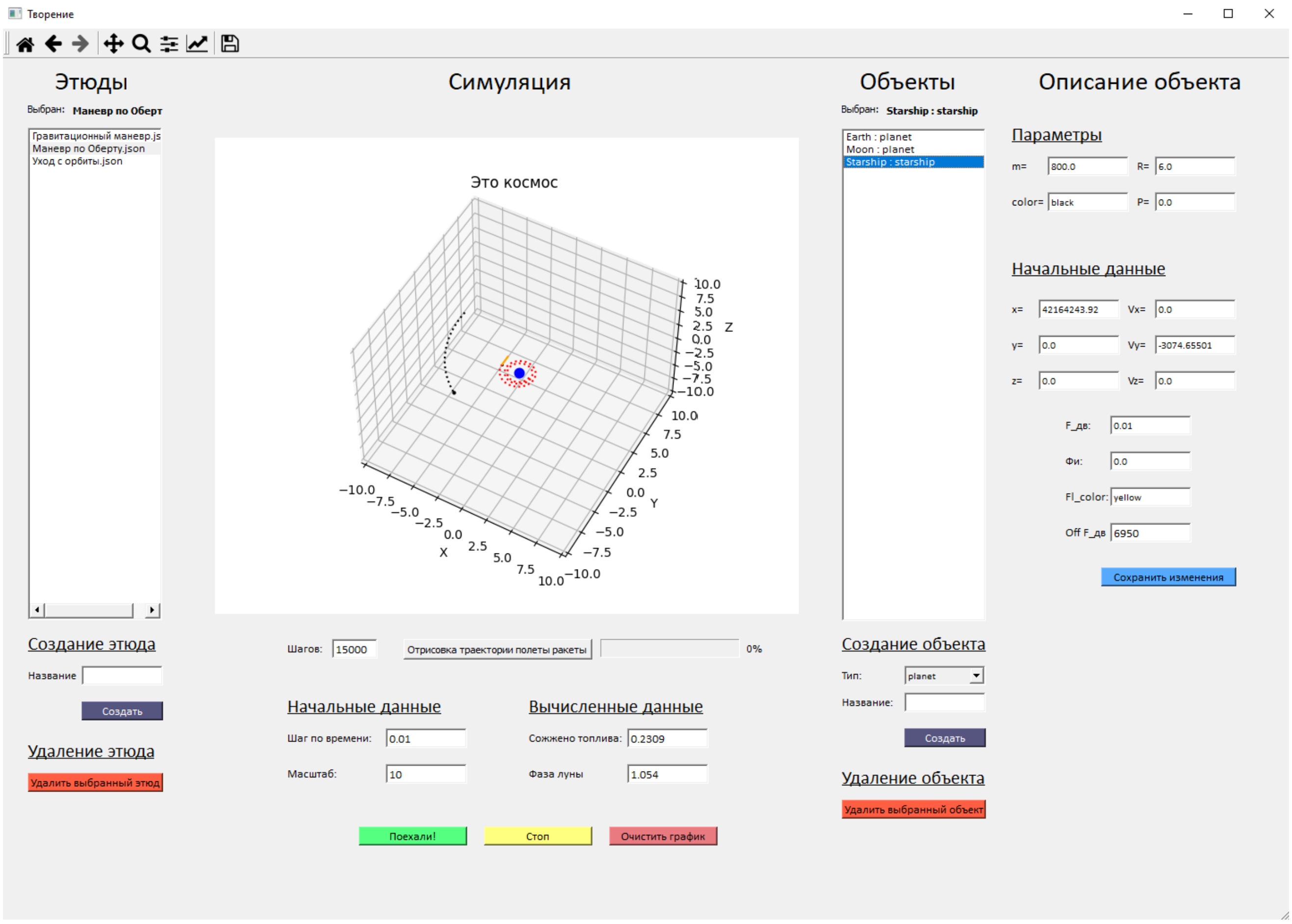


Рис. 2.1. GUI программного обеспечения

2.4. Начальные данные

Для моделирования данных этюдов нам нужно настроить начальные данные. Координаты и скорости Луны, Земли и спутника мы рассчитываем с помощью обезразмерных величин.

Получаем что для Земли:

$$\mathrm{M} = \frac{G * M_{З}}{{R_O}^{\mathbf{3}} * \Omega^{\mathbf{2}}} = 0.999999999998$$

$$\xi = \frac{X_{з}}{R_O} = 0.0$$

$$\eta = \frac{\mathrm{Y}з}{R_O} = 0.0$$

$$V_{\xi} = \frac{\mathrm{V}з}{R_O * 7.29\mathrm{e}^{-5}} = 0.0$$

$$\mathrm{V}_{\eta} = \frac{\mathrm{V}з}{R_O * 7.29\mathrm{e}^{-5}} = 0.0$$

Для Луны:

$$\mathrm{M} = \frac{G * M_{л}}{{R_O}^{\mathbf{3}} * \Omega^{\mathbf{2}}} = 0.01232376679$$

$$\xi = \frac{X_{л}}{R_O} = -7.7639$$

$$\eta = \frac{\mathrm{Y}л}{R_O} = 3.7276$$

$$V_{\xi} = \frac{\mathrm{V}л}{R_O * 7.29\mathrm{e}^{-5}} = -0.027$$

$$\mathrm{V}_{\eta} = \frac{\mathrm{V}л}{R_O * 7.29\mathrm{e}^{-5}} = -0.35$$

Для спутника:

M – незначительная по сравнению с другими небесными телами в системе

$$\xi_{\mathrm{sp}} = \frac{X_{sp}}{R_O} = -1$$

$$\eta_{\mathrm{sp}} = \frac{\mathrm{Ysp}}{R_O} = 0.0$$

$$V_{\xi sp} = \frac{\mathrm{Vsp}}{R_O * 7.29\mathrm{e}^{-5}} = 0.0$$

$$\mathrm{V}_{\eta sp} = \frac{\mathrm{Vsp}}{R_O * 7.29\mathrm{e}^{-5}} = -1.001$$

Уравнения движения считались численно с помощью метода Рунге Куты. Метод Рунге Куты представляет из себя метод 4-го порядка при вычислениях с постоянным шагом интегрирования. В нашем случае с шагом 0.1. Этот метод применяется для обыкновенных дифференциальных уравнения первого порядка.

$$\mathrm{y}' = f(x, y), \qquad y(x_0) = y_0$$

Тогда приближенное значение в последующих точках вычисляется по итерационной формуле:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Вычисление нового значения происходит в четыре стадии:

$$k_1 = f(x_n, y_n)$$

$$k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$$

$$k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$$

$$k_4 = f(x_n + h, y_n + hk_3)$$

Где h – величина шага по сетки x.

Но для получения более точных значений скорости, было принято решение уменьшить шаг интегрирования вблизи Луны. Чтобы вычисляемые значение имели наименьшую погрешность. Уменьшение происходил при приближении к Луне на расстоянии меньшей чем 1 в обезразмерных координатах. Расстояние между спутником и Луной считается по следующей формуле:

$$r_{лс} = \sqrt{(KSI_{SH} - KSI_{л})^2 + (ETA_{SH} - ETA_{л})^2}$$

Уменьшение шага интегрирования происходит в 100 раз до 0.001.

При запуске программы запускается несколько процессов:

1. Идет считывание настроек
2. Создаются все уравнения по заданным параметрам системы
3. Запускается цикл итераций по времени, который с помощью метода Рунге-Кутты дифференцирует уравнения и вычисляет новые координаты, скорости всех объектов системы.
4. Идет мгновенная перерисовка всех объектов в определенный момент времени.

2.5. Первый этюд

Смоделируем наш первый этюд где ракета уходит с орбиты только с помощью двигателя, который выключается при использовании всего досупного кол-ва топлива.:

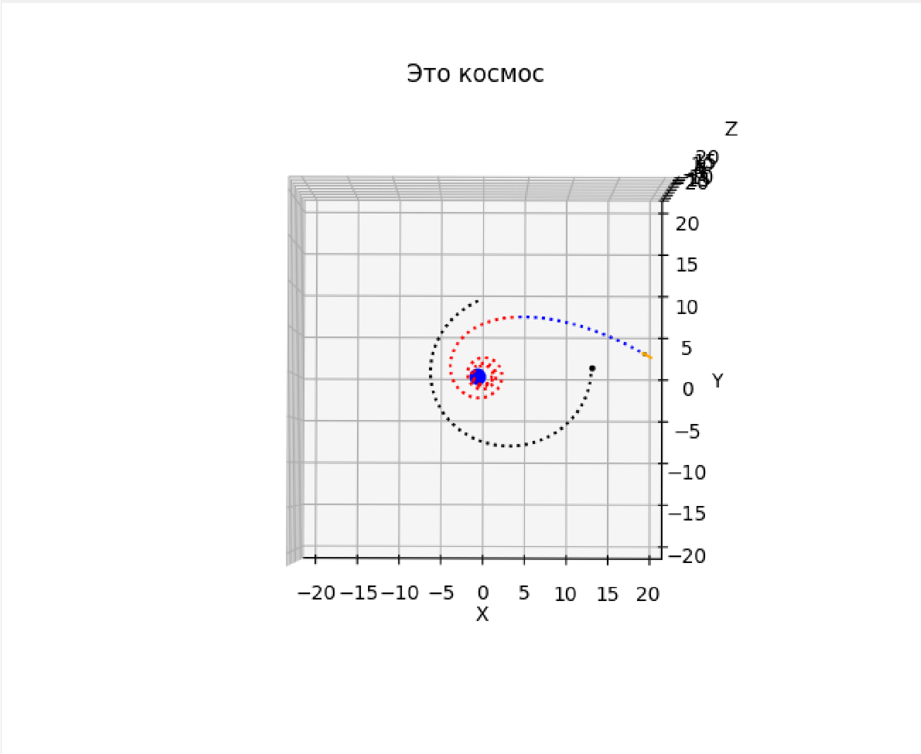


Рис. 2.2. Вид сверху на траекторию (1 этюд).

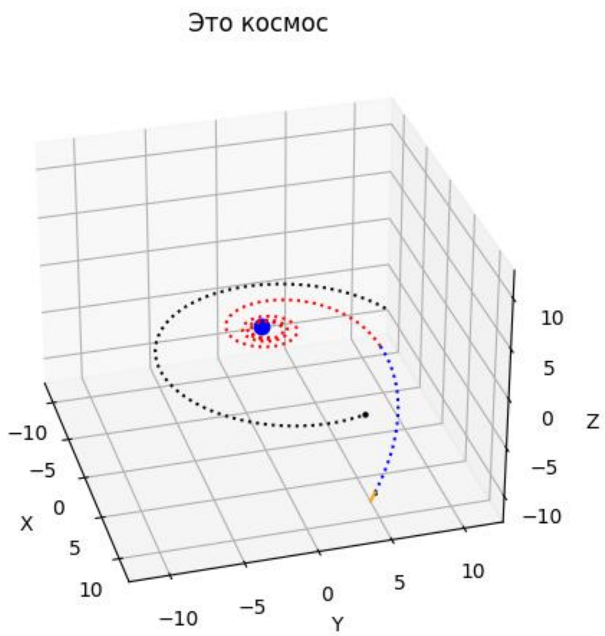


Рис. 2.3. Вид сбоку на траекторию (1 этюд).

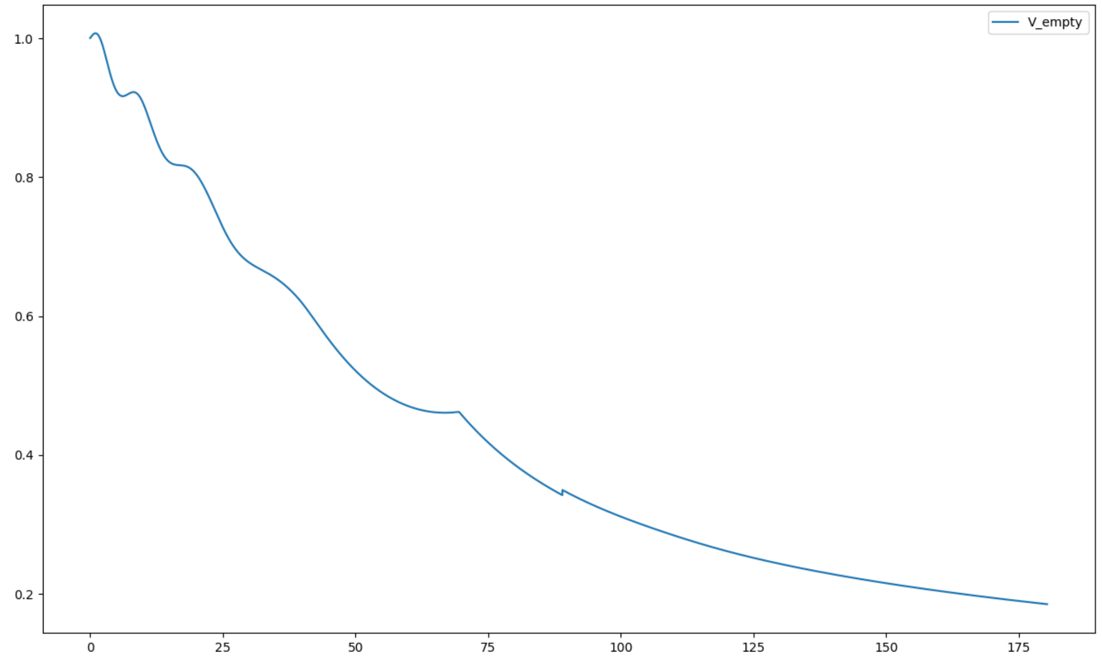


Рис. 2.4. График изменения скорости (1 этюд).

- Финальная скорость: **0.1852**
- Затрачено топлива: **1.44308 *у.е.***

В данном случае мы можем увидеть что при отдалении от Земли скорость нашего спутника постепенно уменьшается.

2.6. Второй этюд

Использование гравитационного маневра. Ищем фазу Луны и учитываем компенсацию топлива.

Важным моментом является определение фазы Луны, которая позволяет нам использовать методы гравитационного маневра с использованием эффекта Оберта или нет. Для вычисления этой фазы было проведено несколько экспериментов. При этом соблюдалось несколько условий. Первое из которых – максимизируем финальную скорость спутника при прохождение им гравитационного поля Луны. Второе условие – проверка того, что мы не подходим к Луне ближе чем 2 радиуса Луны для обеспечения безопасности полета. Вычисляется это значение следующим образом:

$$r_{лс} = \frac{R_л}{R_O} * 2$$

Выведем графики:

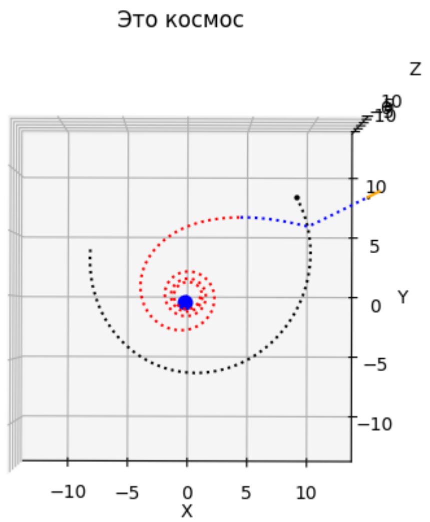


Рис. 2.5. Вид сверху на траекторию (2 этюд).

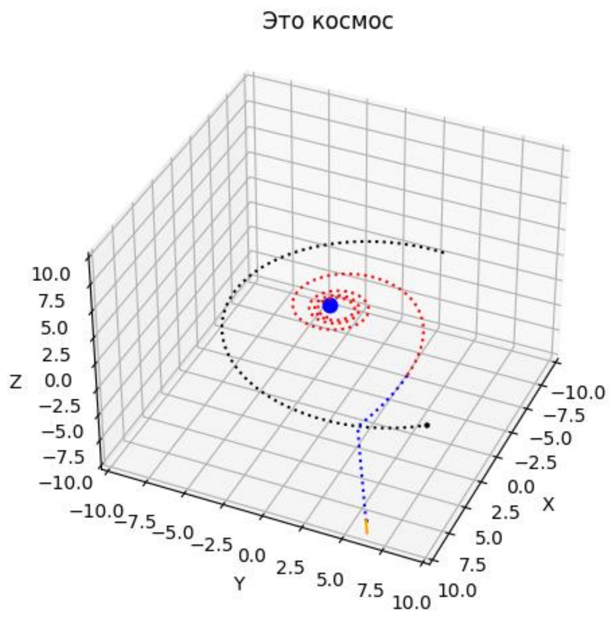


Рис. 2.6. Вид сбоку на траекторию (2 этюд).

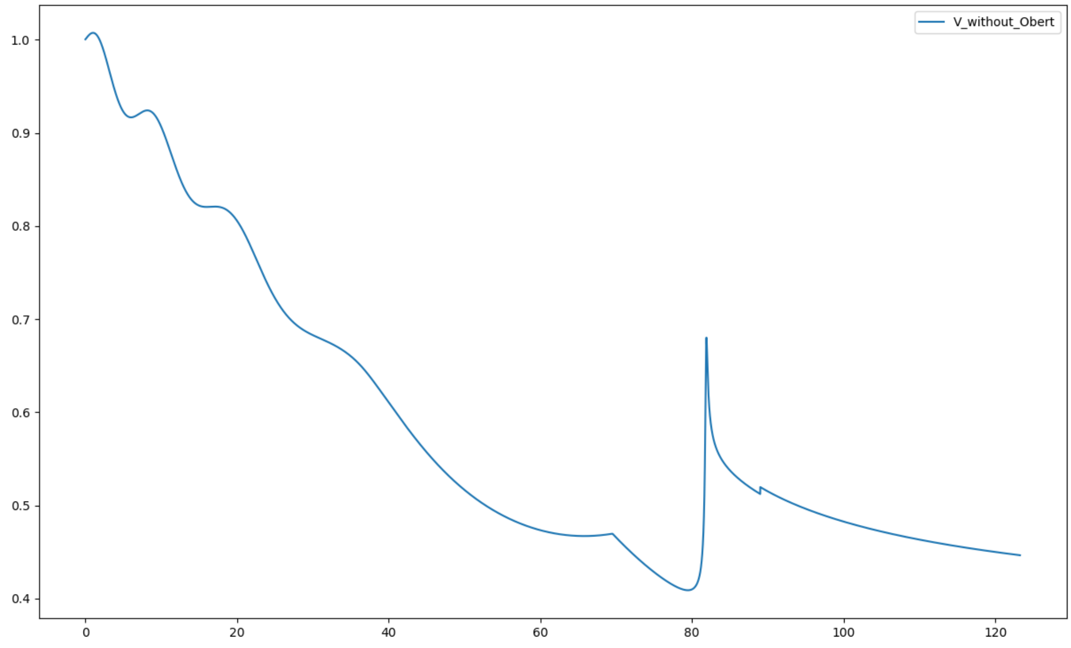


Рис. 2.7. График изменения скорости (2 этюд).

- Финальная скорость: **0.4464**
- Затрачено топлива: **1.44308 *у.е.***

На рисунке 2.7. Мы видим, что после резкого увеличения скорости засчет гравитационного маневра, происходит небольшой скачок скорости представленный на рисунке 2.8.

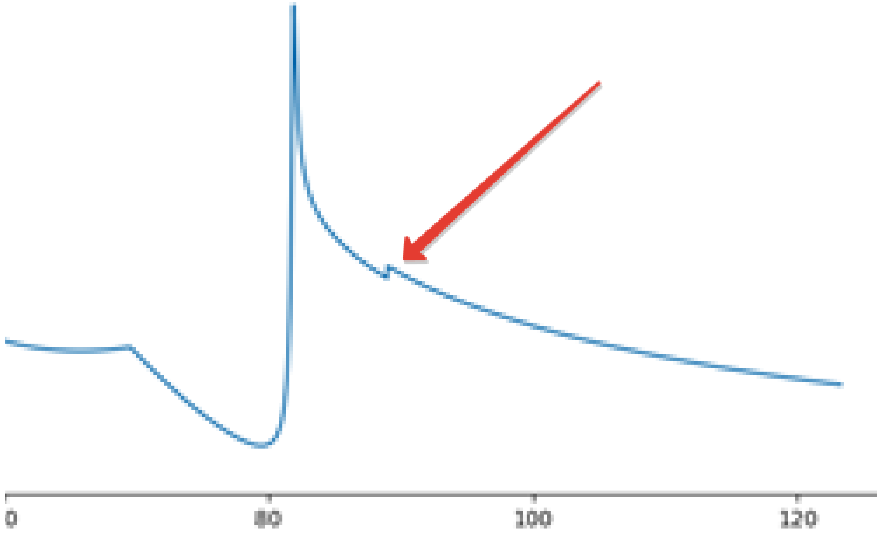


Рис. 2.8. Скачок графика скорости

Этот скачок не что иное как некая “конпенсация” за топливо, которое мы используем в эффекте Оберта. В момент когда мы используем двигатель проходя в грацитационном поле Луны затрачивается определенное кол-во топлива. Для чистоты экзсперимента и точного сравнения результатов моделирования всех 3 этюдов было решено добавить первым двум этюдам дополнительную скорость, равную разнице между использованным топливом в 3 этюде и 2 и 1 этюдах соответственно. Соответственно этот скачок является не чем иным как временным увеличением скорости.

2.7. Третий этюд

Использование гравитационного маневра с эффектом Оберта. Соновной сложность использования эффекта Оберта заключается в том, чтобы определить самые эффективные точки включения и выключения двигтаеля. Для этого программно было реализован алгортим расчета зависимости графика зависимости угла между вектором направления скорости Луны и вектором направления скорости спутника от финальной скорости.

Включение двигателя происходит, когда скорость V_{fi} в полярных координатах меняет свой знак на противоположный.

$$r = (KSI_{SH} - KSI_{л}, ETA_{SH} - ETA_{л}) \tag{2.1}$$

$$V_{fi} = \frac{VKSI_{SH} * r_y - VETA_{SH} * r_x}{\sqrt{r_x^2 + r_y^2}} \tag{2.2}$$

Угол между вектором направления скорости Луны и вектором направления скорости спутника расчитывается следующим образом:

$$V_{fi} = \frac{VKSI_{SH}*VKSI_{л}+VETA_{SH}*VETA_{л}}{\sqrt{(VKSI_{SH}^2+VETA_{SH}^2)+(VKSI_{л}^2+VETA_{л}^2)}} \quad (2.3)$$

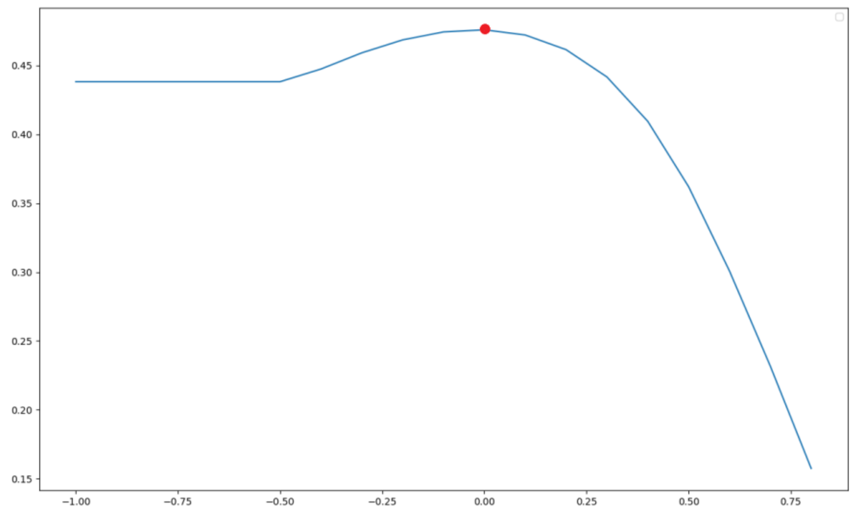


Рис. 2.9. График зависимости угла и финальной скорости

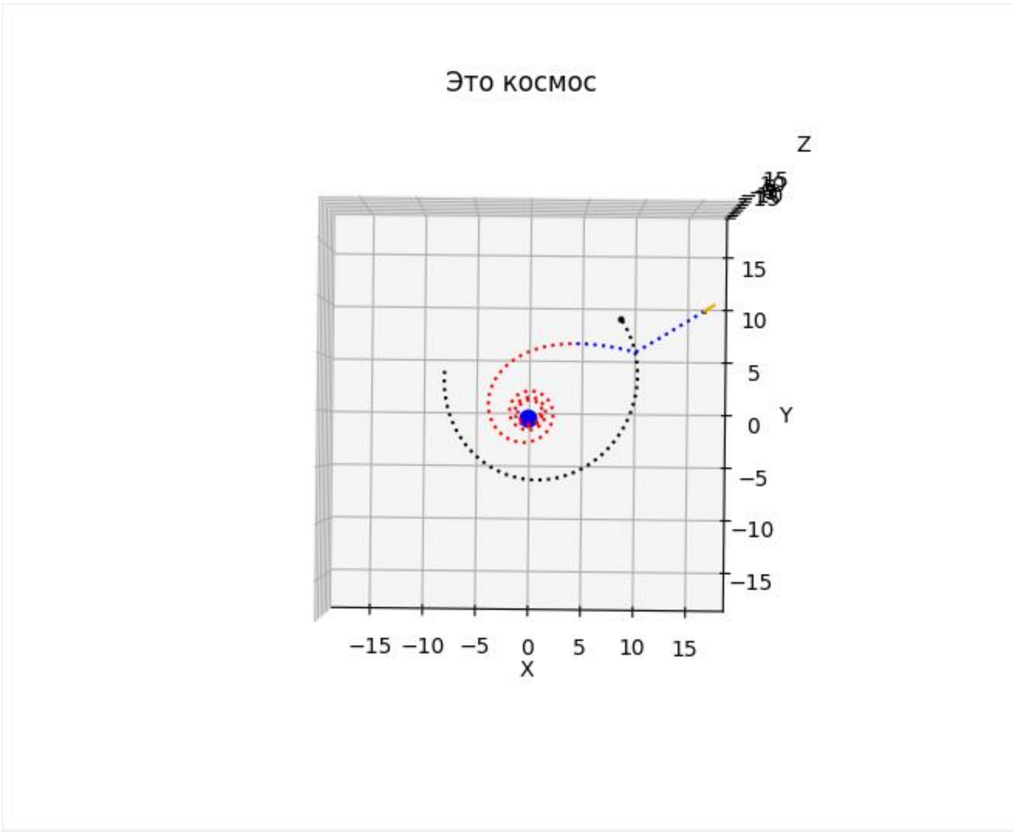


Рис. 2.10. Вид сверху на траекторию (3 этюд).

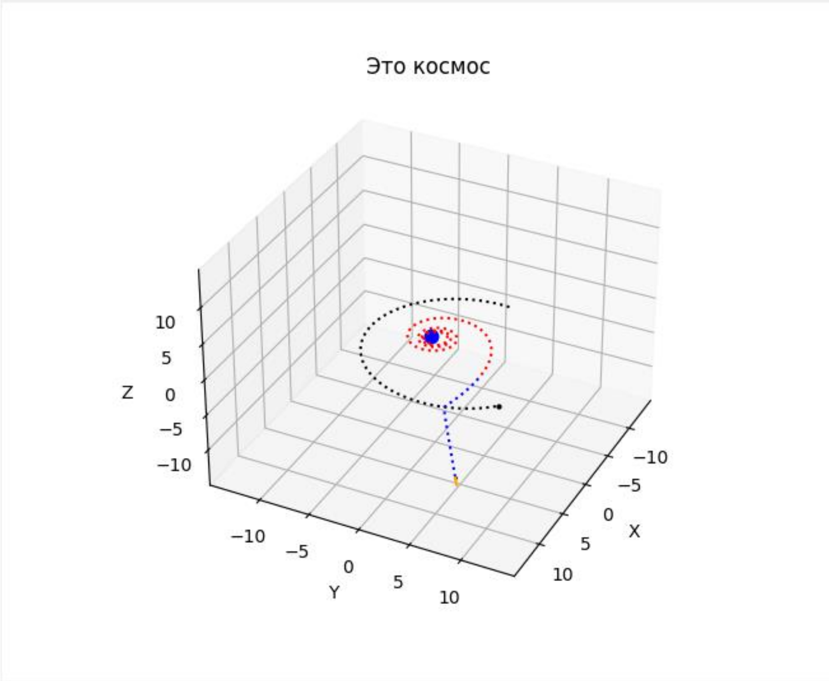


Рис. 2.11. Вид сбоку на траекторию (3 этюд).

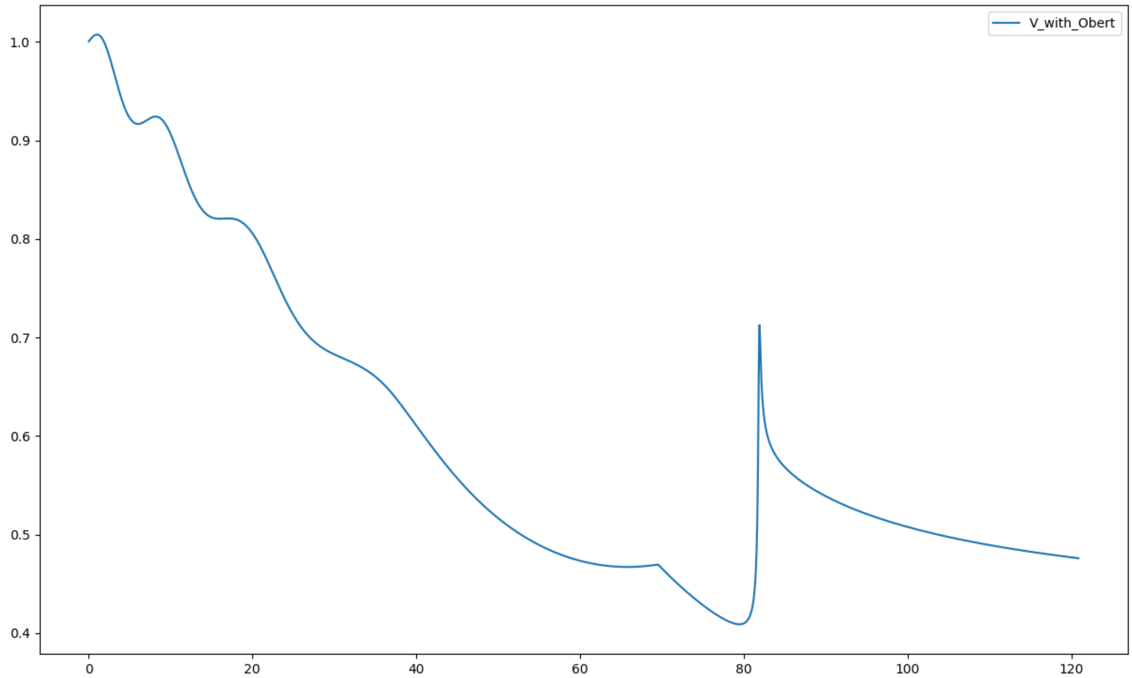


Рис. 2.12. График изменения скорости (3 этюд).

- Финальная скорость: **0.4759**
- Затрачено топлива: **1.44308** ***у.е.***

По графику (2.9) видно, что самым оптимальным решением будет выключать двигатель когда угол между векторами будет ортогональным.

2.8. Сравнение результатов

Проведем сравнения полученных результатов. Выведем на графики траектории всех трех этюдов и график изменения скоростей.

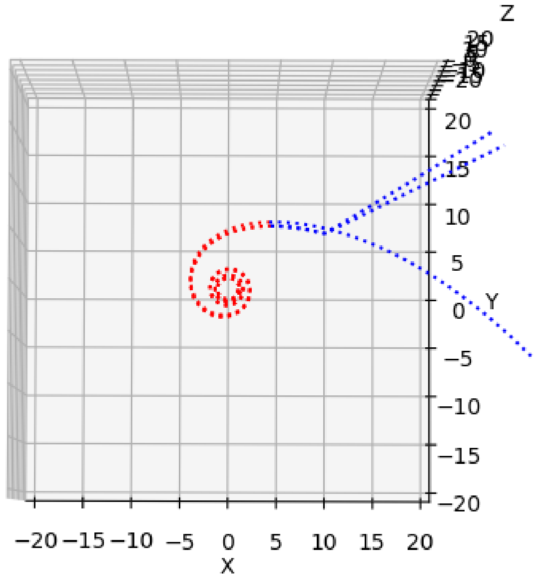


Рис. 2.13. Вид в программном обеспечение на траекторию всех этюдов.

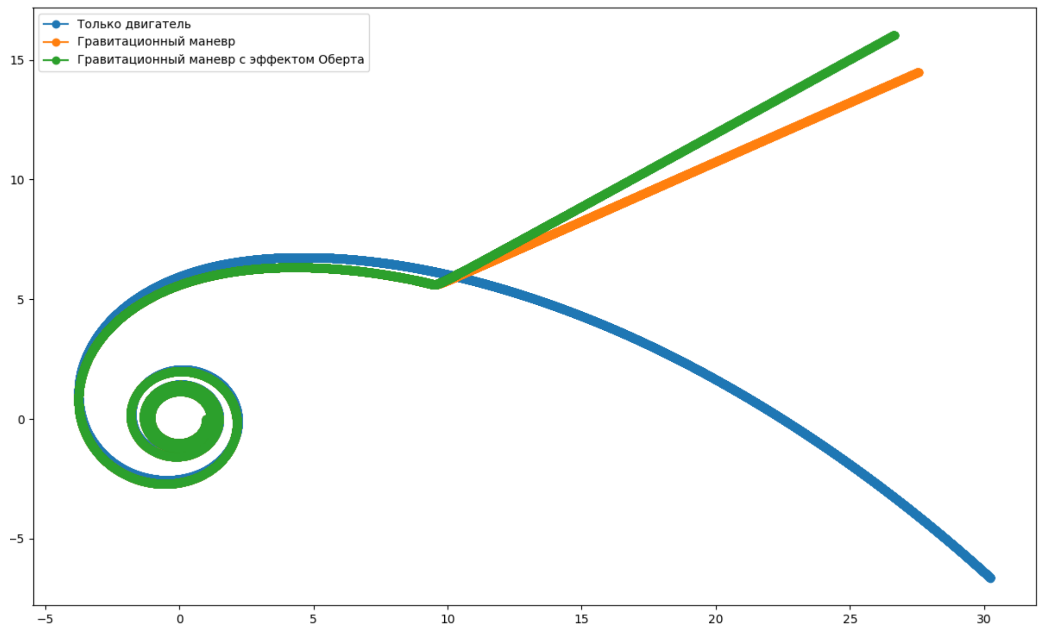


Рис. 2.14. Вид в двухмерном пространтсве на траекторию всех этюдов.

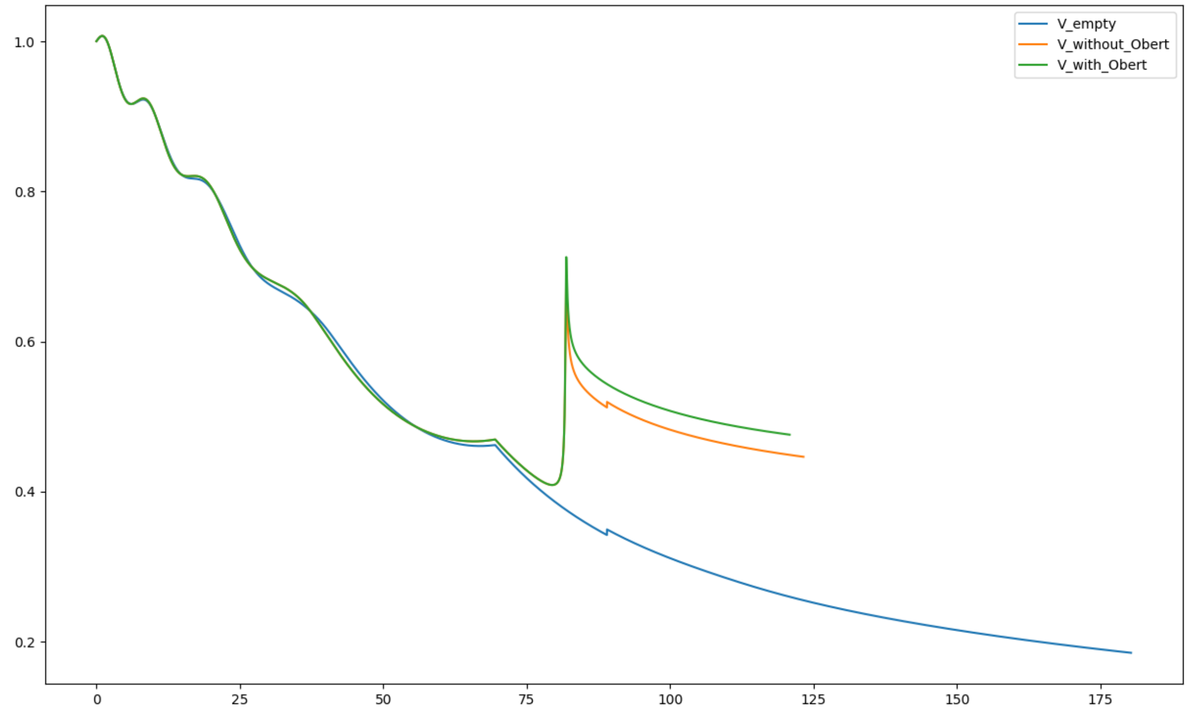


Рис. 2.15. Сравнение графиков скоростей всех этюдов.

Этюд	**Скорость**	**Затрачено топлива**
Первый	**0.1852**	**1.44308 *у.е.***
Второй	**0.4464**	**1.44308 *у.е.***
Третий	**0.4759**	**1.44308 *у.е.***

Таблица 1. Финальные скорости по этюдам

По полученным результатам можно сделать вывод о том, что с использованием гравитационного маневра с эффектом Оберта мы получаем максимальную скорость на границе. Хочется отметить, что все остальные условия были уравнены, например кол-во затраченного топлива на всем промежутке пути.

3. ЗАКЛЮЧЕНИЕ

Были смоделированы 3 независимых этюда с разной методологией выхода космического аппарат на необходимое расстояние от Земли. В ходе моделирования было обнаружено, что для этюдов с использованиям гравитационного маневра существует множество разных решений. Для явного наблюдения этих эффектов были подобраны фазы и параметры небесных тел, при которых скорость спутника будет максимальной.

Отметим, что численное исследование моделирование эффекта Оберта дало неожиданный результат по фазе выключения двигателя. Мы предполагали, что максимальная скорость будет наблюдаться, когда мы будем выключать двигатель, когда скорость спутника будет сонаправлена со скоростью луны, но оказалось, что лучше выключать в момент, когда скорость направлена против расстояния от Земли. Таким образом цель использования эффекта Оберта состоит не в том, чтобы увеличить скорость спутника за счёт скорости Луна, а в том, что развернуть вектор имеющейся у спутника скорости по направлению от Земли.

В заключении можно сказать, что использованием гравитационного маневра с помощью эффекта Оберта является самым оптимальным из всех, что были смоделированы в работе.

4. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Аксенов, Е. П. Теория движения искусственных спутников Земли / Е. П. Аксенов. — Москва : Наука, 2006. — 360 с.

2. Добронравов, В. В. Курс теоретической механики / В. В. Добронравов, Н. Н. Никитин. — Москва : Высшая школа, 1983. — 575 с.

3. Иродов, И. Е. Механика. Основные законы / И. Е. Иродов. — Москва : Резолит, 2019. — 309 с.

4. Лутц, Марк Изучаем Python / Марк Лутц. — Санкт-Петербург : Символ-плюс, 2016. — 848 с.

5. Мирер, С. А. Механика космического полета. Орбитальное движение / С. А. Мирер. — Москва : Резолит, 2007. — 106 с.

6. Мэтиз, Эрик Изучаем Python. Программирование игр, визуализация данных, веб-приложения / Мэтиз Эрик. — Санкт-Петербург : Питер, 2017. — 587 с.

7. Овчинников, М. Ю. Введение в динамику космического полёта / М. Ю. Овчинников. — Москва : МФТИ, 2016. — 208 с.

8. Прохоренок, Н. А. Python 3 и PyQt. Разработка приложений / Н. А. Прохоренок. — Санкт-Петербург : БХВ-Петербург, 2019. — 791 с.

9. Хорошилова, Е. В. Математический анализ: неопределенный интеграл / Е. В. Хорошилова. — Москва : МАКС ПРЕСС, 2019. — 180 с.

10. Чеботарев, Г. А. Аналитические и численные методы небесной механики / Г. А. Чеботарев. — Москва : Наука, 2002. — 369 с.

ПРИЛОЖЕНИЯ

Приложение А. Код моделей солнечной системы

```
class PlanetSystem():
    def __init__(self, planets):
        self.planets = planets

    def add_new_planet(self, planet):
        self.planets.append(planet)

    def add_spaceship(self, spaceShip):
        self.spaceShip = spaceShip

    def replace_system(self, KSI, ETA, ZETA, VKSI, VETA, VZETA, KSI_Sh, ETA_Sh, ZETA_Sh, VKSI_Sh, VETA_Sh, VZETA_Sh, current_step, steps_with_engine_on, Phi_Sh=0, F_curr=0):
        for planet, ksi, eta, zeta, vksi, veta, vzeta in zip(self.planets, KSI, ETA, ZETA, VKSI, VETA, VZETA):
            planet.replace(ksi, eta, zeta, vksi, veta, vzeta)
            planet.re_draw()
        if (self.spaceShip):
            self.spaceShip.replace(KSI_Sh, ETA_Sh, ZETA_Sh, VKSI_Sh, VETA_Sh, VZETA_Sh, Phi_Sh, F_curr)
            self.spaceShip.re_draw(current_step, steps_with_engine_on)

    def replace_system_without_draw(self, KSI, ETA, ZETA, VKSI, VETA, VZETA, KSI_Sh, ETA_Sh, ZETA_Sh, VKSI_Sh, VETA_Sh, VZETA_Sh, Phi_Sh=0, F_curr=0):
        for planet, ksi, eta, zeta, vksi, veta, vzeta in zip(self.planets, KSI, ETA, ZETA, VKSI, VETA, VZETA):
            planet.replace(ksi, eta, zeta, vksi, veta, vzeta)
        if (self.spaceShip):
            self.spaceShip.replace(KSI_Sh, ETA_Sh, ZETA_Sh, VKSI_Sh, VETA_Sh, VZETA_Sh, Phi_Sh, F_curr)



    def draw(self, axes):
        for planet in self.planets:
```

```
            planet.draw(axes)
        if (self.spaceShip):
            self.spaceShip.draw(axes)

    def get_move_equations(self, is_on, is_near_moon=False, is_dobavka=False, dobavka=0):
        n = len(self.planets)
        _strKSI = ''
        _strETA = ''
        _strZETA = ''
        _strVKSI = ''
        _strVETA = ''
        _strVZETA = ''
        for i in range(n):
            _strKSI += f'ksi{i}, '
            _strETA += f'eta{i}, '
            _strZETA += f'zeta{i}, '
            _strVKSI += f'Vksi{i}, '
            _strVETA += f'Veta{i}, '
            _strVZETA += f'Vzeta{i}, '

        G=6.674300000e-11

        KSI = sp.symbols(_strKSI)
        ETA = sp.symbols(_strETA)
        ZETA = sp.symbols(_strZETA)
        VKSI = sp.symbols(_strVKSI)
        VETA = sp.symbols(_strVETA)
        VZETA = sp.symbols(_strVZETA)

        DKSI= [Vksi for Vksi in VKSI]
        DETA = [Veta for Veta in VETA]
        DZETA = [Vzeta for Vzeta in VZETA]
        DVKSI = [
            sum([
```

```
            (planet.k* (ksi - cur_ksi)) / (sp.sqrt((ksi - cur_ksi) ** 2 + (eta - cur_eta) ** 2 + (zeta -
cur_zeta) ** 2) ** 3)
            for ksi, eta, zeta, planet in zip(KSI, ETA, ZETA, self.planets)
            if (ksi != cur_ksi)
        ])
        for cur_ksi, cur_eta, cur_zeta, current_planet in zip(KSI, ETA, ZETA, self.planets)
    ]

    DVETA = [
        sum([
            (planet.k* (eta - cur_eta)) / (sp.sqrt((ksi - cur_ksi) ** 2 + (eta - cur_eta) ** 2 + (zeta -
cur_zeta) ** 2) ** 3)
            for ksi, eta, zeta, planet in zip(KSI, ETA, ZETA, self.planets)
            if (ksi != cur_ksi)
        ])
        for cur_ksi, cur_eta, cur_zeta, current_planet in zip(KSI, ETA, ZETA, self.planets)
    ]
    DVZETA = [
        sum([
            (planet.k*(zeta - cur_zeta)) / (sp.sqrt((ksi - cur_ksi) ** 2 + (eta - cur_eta) ** 2 + (zeta -
cur_zeta) ** 2) ** 3)
            for ksi, eta, zeta, planet in zip(KSI, ETA, ZETA, self.planets)
            if (ksi != cur_ksi)
        ])
        for cur_ksi, cur_eta, cur_zeta, current_planet in zip(KSI, ETA, ZETA, self.planets)
    ]
    self.SpaceBodyMoveEquations = sp.lambdify([KSI, ETA, ZETA, VKSI, VETA, VZETA],
[DKSI, DETA, DZETA, DVKSI, DVETA, DVZETA])

    if (self.spaceShip):
        KSI_Sh = sp.symbols('ksi_Sh')
        ETA_Sh = sp.symbols('eta_Sh')
        ZETA_Sh = sp.symbols('zeta_Sh')
        VKSI_Sh = sp.symbols('Vksi_Sh')
        VETA_Sh = sp.symbols('Veta_Sh')
```

```
VZETA_Sh = sp.symbols('Vzeta_Sh')

F_dv = sp.symbols('f_dv')
Alpha = sp.symbols('alpha')
Beta = sp.symbols('beta')



DKSI_Sh = VKSI_Sh
DETA_Sh = VETA_Sh
DZETA_Sh = VZETA_Sh



if(is_on):
    Fx_dv_vs_Moon = 0
    Fy_dv_vs_Moon = 0
    Fx_dv_vs_Earth = F_dv * VKSI_Sh /(sp.sqrt(VKSI_Sh**2 + VETA_Sh**2))  # Сила х двигателя направленная против земли
    Fy_dv_vs_Earth = F_dv * VETA_Sh /(sp.sqrt(VKSI_Sh**2 + VETA_Sh**2))  # Сила у двигателя направленная против земли
elif(is_near_moon):
    r = [KSI_Sh - KSI[1], ETA_Sh - ETA[1]]
    Vfi = (VKSI_Sh * r[1] - VETA_Sh * r[0]) / sp.sqrt(r[0]**2 + r[1]**2)
    F = 1 * (Vfi**2/sp.sqrt(r[0]**2 + r[1]**2))

    Fx_dv_vs_Earth = 0
    Fy_dv_vs_Earth = 0
    Fx_dv_vs_Moon =  F * (-r[0])
    Fy_dv_vs_Moon =  F * (-r[1])
elif(is_dobavka):
    Fx_dv_vs_Moon = 0
    Fy_dv_vs_Moon = 0
    Fx_dv_vs_Earth = dobavka * VKSI_Sh /(sp.sqrt(VKSI_Sh**2 + VETA_Sh**2))  # Сила х двигателя направленная против земли
    Fy_dv_vs_Earth = dobavka * VETA_Sh /(sp.sqrt(VKSI_Sh**2 + VETA_Sh**2))  # Сила у двигателя направленная против земли
```

```
else:
    Fx_dv_vs_Earth = 0
    Fy_dv_vs_Earth = 0
    Fx_dv_vs_Moon = 0
    Fy_dv_vs_Moon = 0



print(f'[Fx_dv_vs_Earth] ', Fx_dv_vs_Earth)
print(f'[Fy_dv_vs_Earth] ', Fy_dv_vs_Earth)
print(f'[Fx_dv_vs_Moon] ', Fx_dv_vs_Moon)
print(f'[Fy_dv_vs_Moon] ', Fy_dv_vs_Moon)



DVKSI_Sh = sum([
    (planet.k * (ksi - KSI_Sh)) / (sp.sqrt((ksi - KSI_Sh) ** 2 + (eta - ETA_Sh) ** 2 + (zeta
- ZETA_Sh) ** 2) ** 3)
    for ksi, eta, zeta, planet in zip(KSI, ETA, ZETA, self.planets)
]) + Fx_dv_vs_Earth + Fx_dv_vs_Moon

DVETA_Sh = sum([
    (planet.k * (eta - ETA_Sh)) / (sp.sqrt((ksi - KSI_Sh) ** 2 + (eta - ETA_Sh) ** 2 +
(zeta - ZETA_Sh) ** 2) ** 3)
    for ksi, eta, zeta, planet in zip(KSI, ETA, ZETA, self.planets)
]) + Fy_dv_vs_Earth + Fy_dv_vs_Moon

DVZETA_Sh = sum([
    (planet.k * (zeta - ZETA_Sh)) /  (sp.sqrt((ksi - KSI_Sh) ** 2 + (eta - ETA_Sh) ** 2 +
(zeta - ZETA_Sh) ** 2) ** 3)
    for ksi, eta, zeta, planet in zip(KSI, ETA, ZETA, self.planets)
])

self.SpaceShipMoveEquations = sp.lambdify(
    [KSI_Sh, ETA_Sh, ZETA_Sh, VKSI_Sh, VETA_Sh, VZETA_Sh, KSI, ETA, ZETA,
VKSI, VETA, VZETA, F_dv, Alpha,Beta],
    [DKSI_Sh, DETA_Sh, DZETA_Sh, DVKSI_Sh, DVETA_Sh, DVZETA_Sh])
```

```
    def get_state_vectors(self):
        KSI = np.zeros(len(self.planets))
        ETA = np.zeros(len(self.planets))
        ZETA = np.zeros(len(self.planets))
        VKSI = np.zeros(len(self.planets))
        VETA = np.zeros(len(self.planets))
        VZETA = np.zeros(len(self.planets))
        for i in range(len(self.planets)):
            KSI[i] = self.planets[i].ksi
            ETA[i] = self.planets[i].eta
            ZETA[i] = self.planets[i].zeta
            VKSI[i] = self.planets[i].Vksi
            VETA[i] = self.planets[i].Veta
            VZETA[i] = self.planets[i].Vzeta

        return KSI, ETA, ZETA, VKSI, VETA, VZETA


class Planet():
    def __init__(self, ksi0, eta0, zeta0, Vksi0, Veta0, Vzeta0, k, m, R, color):
        self.ksi0 = ksi0
        self.eta0 = eta0
        self.zeta0 = zeta0
        self.Vksi0 = Vksi0
        self.Veta0 = Veta0
        self.Vzeta0 = Vzeta0
        self.k = k
        self.m = m
        self.R = R
        self.color = color

        self.ksi = ksi0
        self.eta = eta0
```

```
        self.zeta = zeta0
        self.Vksi = Vksi0
        self.Veta = Veta0
        self.Vzeta = Vzeta0

        phi = np.linspace(0, 6.28, 20)
        self.PlanetKSI = self.R * np.sin(phi)
        self.PlanetETA = self.R * np.cos(phi)
        self.PlanetZETA = self.R

        self.TraceKSI = np.array([self.ksi])
        self.TraceETA = np.array([self.eta])
        self.TraceZETA = np.array([self.zeta])

    def replace(self, ksi, eta, zeta, vksi, veta, vzeta):
        self.ksi = ksi
        self.eta = eta
        self.zeta = zeta
        self.Vksi = vksi
        self.Veta = veta
        self.Vzeta = vzeta

        self.TraceKSI = np.append(self.TraceKSI, ksi)
        self.TraceETA = np.append(self.TraceETA, eta)
        self.TraceZETA = np.append(self.TraceZETA, zeta)

class SpaceShip():
    def __init__(self, ksi0, eta0, zeta0, Vksi0, Veta0,Vzeta0, m, R, color, F_max, K_stop_engine):
        self.ksi0 = ksi0
        self.eta0 = eta0
        self.zeta0 = zeta0
        self.Vksi0 = Vksi0
        self.Veta0 = Veta0
        self.Vzeta0 = Vzeta0
        self.m = m
```

```
        self.R = R
        self.color = color
        self.K_stop_engine = K_stop_engine

        self.ksi = ksi0
        self.eta = eta0
        self.zeta = zeta0
        self.Vksi = Vksi0
        self.Veta = Veta0
        self.Vzeta = Vzeta0

        self.phi = 0
        self.F_dv = F_max
        self.F_curr = 0

        self.SpaceShipX = self.ksi
        self.SpaceShipY = self.eta
        self.SpaceShipZ = self.zeta

        self.TraceKSI = np.array([self.ksi])
        self.TraceETA = np.array([self.eta])
        self.TraceZETA = np.array([self.zeta])

    def replace(self, ksi, eta,zeta, vksi, veta, vzeta, phi, F_curr):
        self.ksi = ksi
        self.eta = eta
        self.zeta = zeta
        self.Vksi = vksi
        self.Veta = veta
        self.Vzeta = vzeta
        self.phi = phi
        self.F_curr = F_curr
        self.TraceKSI = np.append(self.TraceKSI, ksi)
        self.TraceETA = np.append(self.TraceETA, eta)
        self.TraceZETA = np.append(self.TraceZETA, zeta
```

```
TraceKSI_.extend(self.TraceKSI[steps_with_engine_on[0]['start']:steps_with_engine_on[0]['stop
']])

TraceETA_.extend(self.TraceETA[steps_with_engine_on[0]['start']:steps_with_engine_on[0]['st
op']])

TraceZETA_.extend(self.TraceZETA[steps_with_engine_on[0]['start']:steps_with_engine_on[0]
['stop']])

    TraceKSI_2.extend(self.TraceKSI[steps_with_engine_on[0]['stop']:])
    TraceETA_2.extend(self.TraceETA[steps_with_engine_on[0]['stop']:])
    TraceZETA_2.extend(self.TraceZETA[steps_with_engine_on[0]['stop']:])

    self.DrawedTraceEngineOn.set_data_3d(TraceKSI_, TraceETA_, TraceZETA_)
    self.DrawedTraceEngineOff.set_data_3d(TraceKSI_2, TraceETA_2, TraceZETA_2)

    Fx_dv_vs_Earth = self.F_dv * self.Vksi /(sp.sqrt(self.Vksi**2 + self.Veta**2))  # Сила x
двигателя направленная против земли
    Fy_dv_vs_Earth = self.F_dv * self.Veta /(sp.sqrt(self.Vksi**2 + self.Veta**2))  # Сила y
двигателя направленная против земли
    self.DrawedSpaceShipFlame.set_data_3d(np.array([self.ksi, self.ksi + Fx_dv_vs_Earth *
100]), np.array([self.eta, self.eta + Fy_dv_vs_Earth * 100]), self.zeta)
```

Приложение Б. Код алгоритма моделирования космических систем

```
class SpaceSystemModelling:
    def HereAreWeGo(self, is_draw_only_trajectory=False, shag=0.0):
        def NewPoints(i):
            global phaseObert, Sdobavka, flagStartEngineDobavka, flagStopEngineDobavka, R_earth_spitnik, kToplivaLeft, maxW, signVfi, flagStartEngineMoon, flagStopEngineMoon, max_cnt, t, OnOffEngine, dt, Side, plSystem, ksi, eta, zeta, Vksi, Veta, Vzeta, Dksi, Deta, Dzeta, DVksi, DVeta, DVzeta, ksi_Sh, eta_Sh, zeta_Sh, Vksi_Sh, Veta_Sh, Vzeta_Sh, Dksi_Sh, Deta_Sh, Dzeta_Sh, DVksi_Sh, DVeta_Sh, DVzeta_Sh, F_dv, Alpha, Beta, K_stop_engine
            t += dt

            #Методом Рунге - Кутты
            Dksi1, Deta1,Dzeta1, DVksi1, DVeta1,DVzeta1 = plSystem.SpaceBodyMoveEquations(ksi, eta, zeta, Vksi, Veta,Vzeta)
            Dksi1_Sh, Deta1_Sh,Dzeta1_Sh, DVksi1_Sh, DVeta1_Sh,DVzeta1_Sh = plSystem.SpaceShipMoveEquations(ksi_Sh, eta_Sh,zeta_Sh, Vksi_Sh, Veta_Sh, Vzeta_Sh, ksi, eta, zeta,Vksi, Veta, Vzeta, F_dv, Alpha,Beta)

            Dksi1 = np.array(Dksi1)
            Deta1 = np.array(Deta1)
            Dzeta1 = np.array(Dzeta1)
            DVksi1 = np.array(DVksi1)
            DVeta1 = np.array(DVeta1)
            DVzeta1 = np.array(DVzeta1)

            Dksi1_Sh = np.array(Dksi1_Sh)
            Deta1_Sh = np.array(Deta1_Sh)
            Dzeta1_Sh = np.array(Dzeta1_Sh)
            DVksi1_Sh = np.array(DVksi1_Sh)
            DVeta1_Sh = np.array(DVeta1_Sh)
            DVzeta1_Sh = np.array(DVzeta1_Sh)
```

```
Dksi2, Deta2, Dzeta2, DVksi2, DVeta2, DVzeta2 = plSystem.SpaceBodyMoveEquations(ksi+Dksi1/2*dt, eta+Deta1/2*dt, zeta+Dzeta1/2*dt, Vksi+DVksi1/2*dt, Veta+DVeta1/2*dt,Vzeta+DVzeta1/2*dt)
Dksi2_Sh, Deta2_Sh,Dzeta2_Sh, DVksi2_Sh, DVeta2_Sh,DVzeta2_Sh = plSystem.SpaceShipMoveEquations(
    ksi_Sh+Dksi1_Sh/2*dt, eta_Sh+Deta1_Sh/2*dt, zeta_Sh+Dzeta1_Sh/2*dt, Vksi_Sh+DVksi1_Sh/2*dt, Veta_Sh+DVeta1_Sh/2*dt,Vzeta_Sh+DVzeta1_Sh/2*dt,
    ksi+Dksi1/2*dt, eta+Deta1/2*dt,zeta+Dzeta1/2*dt, Vksi+DVksi1/2*dt, Veta+DVeta1/2*dt, Vzeta+DVzeta1/2*dt, F_dv, Alpha,Beta)

Dksi2 = np.array(Dksi2)
Deta2 = np.array(Deta2)
Dzeta2 = np.array(Dzeta2)
DVksi2 = np.array(DVksi2)
DVeta2 = np.array(DVeta2)
DVzeta2 = np.array(DVzeta2)
Dksi2_Sh = np.array(Dksi2_Sh)
Deta2_Sh = np.array(Deta2_Sh)
Dzeta2_Sh = np.array(Dzeta2_Sh)
DVksi2_Sh = np.array(DVksi2_Sh)
DVeta2_Sh = np.array(DVeta2_Sh)
DVzeta2_Sh = np.array(DVzeta2_Sh)

Dksi3, Deta3, Dzeta3, DVksi3, DVeta3, DVzeta3 = plSystem.SpaceBodyMoveEquations(ksi+Dksi2/2*dt, eta+Deta2/2*dt, zeta+Dzeta2/2*dt, Vksi+DVksi2/2*dt, Veta+DVeta2/2*dt,Vzeta+DVzeta2/2*dt)
Dksi3_Sh, Deta3_Sh, Dzeta3_Sh, DVksi3_Sh, DVeta3_Sh, DVzeta3_Sh = plSystem.SpaceShipMoveEquations(
    ksi_Sh + Dksi2_Sh / 2 * dt, eta_Sh + Deta2_Sh / 2 * dt, zeta_Sh + Dzeta2_Sh / 2 * dt,
    Vksi_Sh + DVksi2_Sh / 2 * dt, Veta_Sh + DVeta2_Sh / 2 * dt, Vzeta_Sh + DVzeta2_Sh / 2 * dt,
    ksi + Dksi2 / 2 * dt, eta + Deta2 / 2 * dt, zeta + Dzeta2 / 2 * dt, Vksi + DVksi2 / 2 * dt,
```

```
            Veta + DVeta2 / 2 * dt, Vzeta + DVzeta2 / 2 * dt, F_dv, Alpha, Beta)

        Dksi3 = np.array(Dksi3)
        Deta3 = np.array(Deta3)
        Dzeta3 = np.array(Dzeta3)
        DVksi3 = np.array(DVksi3)
        DVeta3 = np.array(DVeta3)
        DVzeta3 = np.array(DVzeta3)
        Dksi3_Sh = np.array(Dksi3_Sh)
        Deta3_Sh = np.array(Deta3_Sh)
        Dzeta3_Sh = np.array(Dzeta3_Sh)
        DVksi3_Sh = np.array(DVksi3_Sh)
        DVeta3_Sh = np.array(DVeta3_Sh)
        DVzeta3_Sh = np.array(DVzeta3_Sh)

        Dksi4, Deta4, Dzeta4, DVksi4, DVeta4, DVzeta4  =
plSystem.SpaceBodyMoveEquations(ksi+Dksi3/2*dt, eta+Deta3/2*dt, zeta+Dzeta3/2*dt,
Vksi+DVksi3/2*dt, Veta+DVeta3/2*dt,Vzeta+DVzeta3/2*dt)
        Dksi4_Sh, Deta4_Sh, Dzeta4_Sh, DVksi4_Sh, DVeta4_Sh, DVzeta4_Sh =
plSystem.SpaceShipMoveEquations(
            ksi_Sh + Dksi3_Sh / 2 * dt, eta_Sh + Deta3_Sh / 2 * dt, zeta_Sh +
Dzeta3_Sh / 2 * dt,
            Vksi_Sh + DVksi3_Sh / 2 * dt, Veta_Sh + DVeta3_Sh / 2 * dt,
Vzeta_Sh + DVzeta3_Sh / 2 * dt,
            ksi + Dksi3 / 2 * dt, eta + Deta3 / 2 * dt, zeta + Dzeta3 / 2 * dt, Vksi +
DVksi3 / 2 * dt,
            Veta + DVeta3 / 2 * dt, Vzeta + DVzeta3 / 2 * dt, F_dv, Alpha, Beta)

        Dksi4 = np.array(Dksi4)
        Deta4 = np.array(Deta4)
        Dzeta4 = np.array(Dzeta4)
        DVksi4 = np.array(DVksi4)
        DVeta4 = np.array(DVeta4)
        DVzeta4 = np.array(DVzeta4)
        Dksi4_Sh = np.array(Dksi4_Sh)
```

```
Deta4_Sh = np.array(Deta4_Sh)
Dzeta4_Sh = np.array(Dzeta4_Sh)
DVksi4_Sh = np.array(DVksi4_Sh)
DVeta4_Sh = np.array(DVeta4_Sh)
DVzeta4_Sh = np.array(DVzeta4_Sh)

ksi = ksi + dt/6 * (Dksi1 + 2*Dksi2 + 2*Dksi3 + Dksi4)
eta = eta + dt/6 * (Deta1 + 2*Deta2 + 2*Deta3 + Deta4)
zeta = zeta + dt / 6 * (Dzeta1 + 2 * Dzeta2 + 2 * Dzeta3 + Dzeta4)

Vksi = Vksi + dt/6 * (DVksi1 + 2*DVksi2 + 2*DVksi3 + DVksi4)
Veta = Veta + dt/6 * (DVeta1 + 2*DVeta2 + 2*DVeta3 + DVeta4)
Vzeta = Vzeta + dt / 6 * (DVzeta1 + 2 * DVzeta2 + 2 * DVzeta3 +
DVzeta4)

ksi_Sh = ksi_Sh + dt / 6 * (Dksi1_Sh + 2 * Dksi2_Sh + 2 * Dksi3_Sh +
Dksi4_Sh)
eta_Sh = eta_Sh + dt / 6 * (Deta1_Sh + 2 * Deta2_Sh + 2 * Deta3_Sh +
Deta4_Sh)
zeta_Sh = zeta_Sh + dt / 6 * (Dzeta1_Sh + 2 * Dzeta2_Sh + 2 *
Dzeta3_Sh + Dzeta4_Sh)

Vksi_Sh = Vksi_Sh + dt / 6 * (DVksi1_Sh + 2 * DVksi2_Sh + 2 *
DVksi3_Sh + DVksi4_Sh)
Veta_Sh = Veta_Sh + dt / 6 * (DVeta1_Sh + 2 * DVeta2_Sh + 2 *
DVeta3_Sh + DVeta4_Sh)
Vzeta_Sh = Vzeta_Sh + dt / 6 * (DVzeta1_Sh + 2 * DVzeta2_Sh + 2 *
DVzeta3_Sh + DVzeta4_Sh)

# Увеличиваем шаг интегрирования при приближении к Луне
if(len(ksi) > 1):
    rast = np.sqrt((ksi_Sh - ksi[1])**2 + (eta_Sh - eta[1])**2 +(zeta_Sh
- zeta[1])**2)
    if(rast < 1):
        dt = 0.001
```

```
                else:
                    dt = 0.01

            # Вывод шага
            if(i % 500 == 0):
                print('[step, t, R_earth_spitnik] ', i, t, R_earth_spitnik)



            # Запись данных в файл для анализа
            try:
                if(name_etude == 'Маневр по Оберту.json'):
                    r = [ksi_Sh - ksi[1], eta_Sh - eta[1]]
                    Vr = (Vksi_Sh * r[0] + Veta_Sh * r[1]) / np.sqrt(r[0]**2 +
r[1]**2)
                    Vfi = (Vksi_Sh * r[1] - Veta_Sh * r[0]) / np.sqrt(r[0]**2 +
r[1]**2)
                    w = (Vfi**2)/np.sqrt(r[0]**2 + r[1]**2)
                    cosA = (Vksi_Sh * Vksi[1] + Veta_Sh *
Veta[1])/(np.sqrt(Vksi_Sh**2 + Veta_Sh **2)*np.sqrt(Vksi[1]**2 + Veta[1] **2))

                    self.moveDataCoordinates['cosA'].append(cosA)
                    self.moveDataCoordinates['Vr'].append(Vr)
                    self.moveDataCoordinates['Vfi'].append(Vfi)
                    self.moveDataCoordinates['w'].append(w)

                    self.moveDataCoordinates['r'].append(np.sqrt((ksi_Sh -
ksi[1])**2 + (eta_Sh - eta[1])**2))

                    if(flagStartEngineMoon and not flagStopEngineMoon):
                        phaseObert+=1
                    self.moveDataCoordinates['vklObert'].append(phaseObert)

                    V_sh__Vmoon = np.sqrt(Vksi_Sh**2 + Vksi[1]**2) *
np.sqrt(Veta_Sh**2 + Veta[1]**2) * cosA
```

```
                                        self.moveDataCoordinates['V_sh *
Vmoon'].append(V_sh__Vmoon)
                                        V_fin__Rearth = math.atan2(ksi_Sh - ksi[0], eta_Sh -
eta[0]) - math.atan2(Vksi_Sh, Veta_Sh)
                                        self.moveDataCoordinates['Угол между финальной
скоростью и расстоянием на Землю'].append(V_fin__Rearth)

                                R_earth_spitnik = np.sqrt((ksi_Sh - ksi[0])**2 + (eta_Sh -
eta[0])**2)
                        self.moveDataCoordinates['R_earth_spitnik'].append(R_earth_spitnik)



                                self.moveDataCoordinates['t'].append(t)
                                self.moveDataCoordinates['x'].append(ksi_Sh)
                                self.moveDataCoordinates['y'].append(eta_Sh)
                                self.moveDataCoordinates['V'].append(np.sqrt(Vksi_Sh**2 +
Veta_Sh**2))

                                if(R_earth_spitnik > 31.6 and not self.is_load): #
                                        self.load_info(name_etude)
                                        print('[load] Success')
                                        print('[V_fin__Rearth, cosA]', V_fin__Rearth, cosA)

                        except BaseException as e:
                                print(f'[load] Error: Ошибка в запоминании данных - {e}')

                        # Работа двигателя, если использован метод Оберта
                        if(name_etude == 'Маневр по Оберту.json'):
                                # Включаем двигатель против Луны, когда скорость меняет
свой знак
                                if(signVfi != (-1 if Vfi < 0 else 1) and t > 70 and not
flagStartEngineMoon):
                                        print('[oh yes..] ', cosA)
                                        plSystem.get_move_equations(False, True)
```

```
                flagStartEngineMoon = True

            # Выключем двигатель против Луны, когда центробежная сила
достигает своего максимума
            #if(maxW > w and t > 81.6 and not flagStopEngineMoon):
            if(round(cosA, 3) > shag and t > 81.6 and not
flagStopEngineMoon):
                print('[oh no..]')
                plSystem.get_move_equations(False, False)
                flagStopEngineMoon = True
                # print('[V_sh__Vmoon t]', V_sh__Vmoon, t)
            # elif(maxW < w and t > 81.6 and not flagStopEngineMoon):
            #       maxW = w
        else:
            # Добавочная скорость которую можно использовать по
Оберту
            if (t > 89 and not flagStartEngineDobavka):
                plSystem.get_move_equations(False, False, True,
Sdobavka)
                kToplivaLeft += Sdobavka
                flagStartEngineDobavka = True
            elif(kToplivaLeft >= Sdobavka and t > 89 and not
flagStopEngineDobavka):
                plSystem.get_move_equations(False, False, False, 0)
                flagStopEngineDobavka = True

        # Включение и выключение двигателя по шагу
        for kk in range(len(OnOffEngine)):
            if(i > OnOffEngine[kk]['start'] and not
OnOffEngine[kk]['is_started']):
                plSystem.get_move_equations(True, False)
                OnOffEngine[kk]['is_started'] = True
            elif (i > OnOffEngine[kk]['stop'] and not
OnOffEngine[kk]['is_stoped']):
                plSystem.get_move_equations(False, False)
```

```
                    OnOffEngine[kk]['is_stoped'] = True

            # Отображение потраченного топлива
            if(i <= int(K_stop_engine)):
                kToplivaLeft += dt * F_dv
            elif(flagStartEngineMoon and not flagStopEngineMoon): #or
(flagStartEngineDobavka and not flagStopEngineDobavka))
                kToplivaLeft += dt * w
            self.K_toplivo_out.setText(str(round(kToplivaLeft, 5)))

            # Изменение расположения объектов и ререндер
            if(is_draw_only_trajectory):
              plSystem.replace_system_without_draw(ksi, eta, zeta, Vksi, Veta,Vzeta,
ksi_Sh, eta_Sh, zeta_Sh, Vksi_Sh, Veta_Sh, Vzeta_Sh)
            else:
                plSystem.replace_system(ksi, eta, zeta, Vksi, Veta, Vzeta, ksi_Sh,
eta_Sh, zeta_Sh, Vksi_Sh, Veta_Sh, Vzeta_Sh, i, OnOffEngine)
                drPlanets = [planet.DrawedPlanet for planet in plSystem.planets]
                drTraces = [planet.DrawedTrace for planet in plSystem.planets]
                return [plSystem.spaceShip.DrawedSpaceShip] + drTraces +
drPlanets + [plSystem.spaceShip.DrawedTraceEngineOn] +
[plSystem.spaceShip.DrawedTraceEngineOff] + [plSystem.spaceShip.DrawedSpaceShipFlame]
                # + [plSystem.spaceShip.DrawedTraceAfterMoon] +
                # + [plSystem.spaceShip.DrawedTrace] + \



        global phaseObert, Sdobavka, flagStartEngineDobavka, flagStopEngineDobavka,
R_earth_spitnik, name_etude, kToplivaLeft, maxW, signVfi, flagStartEngineMoon,
flagStopEngineMoon, max_cnt, t, OnOffEngine, Side, dt, plSystem, ksi, eta, zeta, Vksi, Veta,
Vzeta, Dksi, Deta, Dzeta, DVksi, DVeta, DVzeta, ksi_Sh, eta_Sh, zeta_Sh, Vksi_Sh, Veta_Sh,
Vzeta_Sh, Dksi_Sh, Deta_Sh, Dzeta_Sh, DVksi_Sh, DVeta_Sh, DVzeta_Sh, F_dv, Alpha, Beta,
K_stop_engine

        phaseObert = 0
        flagStartEngineDobavka = False
```

```
flagStopEngineDobavka = False
Sdobavka = 0.74798
R_earth_spitnik = 0
kToplivaLeft = 0
signVfi = 1
flagStartEngineMoon = False
flagStopEngineMoon = False
maxW = 0
t = 0
F_dv = 0
Alpha = 0
Beta = 0
# Параметры системы
dt = float(self.TStep_field.text()) # Шаг интегрирования
phi = float(self.moonPhi.text()) # Фаза для Луны
max_cnt = int(self.K_step_model.text()) # Кол-во шагов интегрирования
name_etude = self.chosenEtudeLabel.text() # Название этюда

razm = 4.216424392e7 # Для обезразмеривания
koff = 7.29e-5 # Для обезразмеривания

plSystem = PlanetSystem([])
for i in self.fileData:
    if(i['type'] == 'planet'):
        ksi_, eta_, zeta_ = [i["x"] / razm, i["y"] / razm, i["z"] / razm]

        V_ksi, V_eta, V_zeta = [i["Vx"]  / (koff * razm), i["Vy"]  / (koff * razm),
i["Vz"]  / (koff * razm)]
        R =  i["R"] / razm
        M = i["m"]
        color = i["color"]

        if(i["name"] == 'Earth'):
            ki = 0.9999999998 # Для обезразмеривания
        else:
```

```
# Key
ki = 0.01232376679 # Для обезразмеривания

if(name_etude not in ('Уход с орбиты.json')):
    # Меняем фазу Луны
    ksi_1 = ksi_ * np.cos(phi) - eta_ * np.sin(phi)
    eta_1 = ksi_ * np.sin(phi) + eta_ * np.cos(phi)

    V_ksi1 = V_ksi * np.cos(phi) - V_eta * np.sin(phi)
    V_eta1 = V_ksi * np.sin(phi) + V_eta * np.cos(phi)

    ksi_, eta_, V_ksi, V_eta = ksi_1, eta_1, V_ksi1, V_eta1

plSystem.add_new_planet(Planet(ksi_, eta_, zeta_, V_ksi, V_eta, V_zeta, ki,
M, R, color))
print(f'{ksi_} {eta_} {V_ksi} {V_eta} {ki} {R}')

else:
    ksi_, eta_, zeta_ = [i["x"] / razm, i["y"] / razm, i["z"] / razm]
    V_ksi, V_eta, V_zeta = [i["Vx"]  / (koff * razm), i["Vy"]  / (koff * razm),
i["Vz"]  / (koff * razm)]
    R =  6 * razm / razm
    M = i["m"]
    F_dv =  i["F_dv"]
    K_stop_engine_ = i["K_stop_engine"]

    print(f'{ksi_} {eta_} {V_ksi} {V_eta} {ki} {R}')

    plSystem.add_spaceship(SpaceShip(ksi_, eta_, zeta_, V_ksi, V_eta, V_zeta,
M, R, color, F_dv, K_stop_engine_))

OnOffEngine = [
    {'start': 0, 'stop': int(K_stop_engine_), 'is_started': False, 'is_stoped':
False},
]
```

Московский авиационный институт
(Национальный исследовательский университет)

Дипломная работа
на тему

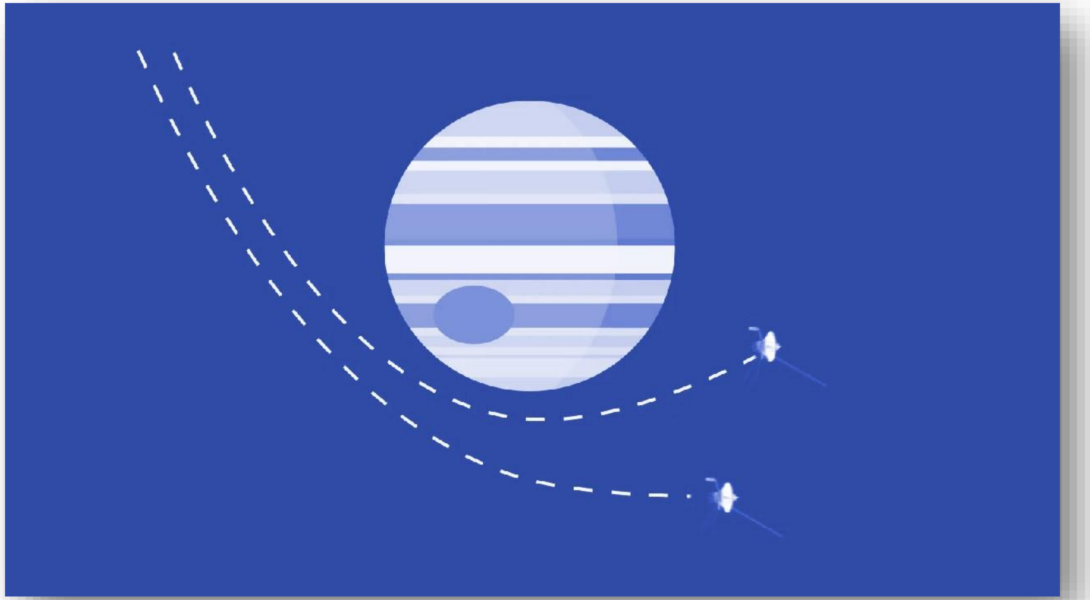
ПОСТРОЕНИЕ МАРШРУТА КОСМИЧЕСКОГО КОРОБЛЯ С ИСПОЛЬЗОВАНИЕМ ЭФФЕКТА ОБЕРТА

Дипломник: Михеев Кирилл Вячеславович
Научный руководитель: Беличенко Михаил Валериевич

Москва, 2022

Проблема

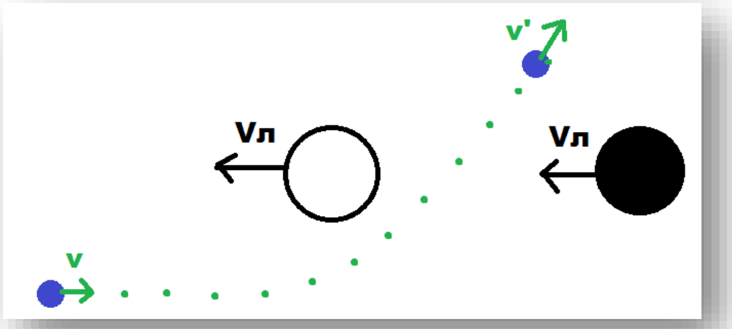
- **Изучение** солнечной системы на дальних расстояниях
- Поиск **эффективных** маршрутов
- **Оптимизация** затрат топлива
- Получение **максимальных скоростей** на траекториях движения спутников



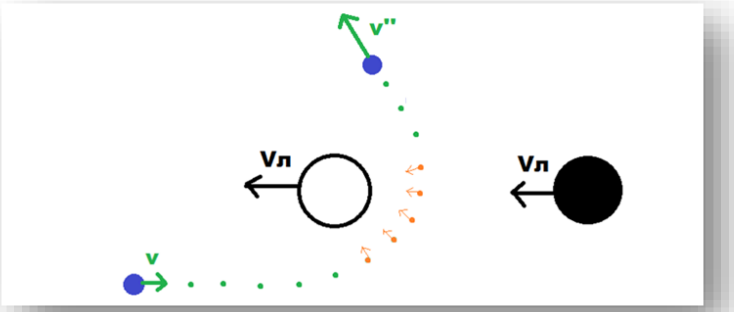
Постановка задачи

Задача моделирования стратегий, построение оптимальной траектории выхода спутника на дальние рубежи, оценка эффективности

- Уход с орбиты посредством двигателей
- Использование гравитационного маневра
- Использование гравитационного маневра с использованием эффекта Оберта

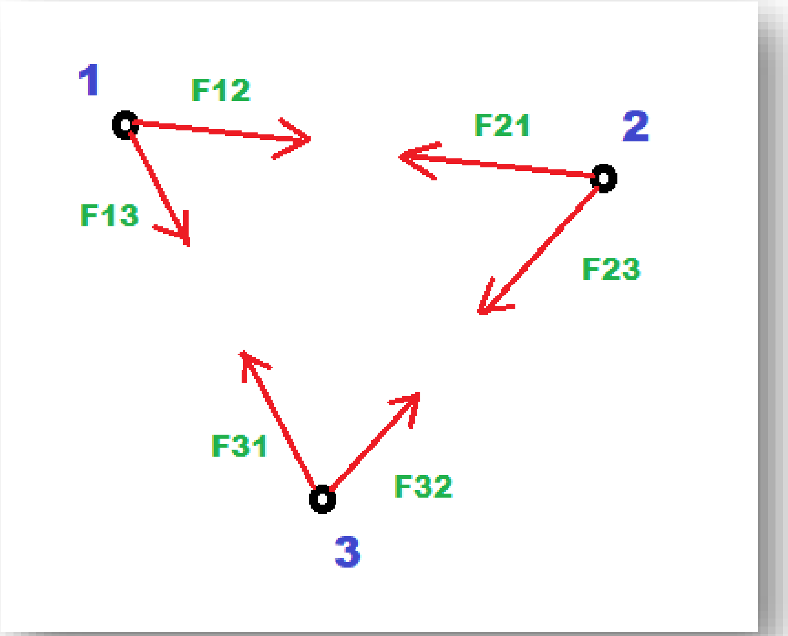


Гравитационный маневр



С эффектом Оберта

Уравнения движения



Симуляция космической системы

(Масса спутника << массы Земли или Луны)

Сила притяжения каждого тела друг к другу определяется следующей **формулой**:

$$\vec{F}_{ij} = \frac{\gamma m_i m_j (\vec{r}_j - \vec{r}_i)}{\left|-\vec{r}_j + \vec{r}_i\right|^3}$$

Для нахождения **ускорений** воспользуемся вторым законом Ньютона:

$$m_i \overline{w}_i = \sum_j \frac{\gamma m_i m_j * (\vec{r}_j - \vec{r}_i)}{\left|-\vec{r}_j + \vec{r}_i\right|^3}$$

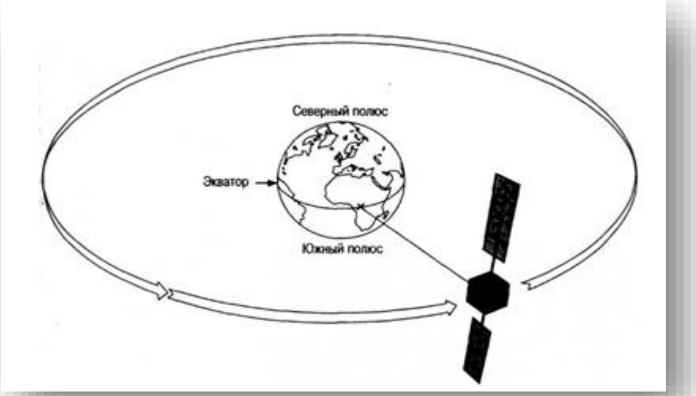
Подготовка

1. Спутник стартует с **геостационарной орбиты**

2. Используются **обезразмерные** уравнения движения

3. **Сила** двигателя считаем **константой** равной 0.01 *у.е*.

4. При выходе с орбиты Земли двигатель работает **вдоль вектора скорости** космического аппарата

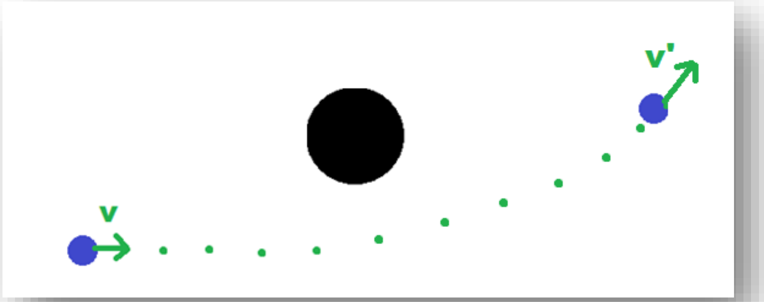


Геостационарная орбита

$$R_O = 4.216e7, \qquad \Omega = \frac{2 * \pi}{286164{,}091}$$

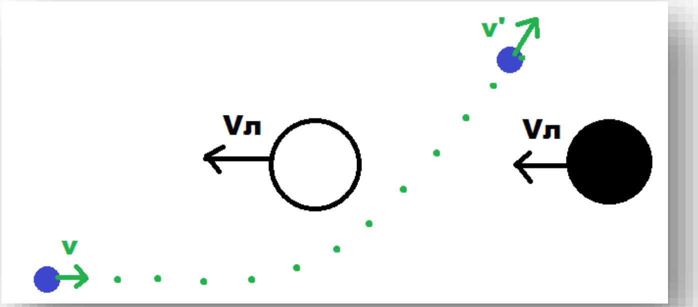
Значения радиуса орбиты и омеги для получения безразмерных величин

Эффект Оберта



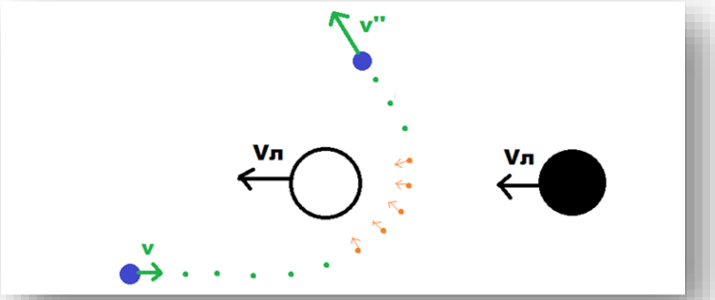
- Луна без движения
- Спутник пролетает рядом

Финальная скорость ***изменила направление***



- Луна двигается
- Спутник пролетает рядом

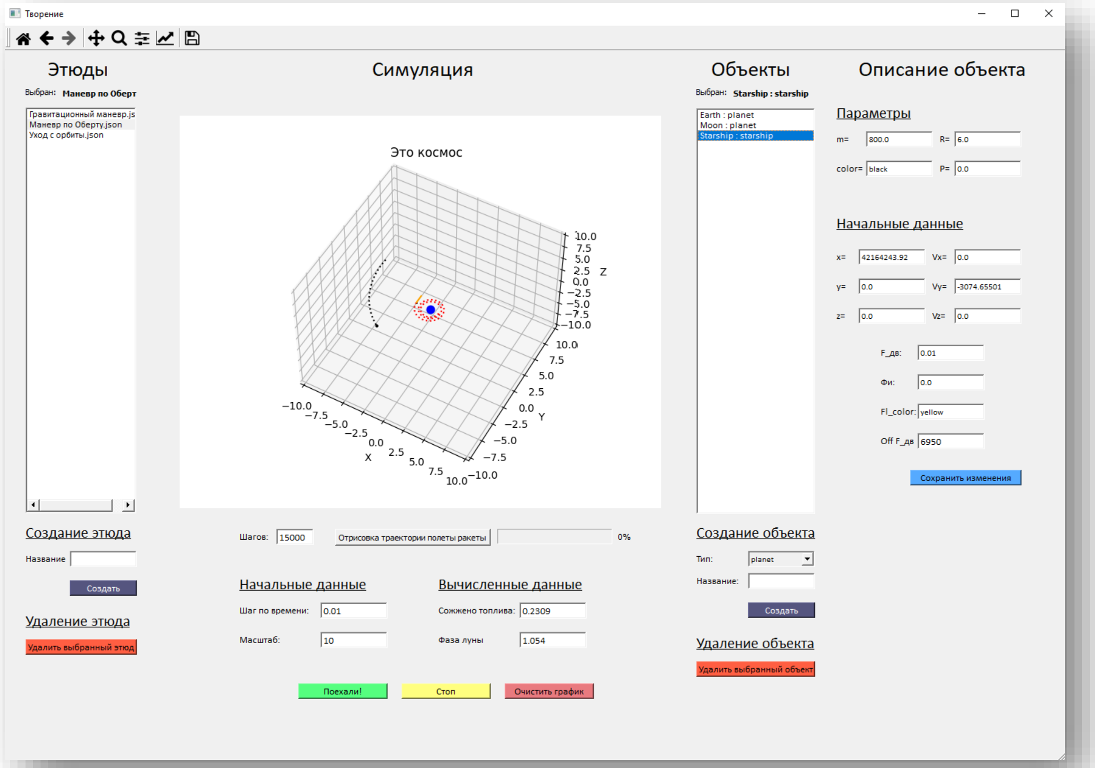
Финальная скорость может быть увеличена *за счёт* ***орбитального импульса***



- Луна двигается
- Спутник включает двигатель в ближайшей к Луне точке

Финальная скорость ***значительно увеличена*** *за счёт* ***орбитального импульса*** *и* ***эффекта Оберта***

Разработка приложения



Разработанное программное обеспечение

Возможности

- **Управление** над **созданием** этюдов, **управлением** над **объектами.**

  Настройка начальных данных симуляции: Шаг интегрирования, масштаб

- Ввод **начальных данных** для небесных тел:

  - Начальные координаты
  - Скорость движения
  - Сила двигателя (для спутника)
  - Масса тела
  - Радиус объекта
  - Фаза старта Луны

- Запуск в **двух режимах**: быстрая от рисовка траектории спутника, полная симуляция с анимацией движения космических тел

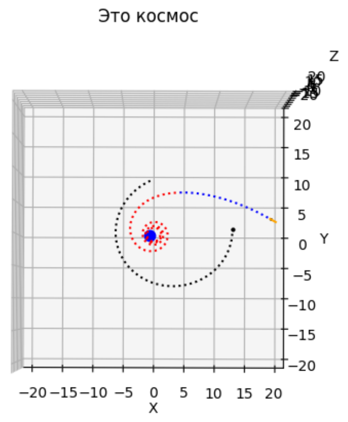
Моделирование этюдов. Первый

Выход с орбиты при помощи двигателя.

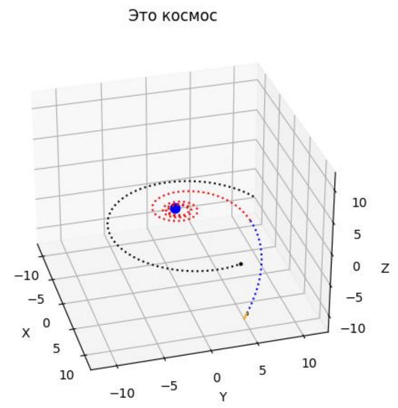
Результаты:

- Финальная скорость: **0.1852**

- Затрачено топлива: **1.44308 *у.е.***



Вид сверху



Вид сбоку

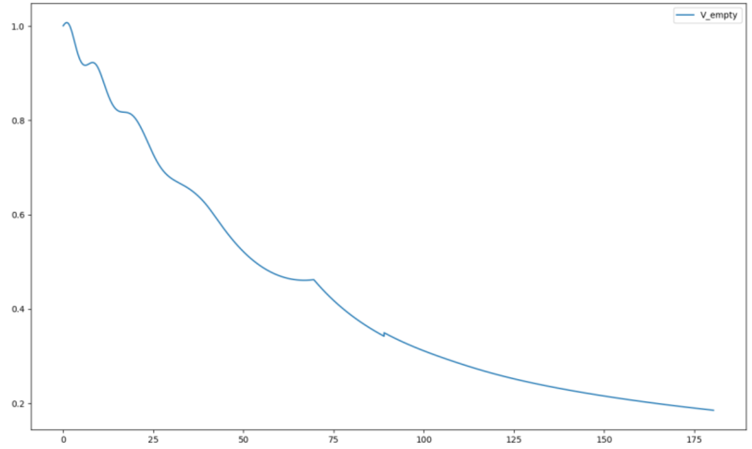


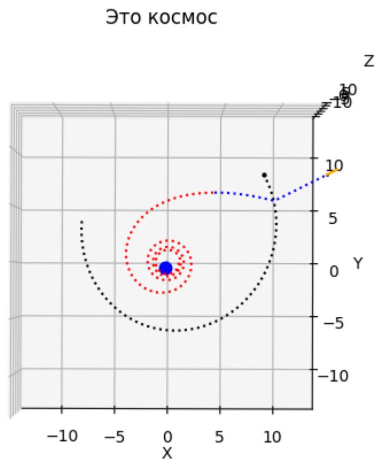
График изменения скорости спутника

Моделирование этюдов. Второй

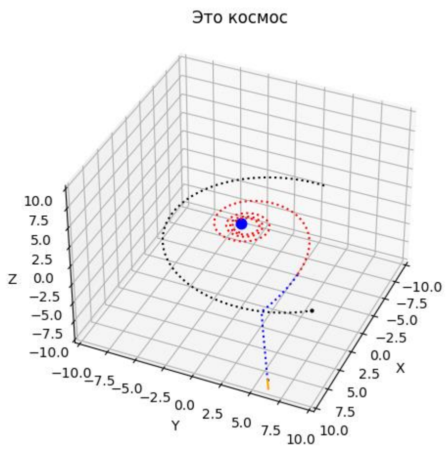
Использование гравитационного маневра. Ищем фазу Луны и учитываем компенсацию топлива.

Результаты:

- Финальная скорость: **0.4464**
- Затрачено топлива: **1.44308** ***у.е.***



Вид сверху



Вид сбоку

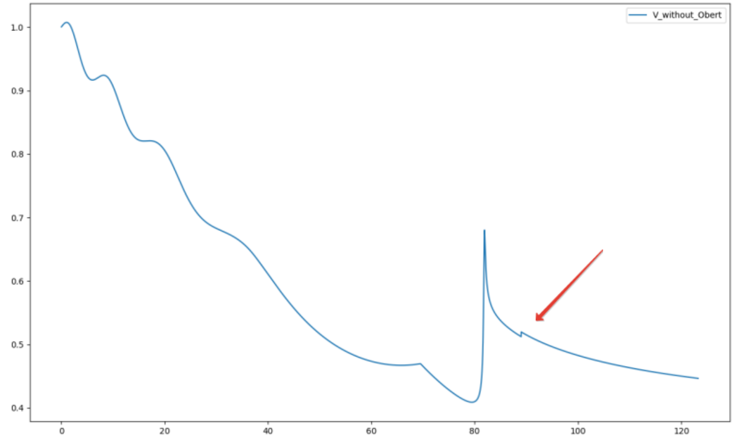


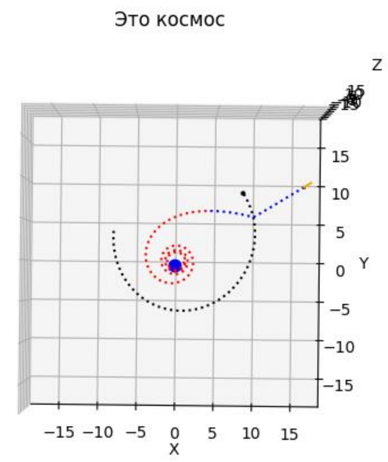
График изменения скорости спутника

Моделирование этюдов. Третий

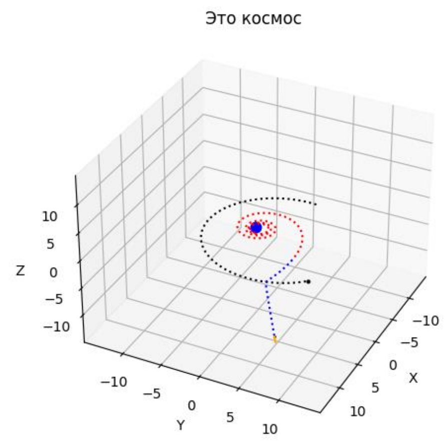
Использование гравитационного маневра с эффектом Оберта. Ищем угол с максимальной скоростью.

Результаты:

- Финальная скорость: **0.4759**
- Затрачено топлива: **1.44308** ***у.е.***



Вид сверху



Вид сбоку

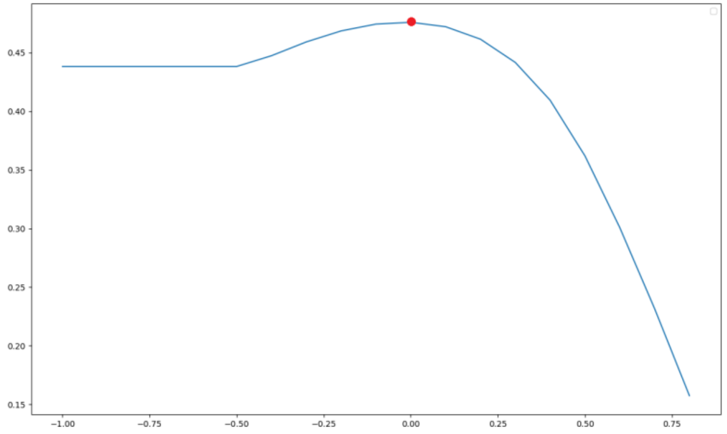


График зависимости угла и финальной скорости

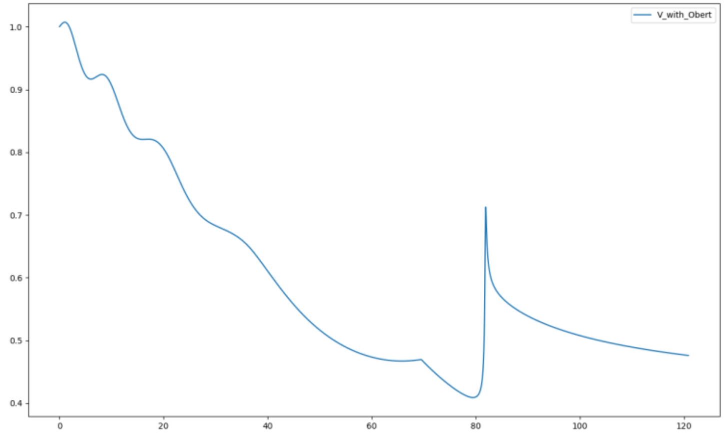


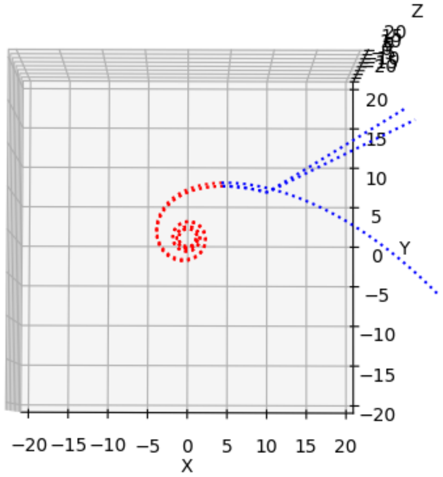
График изменения скорости спутника

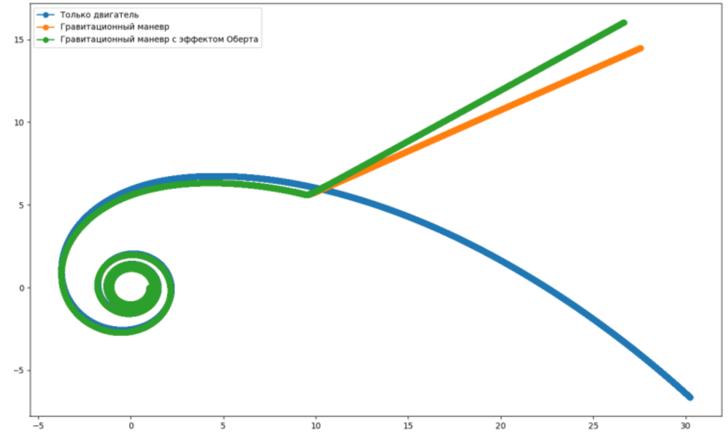
Сравнение результатов

- Качество каждого этюда определяли по финальной скорости на расстоянии Земли

$$r = \frac{\sqrt{R_O^2 * 1000}}{R_O} = 31.6$$

Дистанция фиксации финальной скорости

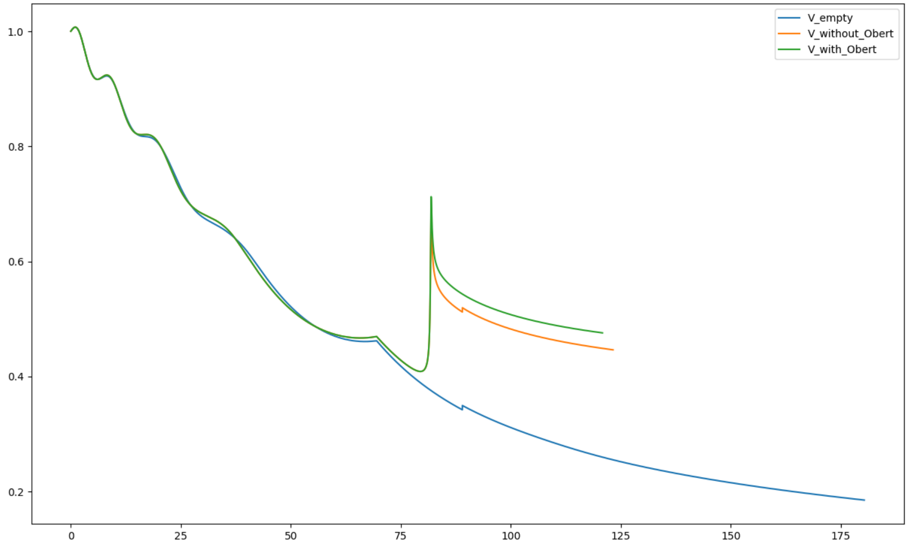




Траектории 3 этюдов

Этюд	Скорость	Затрачено топлива
Первый	**0.1852**	**1.44308 *у.е.***
Второй	**0.4464**	**1.44308 *у.е.***
Третий	**0.4759**	**1.44308 *у.е.***

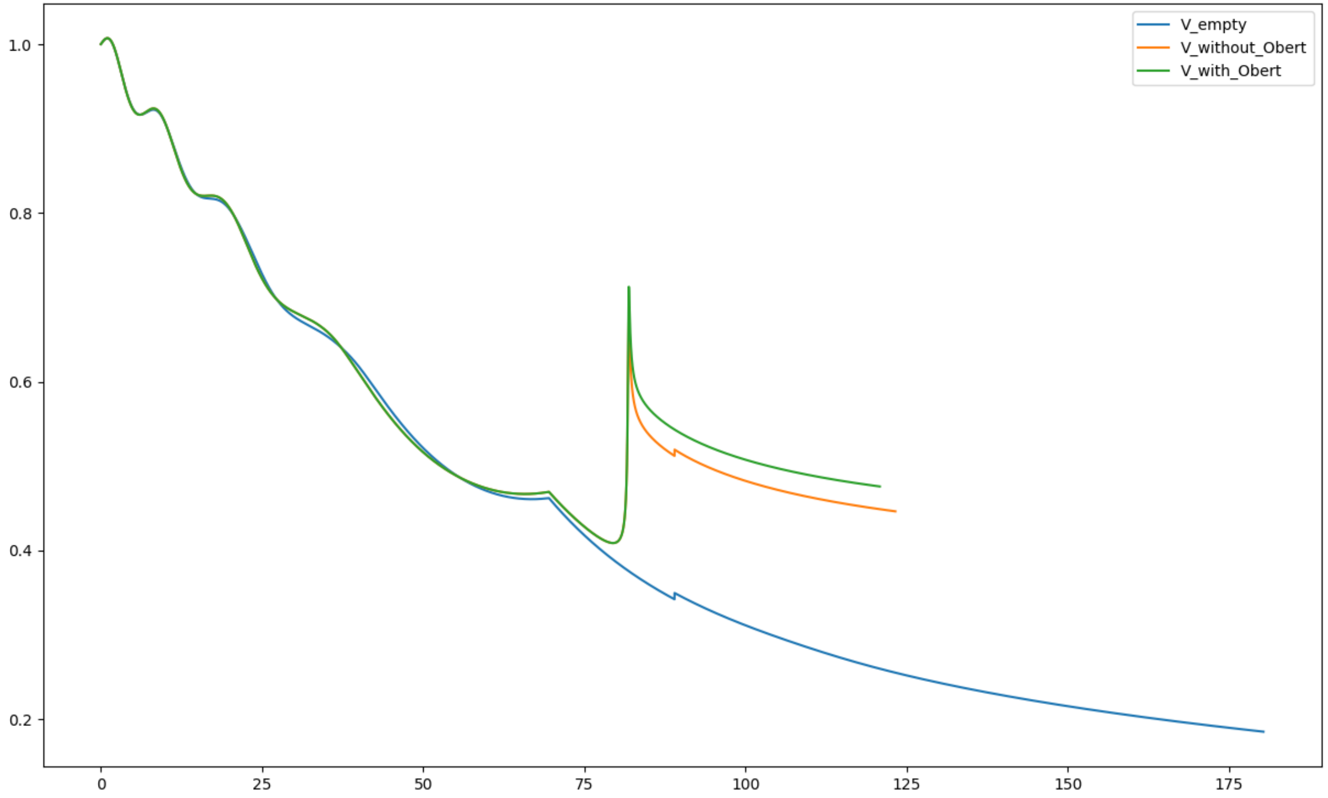
Таблица скоростей



Изменение скоростей 3х этюдов

Заключение

По итогу получаем, что при одинаковой затрате топлива, **скорость** при использовании гравитационного маневра **с эффектом Оберта** выше. Следовательно его использование **эффективнее**.



Изменение скоростей 3х этюдов

Спасибо за внимание!