МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра вычислительной математики

НИКОЛАЮК Никита Валентинович

ИСПОЛЬЗОВАНИЕ ГРАВИТАЦИОННЫХ МАНЁВРОВ ПРИ МЕЖПЛАНЕТНЫХ ПЕРЕЛЁТАХ

Дипломная работа

Научный руководитель:
канд. физ.-мат. наук
доцент А.В Тетерев

Допущена к защите
«__»_____ 2025 г.
Заведующий кафедрой вычислительной математики
канд. физ.-мат. наук, доцент В.И. Репников

Минск, 2025

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Дипломная работа, 62 с., 14 рис., 6 приложений, 12 источников.

ГРАВИТАЦИОННЫЕ МАНЕВРЫ, ЛЕТАТЕЛЬНЫЙ АППАРАТ, ЗАДАЧА ЛАМБЕРТА, МЕЖПЛАНЕТНЫЕ ПЕРЕЛЕТЫ, МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ, ОПТИМИЗАЦИЯ ТРАЕКТОРИЙ, КОСМИЧЕСКИЕ МИССИИ.

Объект исследования — математические модели и алгоритмы, описывающие траектории космических аппаратов при выполнении гравитационных маневров в гравитационных полях планет Солнечной системы.

Цель исследования — разработка программного инструмента для моделирования межпланетных перелетов с использованием гравитационных маневров, включая анализ их эффективности, а также оптимизацию траекторий на основе задачи Ламберта.

Методы исследования включают аналитическое решение уравнений движения в рамках задачи двух тел, применение численных методов для моделирования гиперболических траекторий — методы Эйлера и Рунге-Кутта, а также использование алгоритмов решения задачи Ламберта, в частности метода Иззо, для планирования оптимальных межпланетных перелетов.

В результате работы была успешно разработана математическая модель, описывающая влияние гравитационных маневров на траекторию и скорость летательного аппарата. Создано программное обеспечение, предоставляющее инструменты для визуализации траекторий перелетов, анализа изменений скорости и оптимизации параметров миссий. В ходе исследования проведена имитация двух ключевых сценариев — маневра ускорения и маневра торможения — с использованием гравитационного поля Юпитера, повторены полеты космических аппаратов «Новые горизонты», «Вояджер-2», а также найден следующий возможный полет, который повторяет путь «Вояджера-2».

Автор работы подтверждает, что приведенный в ней расчетно-аналитический материал правильно и объективно отражает состояние исследуемого процесса, а все заимствованные из литературных и других источников теоретические, методологические и методические положения и концепции сопровождаются ссылками на их авторов.

_____

(подпись студента)

РЭФЕРАТ

Дыпломная праца, 62 с., 14 мал., 6 дадаткаў, 12 крыніц.

ГРАВІТАЦЫЙНЫЯ МАНЕЎРЫ, ЛЯТАЛЬНЫ АПАРАТ, ЗАДАЧА ЛАМБЕРТА, МІЖПЛАНЕТНЫЯ ПЕРАЛЁТЫ, МАТЭМАТЫЧНАЕ МАДЭЛЯВАННЕ, АПТЫМІЗАЦЫЯ ТРАЕКТОРЫЙ, КАСМІЧНЫЯ МІСІІ.

Аб’ект даследавання — матэматычныя мадэлі і алгарытмы, якія апісваюць траекторыі касмічных апаратаў пры выкананні гравітацыйных манеўраў у гравітацыйных палях планет Сонечнай сістэмы.

Мэта даследавання — пабудова праграмнага інструмента для мадэлявання міжпланетных пералётаў з выкарыстаннем гравітацыйных манеўраў, уключаючы аналіз іх эфектыўнасці, а таксама аптымізацыю траекторый на аснове задачы Ламберта.

Метады даследавання ўключаюць аналітычнае рашэнне ўраўненняў руху ў рамках задачы двух цел, прымяненне лікавых метадаў для мадэлявання гіпербалічных траекторый — метады Эйлера і Рунге-Кута, а таксама выкарыстанне алгарытмаў рашэння задачы Ламберта, у прыватнасці метаду Ізза, для планавання аптымальных міжпланетных пералётаў.

У выніку працы была паспяхова распрацавана матэматычная мадэль, якая апісвае ўплыў гравітацыйных манеўраў на траекторыю і хуткасць лятальнага апарата. Створанае праграмнае забеспячэнне прадастаўляе інструменты для візуалізацыі траекторый пералётаў, аналізу змяненняў хуткасці і аптымізацыі параметраў місій. У ходзе даследавання праведзена імітацыя двух ключавых сцэнарыяў — манеўру паскарэння і манеўру тармажэння — з выкарыстаннем гравітацыйнага поля Юпітэра, паўтораны палёты касмічных апаратаў «Новыя гарызонты», «Вояджэр-2», а таксама знойдзены наступны магчымы палёт, які паўтарае шлях «Вояджэра-2».

Аўтар працы пацвярджае, што прыведзены ў ёй разлікова-аналітычны матэрыял правільна і аб’ектыўна адлюстроўвае стан даследаванага працэсу, а ўсе запазычаныя з літаратурных і іншых крыніц тэарэтычныя, метадалагічныя і метадычныя палажэнні і канцэпцыі суправаджаюцца спасылкамі на іх аўтараў.

(подпіс студэнта)

ANNOTATION

Thesis, 62 p., 14 ill., 6 appendices, 12 sources.

GRAVITY ASSISTS, SPACECRAFT, LAMBERT'S PROBLEM, INTERPLANETARY FLIGHTS, MATHEMATICAL MODELING, TRAJECTORY OPTIMIZATION, SPACE MISSIONS.

The object of research is mathematical models and algorithms describing spacecraft trajectories when performing gravitational maneuvers in the gravitational fields of the planets of the Solar system.

The purpose of the study is to develop a software tool for modeling interplanetary flights using gravitational maneuvers, including analysis of their effectiveness, as well as optimization of trajectories based on the Lambert's problem.

The research methods include the analytical solution of the equations of motion within the framework of the two-body problem, the use of numerical methods for modeling hyperbolic trajectories — Euler and Runge-Kutta methods, as well as the use of algorithms for solving the Lambert problem, in particular the Izzo method, for planning optimal interplanetary flights.

As a result of the work, a mathematical model has been successfully developed that describes the effect of gravitational maneuvers on the trajectory and speed of an aircraft. Created software that provides tools for visualizing flight paths, analyzing speed changes, and optimizing mission parameters. During the study, two key scenarios were simulated — an acceleration maneuver and a deceleration maneuver — using the gravitational field of Jupiter, the flights of the New Horizons and Voyager 2 spacecraft were repeated, and the next possible flight was found, which follows the path of Voyager 2.

The author of the work confirms that computational and analytical material presented in it correctly and objectively reproduces the picture of investigated process, and all the theoretical, methodological and methodical positions and concepts borrowed from literary and other sources are given references to their authors.

_____

(Student's signature)

ВВЕДЕНИЕ

Исследование и освоение космоса являются одной из наиболее захватывающих и перспективных областей научных исследований и технологического развития в современном мире. Вместе с тем, межпланетные перелеты исключительно важны для расширения наших знаний о Солнечной системе и возможности колонизации других планет.

В основе этих амбициозных полетов лежат не только технические и инженерные аспекты, но и уникальные методы космической навигации, такие как гравитационные маневры. Они представляют собой использование гравитационного поля планеты или другого космического объекта для изменения траектории и скорости космического аппарата.

В данной работе мы сосредоточимся на исследовании и анализе гравитационных маневров в контексте межпланетных перелетов. Наша цель — понять, как эти маневры могут быть использованы для оптимизации траекторий перелетов между планетами, уменьшения затрат топлива и времени перелета, а также повышения эффективности миссий.

В главе 1 рассматриваются теоретические основы гравитационных маневров: историческое развитие от первых концепций до современных миссий, физические принципы работы, ключевые факторы эффективности, а также преимущества и сопутствующие риски.

Глава 2 посвящена математическому моделированию маневров: анализу гиперболических траекторий в рамках задачи двух тел, трёхмерному моделированию гравитационных полей методом Рунге-Кутта, постановке и решению задачи Ламберта (включая алгоритм Иззо) для оптимизации межпланетных переходов.

В Главе 3 представлена реализация программного инструмента на языке Python: моделирование, визуализация и оптимизация межпланетных миссий, архитектура модели, тестирование на сценариях ускорения/торможения, верификация точности на траекториях миссий «Новые горизонты» и «Вояджер-2», а также проектирование, аналогичного «Вояджеру-2», полета в XXII веке.

ГЛАВА 1.
РОЛЬ ГРАВИТАЦИОННЫХ МАНЕВРОВ В МЕЖПЛАНЕТНЫХ ПЕРЕЛЕТАХ

1.1 Историческое развитие гравитационных маневров

Идея использования гравитации небесных тел для изменения траектории космических аппаратов зародилась задолго до начала космической эры. Первые теоретические предпосылки были заложены в работах Иоганна Кеплера и Исаака Ньютона, однако практическое применение гравитационных маневров стало возможным лишь с развитием ракетных технологий в середине XX века. Отправной точкой стал 1959 год, когда советская станция «Луна-3» впервые использовала гравитационное поле Луны для коррекции своей траектории, что позволило ей сфотографировать обратную сторону спутника Земли [12]. Этот успех доказал, что гравитационные маневры могут быть не только теоретической концепцией, но и рабочим инструментом космической навигации.

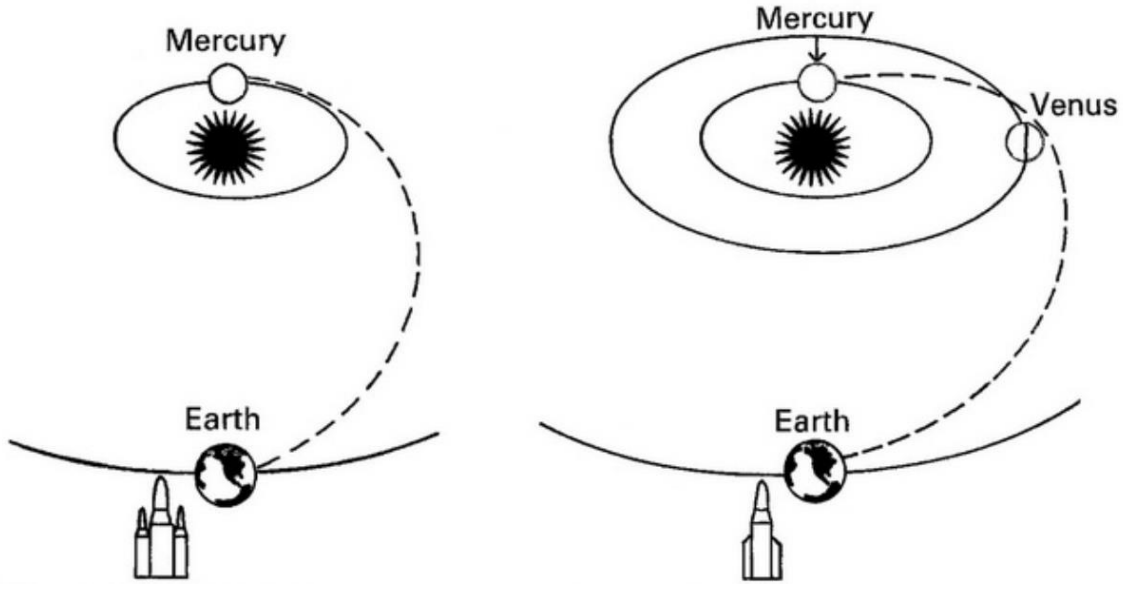


Рисунок 1.1 – Схематическое изображение вариантов запуска летательного аппарата к Меркурию

Настоящий прорыв произошел в 1974 году, когда аппарат NASA «Маринер-10» совершил гравитационный маневр у Венеры, чтобы достичь

Меркурия (рисунок 1.1) [12]. Это событие открыло эру межпланетных перелетов, где гравитация планет стала заменой топливным ресурсам. Пиком развития метода стали миссии «Вояджер-1» и «Вояджер-2», запущенные в 1977 году (рисунок 1.2) [12]. «Вояджер-2», например, выполнил серию маневров у Юпитера, Сатурна, Урана и Нептуна, что позволило ему не только исследовать все планеты-гиганты, но и покинуть Солнечную систему. Без гравитационных маневров такая миссия потребовала бы непомерных затрат топлива и осталась бы технически нереализуемой [1].

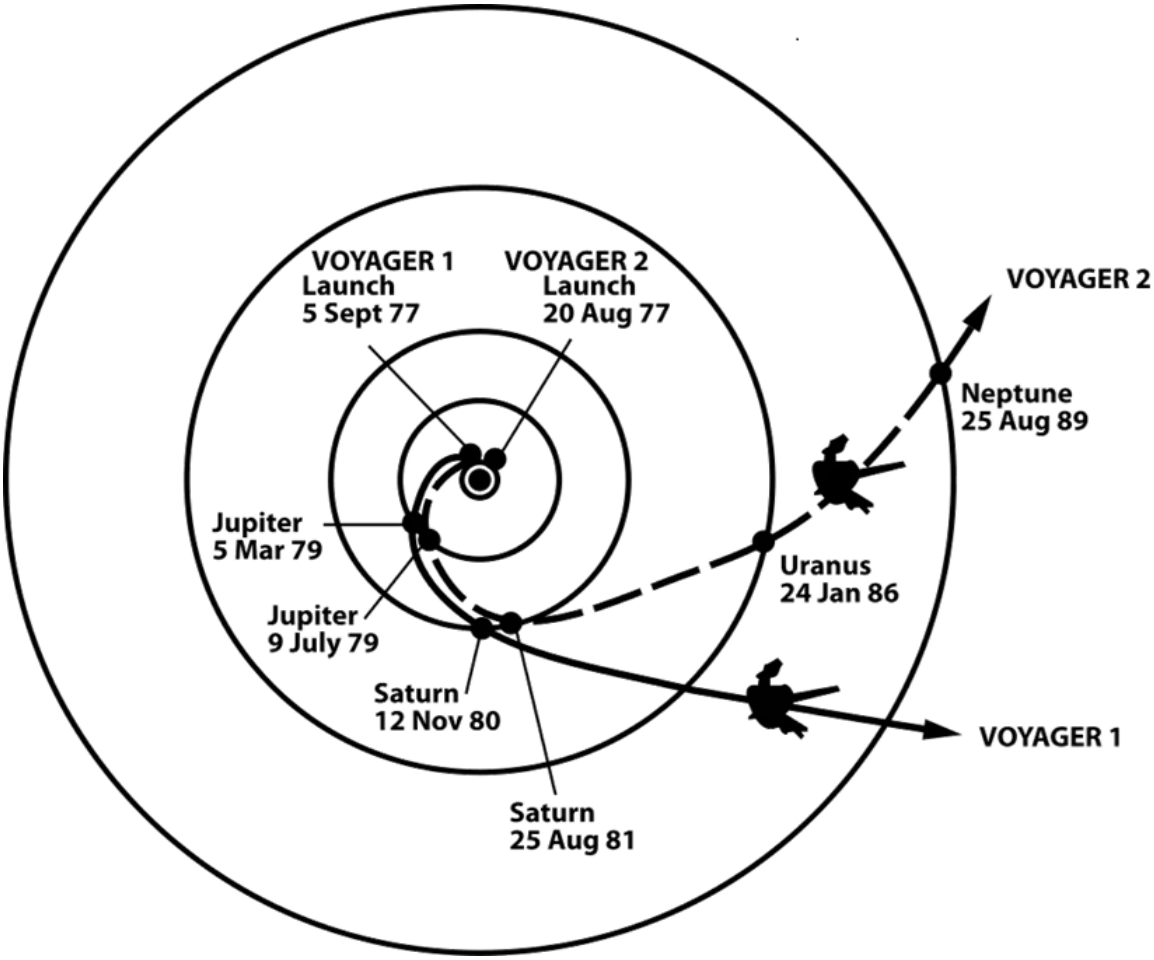


Рисунок 1.2 – Схематическое изображение траектории полета летательных аппаратов «Вояджер-1» и «Вояджер-2»

Другой пример — миссия «Новые горизонты», где маневр у Юпитера сократил время полета к Плутону с 12 до 9 лет [12]. Это позволило аппарату провести детальные исследования атмосферы и поверхности карликовой планеты. Подробнее рассмотрим этот маневр в разделе 3.4.

Современные миссии демонстрируют новые стратегии применения

гравитационных маневров. Зонд «Паркер Солар Проб» 7 раз использовал гравитацию Венеры для сближения с поверхностью Солнца (фотосфере) на рекордное расстояние 8.8 радиуса Солнца (6.1 миллиона километров, или 0.04 а.е.), и по состоянию на 2025 год он является самым быстрым искусственным объектом, созданным человеком. Им была достигнута гелиоцентрическая скорость 692 000 км/ч (192 км/с), составляющая 0.064% от скорости света (рисунок 1.3) [1].

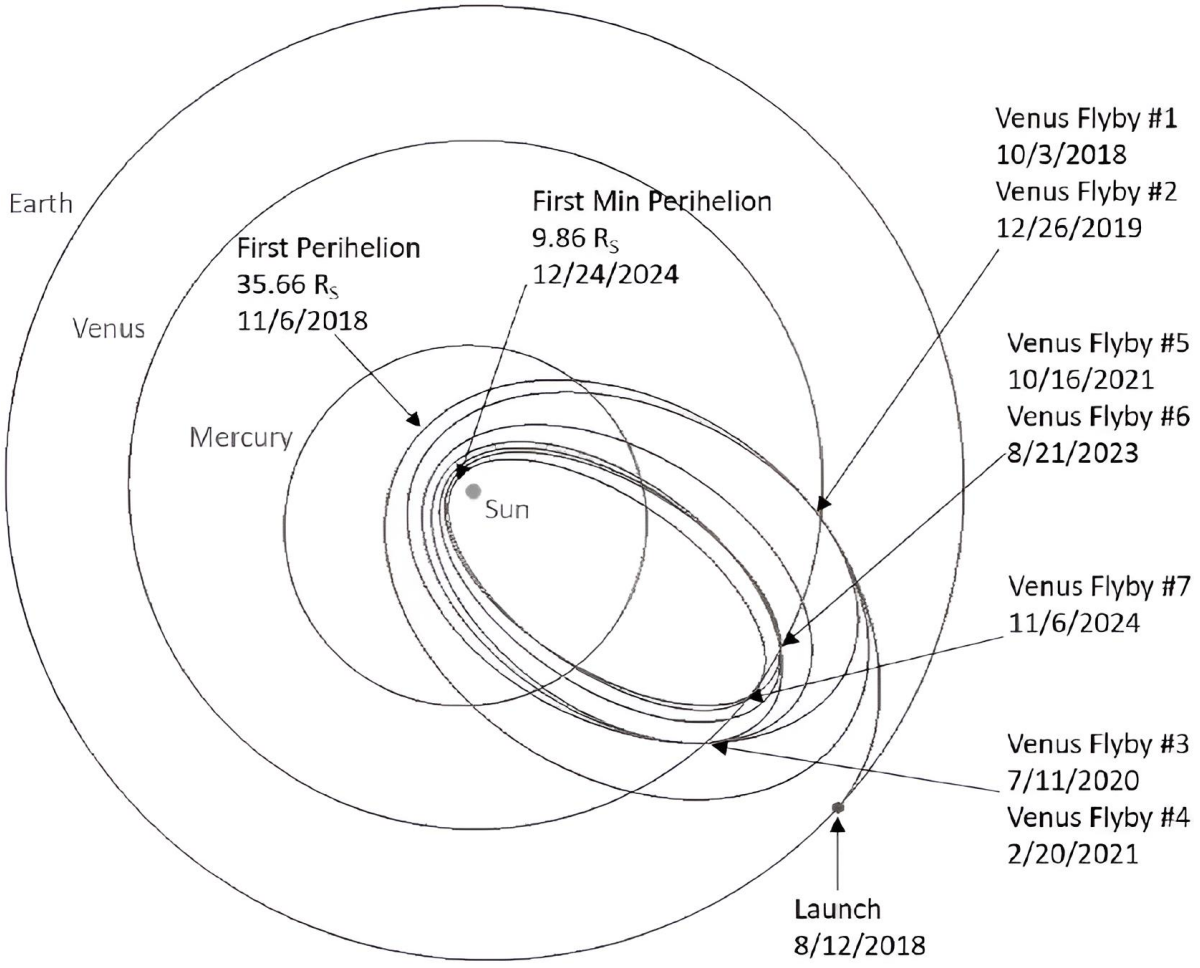


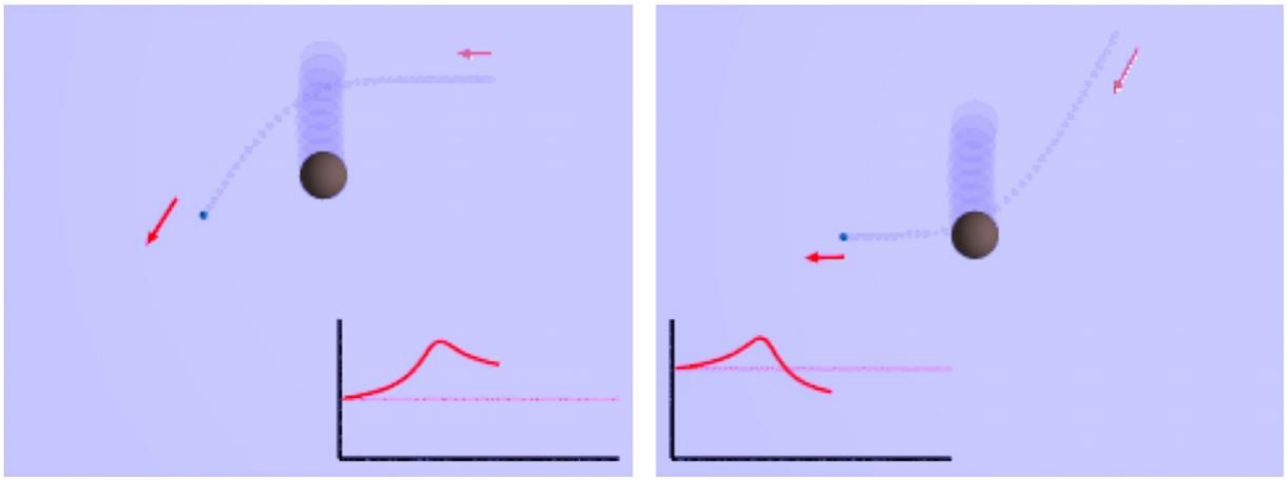
Рисунок 1.3 – Схематическое изображение траектории полета летательного аппарата Паркер Солар Проб

Перспективным направлением являются маневры у малых тел, таких как астероиды или спутники планет. Например, теоретическая миссия к спутнику Юпитера Европе может включать маневр у Ганимеда для коррекции траектории. Развитие алгоритмов оптимизации, таких как метод Иззо для решения задачи Ламберта, позволяет планировать сложные маршруты с

последовательным использованием нескольких планет [8].

1.2 Принципы и механизмы использования

Маневр основан на взаимодействии космического аппарата с гравитационным полем планеты. Когда аппарат приближается к небесному телу, его траектория искривляется под действием гравитации, что приводит к изменению скорости и направления движения. В системе отсчета, связанной с планетой, скорость аппарата остается постоянной по модулю, но меняет направление. Однако в гелиоцентрической системе отсчета происходит обмен энергией: если аппарат движется в направлении орбитального движения планеты, он ускоряется; если против — замедляется (рисунок 1.4) [1]. Например, при маневре у Юпитера аппарат «Вояджер-2» увеличил свою скорость на 15 км/с, что эквивалентно работе двигателей в течение нескольких месяцев [12].



а **б**

Рисунок 1.4 – Изменение скорости и траекторий при гравитационных маневрах. а – маневр ускорения; б – маневр торможения

Эффективность маневра зависит от нескольких ключевых факторов. Во-первых, масса планеты играет критическую роль: чем массивнее тело, тем сильнее его гравитационное воздействие. К примеру, Юпитер, будучи крупнейшей планетой Солнечной системы, обеспечивает максимальный прирост скорости для космического аппарата. Во-вторых, скорость сближения аппарата должна быть оптимально направлена так, чтобы гиперболическая траектория способствовала максимальному обмену энергией. Кроме того,

минимальное расстояние пролета (перицентр) имеет значение: чем ближе аппарат подходит к планете, тем сильнее гравитационное воздействие. Ключевым условием успеха является точный расчет точки сближения с планетой, которая определяет угол отклонения и итоговый прирост скорости [2].

1.3 Преимущества, ограничения и риски

Использование гравитационных маневров — хоть и не простой, однако очень эффективный подход в планировании и выполнении межпланетных перелетов, обладающий рядом преимуществ.

Главное преимущество гравитационных маневров — значительная экономия топлива. Например, миссия «Кассини-Гюйгенс», изучавшая Сатурн, сэкономила 50% топлива благодаря четырем маневрам у Венеры и Земли [3]. Это позволило увеличить полезную нагрузку за счет научного оборудования, такого как спектрометры и камеры высокого разрешения. Кроме того, экономия топлива позволяет снижать общую массу космического аппарата, что существенно упрощает его запуск и уменьшает стоимость миссии.

Гравитационные маневры позволяют осуществлять более сложные и амбициозные научные задачи, расширяя горизонты астрономии и планетологии, открывая доступ к траекториям, недостижимым при использовании только двигателей. Зонд «Улисс», запущенный в 1990 году, использовал гравитацию Юпитера для выхода на орбиту, перпендикулярную плоскости эклиптики, что дало возможность впервые исследовать полярные зоны Солнца (рисунок 1.5) [7].

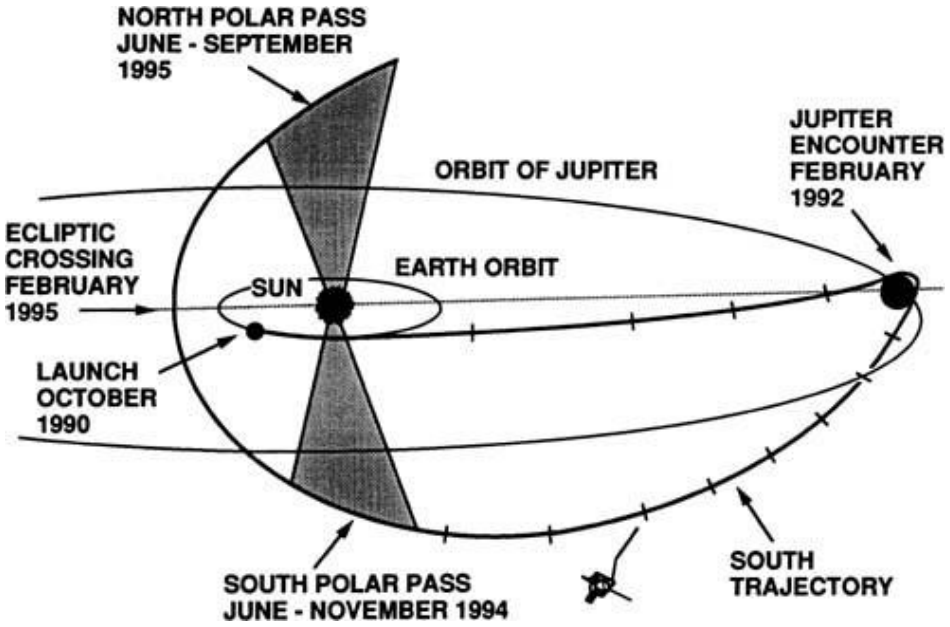


Рисунок 1.5 – Схематическое изображение траектории полета летательного аппарата Улисс

Несмотря на преимущества, гравитационные маневры имеют ряд ограничений. Их эффективность зависит от взаимного расположения планет, что создает узкие «окна запуска». Например, для полета к Урану с использованием маневра у Юпитера благоприятная конфигурация возникает раз в 14 лет [2]. Это ограничение требует тщательного долгосрочного планирования миссий, что может затруднить их реализацию в краткосрочной перспективе. Кроме того, траектории, использующие гравитационные маневры, могут быть более сложными и долгими, что увеличивает общее время выполнения миссии.

Риски включают неточности расчетов, которые могут привести к потере аппарата. В 1992 году зонд «Марс Обсервер» не смог выйти на орбиту Марса из-за ошибки в гравитационных расчетах [7]. Дополнительную угрозу представляют микрометеориты и космический мусор, особенно вблизи планет с кольцами, таких как Сатурн.

ГЛАВА 2.
МОДЕЛИРОВАНИЕ ГРАВИТАЦИОННЫХ МАНЕВРОВ

2.1 Постановка задачи

Разработка траекторий космических аппаратов с использованием гравитационных манёвров требует построения точной математической модели, представляющую движение тела под воздействием гравитационных сил планет и Солнца. Целью моей работы является систематизация и анализ моделей, описывающих поведение космического аппарата на межпланетных перелётах с учётом влияния массивных тел.

В первую очередь необходимо рассмотреть гиперболические траектории вблизи планет, поскольку именно они лежат в основе гравитационного манёвра. Необходимо учесть фундаментальные физические законы, такие как закон всемирного тяготения и сохранения импульса, а также построить математические выражения, описывающие ускорение и изменение траектории аппарата при его сближении с планетой.

Кроме того, ключевым этапом расчёта оптимальной траектории является решение задачи Ламберта, как основы для планирования переходов между орбитами планет. Её решение позволяет находить возможные траектории между двумя точками пространства за заданный промежуток времени.

Таким образом, основная задача — построение комплексной математической модели, которая должна объединять описание гиперболических траекторий вблизи планет, возникающих при пролёте космического аппарата в поле их тяготения, и траекторию межпланетного перелёта между орбитами двух планет. Это позволяет учитывать как локальные возмущения движения при сближении с планетой, так и глобальные параметры маршрута, определяющие общее перемещение в гравитационном поле Солнца и других тел системы.

2.2 Гиперболические траектории вблизи планет

Физическая основа гравитационных манёвров базируется на законах небесной механики, в первую очередь на законе всемирного тяготения Ньютона. Согласно этому закону, сила притяжения между двумя телами пропорциональна произведению их масс и обратно пропорциональна квадрату

расстояния между ними. При приближении космического аппарата к планете гравитационное взаимодействие между ними становится доминирующим фактором, определяющим траекторию полёта. Важно отметить, что при этом сохраняется общий импульс системы «планета-аппарат», но из-за огромной разницы в массах изменение скорости планеты оказывается ничтожно малым, в то время как аппарат может получить значительное приращение скорости [2].

Для точного расчёта параметров гравитационного манёвра необходимо учитывать сложную динамику движения. Траектория аппарата вблизи планеты представляет собой гиперболу, параметры которой определяются начальной скоростью аппарата, расстоянием до планеты и их взаимным расположением. Математическое моделирование такого движения требует решения системы дифференциальных уравнений, учитывающих гравитационное воздействие как со стороны планеты, так и со стороны Солнца. На практике это реализуется с помощью численных методов, которые позволяют шаг за шагом рассчитывать изменение координат и скорости аппарата.

Для примера рассмотрим двумерный случай манёвра вблизи Юпитера. Благодаря своей огромной массе Юпитер способен существенно изменять траекторию пролетающих мимо него космических аппаратов.

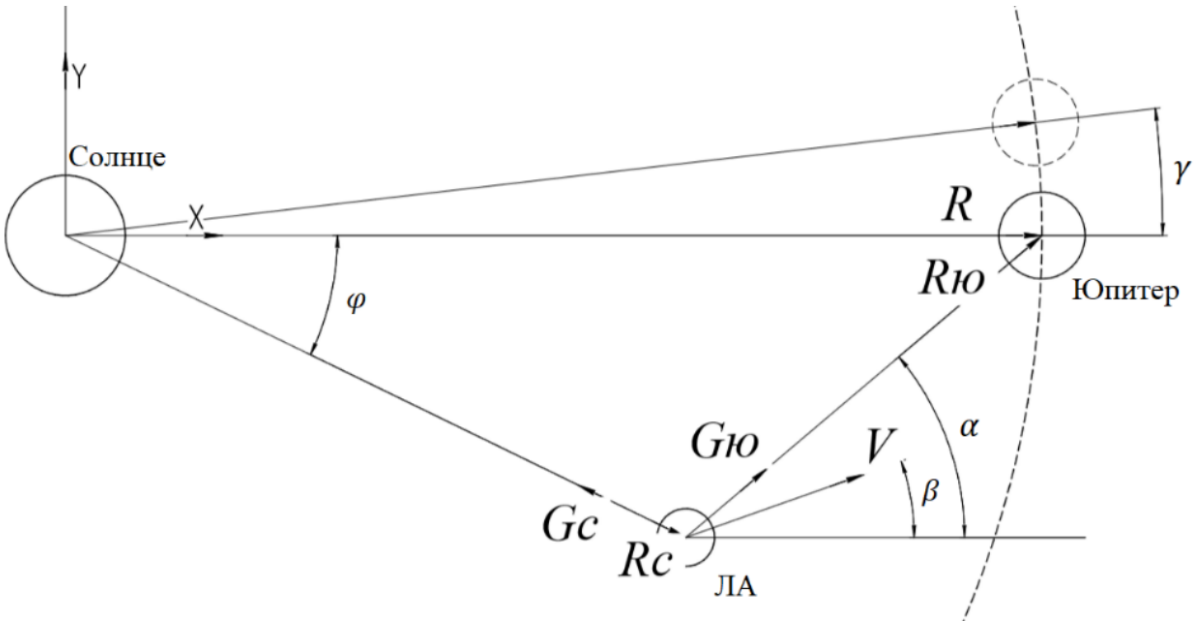


Рисунок 2.1 – Схема действующих сил

На рисунке 2.1 представлены действующие силы. Основой для расчёта движения космического аппарата вблизи планет служит закон всемирного тяготения Ньютона. Исходная формула гравитационного взаимодействия

между двумя телами имеет вид:

$$F = G\frac{mM}{R^2},$$

где G — гравитационная постоянная, M и m — массы взаимодействующих тел, R — расстояние между ними.

При рассмотрении движения аппарата массой m вблизи планеты массой M, действующая на аппарат сила вызывает ускорение:

$$a = \frac{F}{m} = G\frac{M}{R^2}.$$

Для описания орбитального движения удобно перейти к угловым характеристикам. Угловой момент L системы сохраняется:

$$L = mvR = const,$$

где v — орбитальная скорость. Комбинируя это с выражением для центростремительного ускорения $a = v^2/R$, получаем связь между скоростью и расстоянием:

$$v = \sqrt{\frac{GM}{R}}.$$

Угловая скорость ω (рад/с) связана с линейной скоростью соотношением $\omega = v/R$, что даёт:

$$\omega = \frac{d\gamma}{dt} = \sqrt{\frac{GM}{R^3}}.$$

Формула гравитационного взаимодействия между двумя телами получит вид:

$$ma = G\frac{mM}{R^3}.$$

Для определения орбиты Юпитера воспользуемся следующим соотношением, для чего преобразуем выражение $ma = G\frac{mM_c}{R^3}$, получим:

$$\begin{cases} dx_{ю} = lcos(d\gamma) \\ dy_{ю} = lcos(d\gamma) \end{cases},$$

$$\begin{cases} x_{ю} = x_0 + dx_{ю} \\ y_{ю} = y_0 + dy_{ю} \end{cases},$$

где $l = d\gamma R$.

Проекции ускорений на оси:

$$OX : f\left(\frac{M_{\text{ю}}}{R_{\text{ю}}^2}\cos(\alpha)-\frac{M_c}{R_c^2}\cos(\varphi)\right)=\frac{dv_x}{dt},$$

$$OY : f\left(\frac{M_{\text{ю}}}{R_{\text{ю}}^2}\sin(\alpha)-\frac{M_c}{R_c^2}\sin(\varphi)\right)=\frac{dv_y}{dt}.$$

Изменение скорости:

$$\begin{cases} V_x = V_{ox} + dv_x \\ V_y = V_{oy} + dv_y \end{cases},$$

где $V_{ox} = Vcos(\beta)$, $V_{oy} = Vsin(\beta)$, β — угол между направлением скорости и осью OX.

Изменение координат за время dt:

$$\begin{cases} dx = V_x dt \\ dy = V_y dt \end{cases},$$

$$\begin{cases} x = x_0 + dx \\ y = y_0 + dy \end{cases}.$$

Расстояния до Юпитера ($R_{\text{ю}}$) и Солнца (R_c):

$$R_{\text{ю}} = \sqrt{(x_{\text{ю}} - x)^2 + (y_{\text{ю}} - y)^2}\ ;$$

$$R_c = \sqrt{x^2 + y^2}\ .$$

Определение углов сводиться к их геометрическому определению через координаты известных точек с помощью функции арктангенса:

$$\varphi = \arctan\left(\frac{y}{x}\right);$$

$$\alpha = \arctan\left(\frac{y_{\text{ю}} - y}{x_{\text{ю}} - x}\right).$$

Такой подход позволяет шаг за шагом моделировать траекторию космического аппарата, учитывая гравитационное влияние всех значимых тел в системе [2].

2.3 Трехмерная модель гравитационного манёвра

Для повышения точности моделирования гравитационного манёвра и перехода от упрощенной двумерной схемы к более реалистичному описанию межпланетного полёта, была разработана и реализована модель движения космического аппарата в трёхмерном пространстве с учётом гравитационного

влияния всех значимых тел Солнечной системы. Такой подход позволяет более точно учитывать как эффект сближения с планетой, так и общее гравитационное поле, формируемое массивными телами, прежде всего Солнцем и крупнейшими планетами-гигантами, такими как Юпитер.

В отличие от описанной ранее двумерной модели, использующей метод Эйлера, в данной работе применяется более точный численный метод интегрирования — метод Рунге-Кутта четвёртого порядка. Этот метод широко используется в задачах небесной механики и динамики, поскольку обеспечивает хорошее сочетание точности и вычислительной стабильности. Метод Рунге-Кутта позволяет аппроксимировать решение дифференциальных уравнений, описывающих движение аппарата под действием переменных сил, с высокой степенью точности без необходимости чрезмерно уменьшать временной шаг. На каждом временном интервале Δt алгоритм выполняет следующие шаги:

$$k_{1r} = \overrightarrow{v_n} \qquad k_{1a} = \vec{a}(t_n, \overrightarrow{r_n}, \overrightarrow{v_n})$$

$$k_{2r} = \overrightarrow{v_n} + \frac{\Delta t}{2} k_{1a} \qquad k_{2a} = \vec{a}(t_n + \frac{\Delta t}{2}, \overrightarrow{r_n} + \frac{\Delta t}{2} k_{1r}, \overrightarrow{v_n} + \frac{\Delta t}{2} k_{1a})$$

$$k_{3r} = \overrightarrow{v_n} + \frac{\Delta t}{2} k_{2a} \qquad k_{3a} = \vec{a}(t_n + \frac{\Delta t}{2}, \overrightarrow{r_n} + \frac{\Delta t}{2} k_{2r}, \overrightarrow{v_n} + \frac{\Delta t}{2} k_{2a})$$

$$k_{4r} = \overrightarrow{v_n} + \frac{\Delta t}{2} k_{3a} \qquad k_{4a} = \vec{a}(t_n + \frac{\Delta t}{2}, \overrightarrow{r_n} + \frac{\Delta t}{2} k_{3r}, \overrightarrow{v_n} + \frac{\Delta t}{2} k_{3a})$$

где $\vec{a}(t, \vec{r}, \vec{v})$ — функция ускорения (гравитационного воздействия), зависящая от времени, положения и скорости соответственно. Обновлённое положение и скорость находятся по формулам:

$$\overrightarrow{r_{n+1}} = \overrightarrow{r_n} + \frac{\Delta t}{6}(k_{1r} + 2k_{2r} + 2k_{3r} + k_{4r}),$$

$$\overrightarrow{v_{n+1}} = \overrightarrow{v_n} + \frac{\Delta t}{6}(a_{1a} + 2a_{2a} + 2a_{3a} + a_{4a}).$$

Расчёт гравитационного ускорения производится с учетом вклада от всех массивных тел, входящих в модель. Для этого используется закон всемирного тяготения Ньютона в векторной форме:

$$\vec{a} = -G \sum_{i=1}^{N} \frac{M_i(\vec{r} - \vec{r_i})}{|\vec{r} - \vec{r_i}|^3},$$

где $\vec{r}$ — положение аппарата, $\vec{r_i}$ — положение i-го тела, M_i — его масса, а G — гравитационная постоянная.

2.4 Задача Ламберта

Задача Ламберта, иногда называемая орбитальной краевой задачей, занимает центральное место в астродинамике и небесной механике. Она формулируется как задача определения орбиты космического аппарата, соединяющей две заданные точки в пространстве за фиксированное время полёта, при условии движения в гравитационном поле центрального тела. Математическая постановка задачи включает следующие параметры: гравитационный параметр центрального тела μ, радиус-векторы начального r1 и конечного r2 положений, а также время перелёта Δt. Решение задачи предполагает нахождение векторов начальной v1 и конечной v2 скоростей, удовлетворяющих уравнениям движения в рамках задачи двух тел. Несмотря на кажущуюся простоту, задача Ламберта не имеет аналитического решения в общем случае, что обусловлено эллиптической, гиперболической или параболической природой возможных траекторий, а также зависимостью от числа витков M вокруг центрального тела (рисунок 2.2) [9].

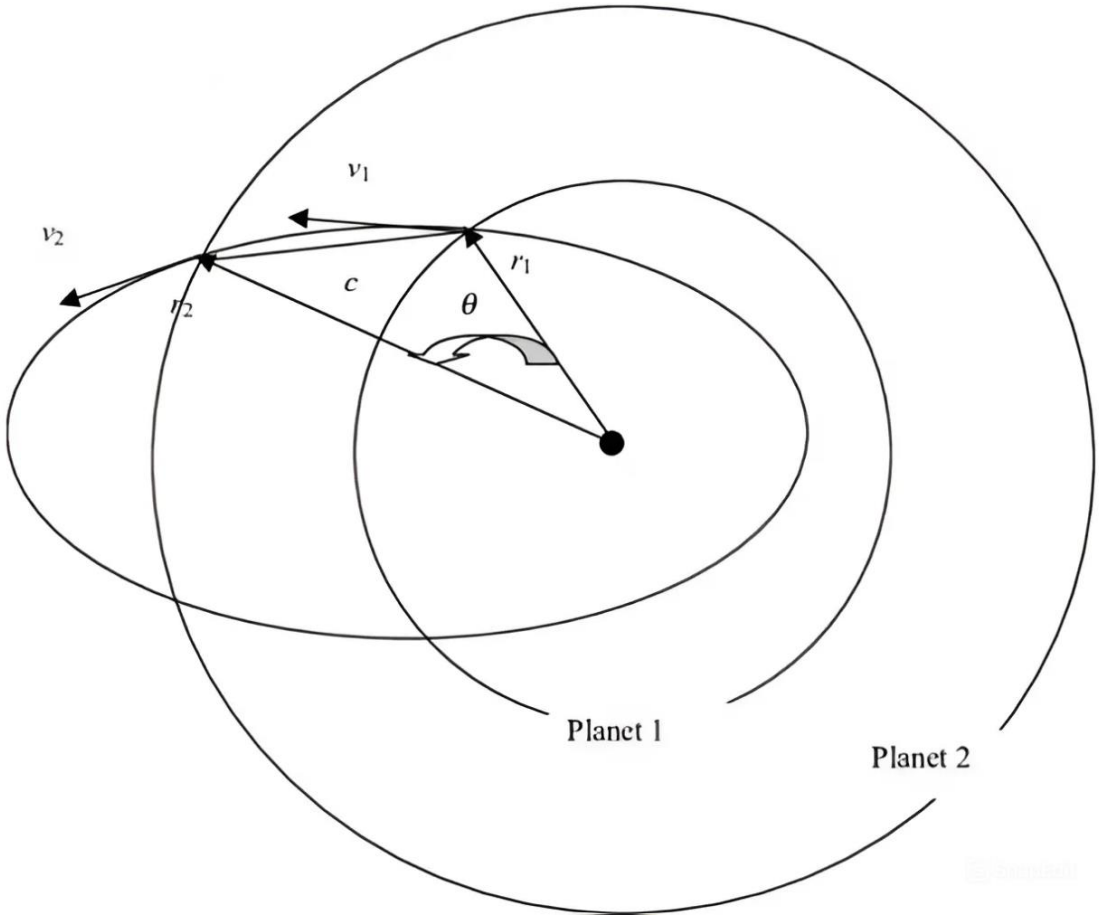


Рисунок 2.2 – Схема простого полета летательного аппарата с «планеты 1» на «планету 2» с применением задачи Ламберта

Исторически задача Ламберта возникла из работ Леонарда Эйлера и Иоганна Ламберта, изучавших свойства конических сечений в гравитационных полях. Ламберт установил, что время перелёта между двумя точками зависит только от суммы расстояний до центрального тела, длины хорды, соединяющей эти точки, и большой полуоси орбиты. Этот результат, известный как теорема Ламберта, лег в основу современных алгоритмов решения задачи [6].

В космическую эру необходимость в создании надежного алгоритма, способного работать в широком диапазоне условий, привела к повторному изучению проблемы Ламберта. Среди множества вкладов того периода выделяется работа Ланкастера и Бланшарда (1969), которая свела решение проблемы к итерациям, требующим вычисления лишь одной обратной тригонометрической или гиперболической функции. Позже Гудинг (1990) развил эти результаты и опубликовал метод, обеспечивающий высокую точность всего за три итерации для всех геометрий. Алгоритм Гудинга использует итерации Хэлли, дополненные продуманными эвристиками для задания начального приближения, а его подход к восстановлению векторов конечной скорости примечателен чисто алгебраической природой. Полученная процедура чрезвычайно эффективна, обладая низкой вычислительной стоимостью и высокой точностью. Ряд исследований тщательно протестировали подход Гудинга, подтвердив его превосходство над другими решателями Ламберта. Его метод признан наиболее точным и быстрым из существующих. Помимо алгоритма Гудинга, было предложено множество других решений, различающихся по крайней мере в одном из трёх ключевых аспектов: переменная итерации (непосредственно связанная с уравнением времени полёта), алгоритм итераций, начальное приближение и восстановление векторов конечной скорости (рисунок 2.2) [8].

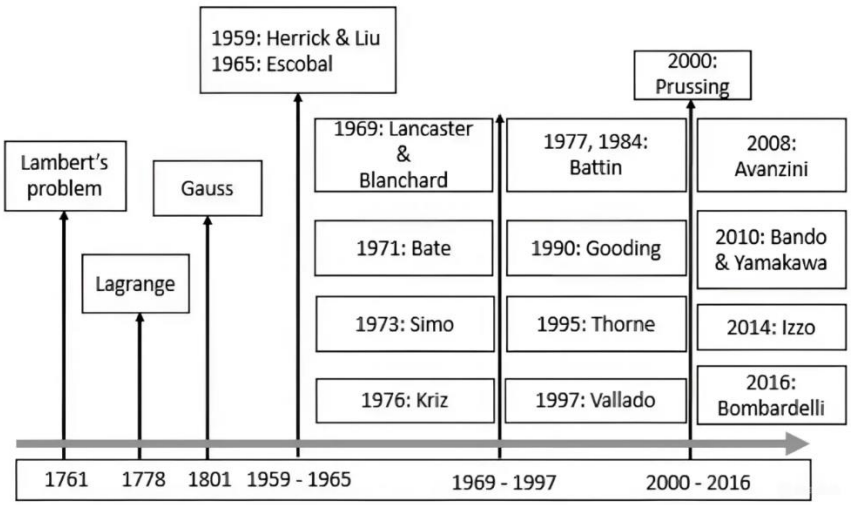


Рисунок 2.2 – Хронология исследований задачи Ламберта

2.5 Алгоритм решения задачи Ламберта

Теорема Ламберта утверждает, что время полета t необходимое для перемещения по кеплеровской орбите от r_1 до r_2, является функцией большой полуоси орбиты a, суммы радиусов $r_1 + r_2$ и хорды c треугольника, образованного векторами r_1 и r_2 [6].

Для начала вводятся эксцентрическая аномалия E и гиперболическая аномалия H через преобразования Сандмана:

$$rdE = ndt \text{ и } rdH = Ndt,$$

где $\mathrm{n} = \sqrt{\mu/a^3}$ — среднее движение (для эллиптических орбит), а $\mathrm{N} = \sqrt{-\mu/a^3}$ — его гиперболический аналог. Поскольку универсальные переменные не используются, уравнения приводятся отдельно для эллиптического $a > 0$ и гиперболического $a < 0$ случаев.

Также вводится редуцированная эксцентрическая аномалия $E_r \in [0,2\pi]$, так что при совершении $\widetilde{M}$ полных оборотов $E = E_r + 2\widetilde{M}\pi$. Для упрощения обозначений далее индекс r опускается, и E подразумевает редуцированную эксцентрическую аномалию.

Для эллиптической орбиты ($a > 0$):

$$\begin{cases} r = a(1 - e\cos E), \\ nt = E - e\sin E + 2\widetilde{M}\pi, \\ r\cos f = a(\cos E - e), \\ r\sin f = a\sqrt{1-e^2}\sin E, \end{cases} \tag{3.1}$$

где $\widetilde{M}$ — число полных оборотов, $E \in [0,2\pi]$ — редуцированная эксцентрическая аномалия, а f — истинная аномалия.

Первое уравнение связывает радиус орбиты r с эксцентрической аномалией E, второе представляет знаменитое уравнение Кеплера, связывающее эксцентрическую аномалию с временем полёта, а следующие два определяют связь между истинной аномалией f и E.

Аналогично для гиперболической орбиты ($a < 0$): H

$$\begin{cases} r = a(1 - e\,cosh\,H), \\ Nt = e\,sinh\,H - H, \\ r\,cos\,f = a(cosh\,H - e), \\ r\,sin\,f = -a\sqrt{e^2 - 1}sinh\,H. \end{cases} \tag{3.2}$$

Время полета между точками r_1 и r_2 выражается как:

$$\sqrt{\mu}(t_2 - t_1) = \begin{cases} a^{3/2}(E_2 - E_1 + ecosE_1 - ecosE_2 + 2M\pi) \\ -a^{3/2}\big(ecoshH_2 - ecosH_1 - (H_2 - H_1)\big) \end{cases} \tag{3.3}$$

где $M = \widetilde{\mathrm{M}}_2 - \widetilde{\mathrm{M}}_1$ — число полных оборотов за время перелета.

Вводятся вспомогательные углы:

$$\psi = \begin{cases} \frac{E_2 - E_1}{2} \\ \frac{H_2 - H_1}{2}, \end{cases} \qquad cos\,\varphi = \begin{cases} ecos\,\frac{E_2 + E_1}{2} \\ ecosh\,\frac{H_2 + H_1}{2} \end{cases} \tag{3.4}$$

где $\psi \in [0, \pi]$для обоих типов орбит, а $\varphi \in [0, \pi]$ (эллиптический случай) или $\varphi \geq 0$ (гиперболический случай). Уравнение времени полета принимает вид:

$$\sqrt{\mu}(t_2 - t_1) = \begin{cases} 2a^{3/2}(\psi - cos\varphi\,sin\psi + M\pi), \\ -2a^{3/2}(cosh\varphi\,sinh\psi - \psi). \end{cases} \tag{3.5}$$

Введённые величины φ и ψ зависят только от геометрии задачи и большой полуоси a. Это подтверждается вычислением $c^2 = (\mathrm{r}_2 \mathrm{cos} f_2 - r_1 \mathrm{cos} f_1)^2 - (r_2 \mathrm{sin} f_2 - r_1 \mathrm{sin} f_1)^2$ из (3.1) и (3.2), где:

$$r_1 + r_2 = \begin{cases} 2a(1 - cos\,\psi\,cos\,\varphi), \\ 2a(1 - cosh\,\psi\,cosh\,\varphi), \end{cases} \tag{3.6}$$

$$c = \begin{cases} 2a\,sin\,\psi\,sin\,\varphi, \\ -2a\,sinh\,\psi\,sinh\,\varphi. \end{cases} \tag{3.7}$$

Таким образом, время полета, заданное уравнением (3.5), зависит от a, c и $r_1 + r_2$. Для дальнейшего анализа вводятся углы:

$$\alpha = \varphi + \psi, \beta = \varphi - \psi \tag{3.8}$$

где для эллиптического случая $\alpha \in [0,2\pi], \beta \in [-\pi, \pi]$, а для гиперболического — $\alpha \geq 0, \beta \geq -\pi$. Эти границы важны для разрешения неоднозначностей [8]. Что позволяет переписать уравнение времени полета в компактной форме:

$$\sqrt{\mu}(t_2 - t_1) = \begin{cases} a^{3/2}\big((\alpha - \sin\alpha) - (\beta - \sin\beta) + 2M\pi\big), \\ -2a^{3/2}\big((\sinh\alpha - \alpha) - (\sinh\beta - \beta)\big). \end{cases} \tag{3.9}$$

вычисляя $a + b$ по уравнениям (3.6) и (3.7), можно легко найти:

$$\frac{s}{2a} = \begin{cases} \sin^2\frac{\alpha}{2}, \\ -\sinh^2\frac{\alpha}{2}, \end{cases} \quad \frac{s-c}{2a} = \begin{cases} \sin^2\frac{\beta}{2} \\ -\sinh^2\frac{\beta}{2} \end{cases} \tag{3.10-3.11}$$

где $s = (r_1 + r_2 + c)/2$ — полупериметр. Эти соотношения, впервые выведенные Лагранжем, использовались в доказательстве теоремы Ламберта [9].

Квадрант угла β определяется через разложение $\cos\left(\frac{\theta}{2}\right) = \cos\left(\frac{f_1 - f_2}{2}\right)$:

$$\sqrt{r_1 r_2}\cos\frac{\theta}{2} = \begin{cases} 2a\sin\frac{\alpha}{2}\sin\frac{\beta}{2}, \\ -2a\sinh\frac{\alpha}{2}\sinh\frac{\beta}{2}. \end{cases} \tag{3.12}$$

Поскольку $\sin\frac{\alpha}{2}, \sinh\frac{\alpha}{2} \geq 0$, знаки $\sin\frac{\beta}{2}, \sinh\frac{\beta}{2}$ совпадает со знаком $\cos\frac{\theta}{2}$. Таким образом, $\beta \in [-\pi, 0]$ при $\theta \geq \pi$ и $\beta > 0$ при $\theta \in [0, \pi]$. Неоднозначность угла α остается, так как две различные эллиптические орбиты с одинаковой a могут соединять r_1 и r_2, давая разные времена полета.

Далее следуем подходу Бланшара и Ланкастера, и выведем заново некоторые из соотношений. Рассмотрим параметр λ, определенный как:

$$s\lambda = \sqrt{r_1 r_2}\cos\frac{\theta}{2}$$

где $s = (r_1 + r_2 + c)/2$ — полупериметр [9]. Используя уравнение (3.12) и подставляя выражения из уравнений (3.10) и (3.11), легко показать, что:

$$\lambda^2 = \frac{s - c}{s}$$

Параметр $\lambda \in [-1,1]$ положителен при $\theta \in [0,\pi]$ и отрицателен при $\theta \in [0,2\pi]$. Значения λ^2, близкие к единице, соответствуют нулевой длине хорды — случаю, требующему особой обработки из-за вырожденной геометрии [9].

Из уравнений (3.10) и (3.11) также следует соотношение:

$$sin\frac{\alpha}{2} = \lambda sin\frac{\beta}{2} \tag{3.13}$$

Вводится безразмерное время полёта:

$$T = \frac{1}{2}\sqrt{\frac{\mu}{a_m^3}}(t_2 - t_1) = \sqrt{\frac{2\mu}{s^3}}(t_2 - t_1),$$

где $a_m = s/2$ — большая полуось эллипса с минимальной энергией. Преимущество использования λ и T в том, что T зависит только от λ и отношения a/a_m, что значительно упрощает классификацию возможных задач Ламберта.

Далее, вводим новые величины x и y:

$$x = \begin{cases} \cos\dfrac{\alpha}{2}, & \text{эллиптический случай} \\ \cosh\dfrac{\alpha}{2}, & \text{гиперболический случай} \end{cases}, \; y = \begin{cases} \cos\left(\dfrac{\beta}{2}\right), & \text{эллиптический случай} \\ \cosh\left(\dfrac{\beta}{2}\right), & \text{гиперболический случай} \end{cases}, \tag{3.14}$$

что подразумевает:

$$\begin{cases} \sqrt{1-x^2} = \sin\dfrac{\alpha}{2} \\ \sqrt{x^2-1} = \sinh\dfrac{\alpha}{2} \end{cases}, \qquad \begin{cases} \lambda\sqrt{1-x^2} = \sin\dfrac{\beta}{2} \\ \lambda\sqrt{x^2-1} = \sinh\dfrac{\beta}{2} \end{cases} \tag{3.15}$$

Используя эти соотношения, можно напрямую связать вспомогательные углы с x

$$\begin{cases} \cos\varphi = xy - \lambda(1-x^2), \\ \cosh\varphi = xy + \lambda(x^2-1). \end{cases} \qquad \begin{cases} \sin\varphi = (y + x\lambda)\sqrt{1-x^2}, \\ \sinh\varphi = (y + x\lambda)\sqrt{x^2-1}. \end{cases} \tag{3.16}$$

$$\begin{cases} \cos\psi = xy + \lambda(1-x^2), \\ \cosh\psi = xy - \lambda(x^2-1). \end{cases} \qquad \begin{cases} \sin\psi = (y - x\lambda)\sqrt{1-x^2}, \\ \sinh\psi = (y - x\lambda)\sqrt{x^2-1}. \end{cases} \tag{3.17}$$

Таким образом, мы можем преобразовать уравнение времени полёта к

универсальной форме, применимой для всех случаев (параболического, гиперболического, эллиптического):

$$T = \frac{1}{1-x^2}\left(\frac{\psi+M\pi}{\sqrt{|1-x^2|}} - x + \lambda y\right), \tag{3.18}$$

где для гиперболического и параболического движения $M = 0$, так как неограниченное движение исключает полные обороты. Вспомогательный угол ψ вычисляется через соответствующие обратные функции из уравнений (3.17), сводя расчёт времени полёта к одному обратному тригонометрическому или гиперболическому вычислению [8].

Учитывая ограничения на α из определения x следует, что $x \in [-1, \infty)$. Значения $x > 1$ соответствуют гиперболическому движению, $x = 1$ — параболическому, а $x < 1$ — эллиптическому. Поскольку

$$1 - x^2 = \sin^2\frac{\alpha}{2} = \frac{s}{2a} = \frac{a_m}{a},$$

Значение $x = 0$ соответствует эллипсу с минимальной энергией. Задачи Ламберта с одинаковыми λ (т.е. одинаковыми c/s) приводят к одинаковым x, что позволяет назвать x инвариантным параметром Ламберта [8].

При $x = 0$ уравнение (3.18) даёт:

$$T(x = 0) = T_M = \arccos\lambda + \lambda\sqrt{1 - x^2 + M\pi} = T_0 + M\pi, \tag{3.19}$$

где T_{00} — значение T для случая одного оборота ($M = 0$).

Вблизи $x \approx 1$ (параболический случай) возникает потеря точности из-за численной неустойчивости. Для устранения этой проблемы используется разложение в ряд на основе результата Баттина:

$$\begin{cases} \eta = y - \lambda x, \\ S_1 = \frac{1}{2}(1 - \lambda - x\eta), \\ Q = \frac{4}{3}\ {}_1F_2\left(3,1,\frac{5}{2},S_1\right), \\ 2T = \eta^3 Q + 4\lambda\eta. \end{cases} \tag{3.20}$$

где ${}_1F_2(a, b, c, d)$ — это гауссова или обыкновенная гипергеометрическая функция [8]. Это можно оценить путем прямого вычисления ассоциированного гипергеометрического ряда. Замечая, что $S_1 \to 0$ при $x \to 1$, количество сохраняемых членов в ряде всегда невелико, когда ряд используется в окрестности $x = 1$. Исходя из Баттина, мы изучаем параболический случай, подставляя $x = 1$, $y = 1$ в уравнение (3.20) и таким образом получаем

выражение:

$$T(x=1)=T_1=\frac{2}{3}(1-\lambda^2) \tag{3.21}$$

что связывает геометрию треугольника, образованного двумя наблюдениями объекта на параболической орбите, с безразмерным временем между ними [6]. Производные времени полёта вычисляются как:

$$(1-x^2)\frac{dT}{dx}=3Tx-2+2\lambda^3\frac{x}{y},$$

$$(1-x^2)\frac{d^2T}{dx^2}=3T+5x\frac{dT}{dx}+2(1-\lambda^2)\frac{\lambda^3}{y^3},$$

$$(1-x^2)\frac{d^3T}{dx^3}=7x\frac{d^2T}{dx^2}+8\frac{dT}{dx}-6(1-\lambda^2)\frac{\lambda^5 x}{y^5}. \tag{3.22}$$

что справедливо для всех случаев, кроме $\lambda^2=1, x=0$ и $x=1, \forall\lambda$. Применяя правило Лопиталя к первому уравнению, получаем производную времени полёта для параболы $(M=0)$:

$$\frac{dT}{dx}\Big|_{x=1}=\frac{2}{5}\left(\lambda^5-1\right). \tag{3.23}$$

Это выражение, вместе с уравнением (3.21), используется для получения хороших начальных приближений для T. Путем прямой подстановки можно также легко показать, что

$$\left.\frac{dT}{dx}\right|_{x=0}=-2.$$

Огромное преимущество уравнения времени полета в уравнении (3.18), как показано Ланкастером и Бланшаром (1969), заключается в том, что оно имеет низкую вычислительную стоимость для вычисления T и является довольно простым. Используется только тригонометрическое (или гиперболическое) преобразование, два квадратных корня и несколько умножений и делений можно легко вычислить эти численные значения. Другие подходы, основанные на геометрических соображениях или универсальных формах, имеют ограниченные возможности для соответствия такой простой записи [9].

Теперь введем новую переменную:

$$\xi = \begin{cases} \log(1+x), & M = 0 \\ \log\left(\dfrac{1+x}{1-x}\right), & M > 0 \end{cases}, \quad \tau = \log(T), \tag{3.24}$$

где M— число полных оборотов. Область определения кривых времени полёта расширяется до $[-\infty, \infty]$. Для случая $M = 0$ область значений также расширяется аналогичным образом. Введённый параметр ξ является инвариантом Ламберта согласно определению Гудинга (Gooding 1990), так как он представляет собой преобразование инвариантной переменной x [8].

Изучим новые дифференциальные свойства. Производные для $M = 0$:

$$d\xi = \begin{cases} \dfrac{1}{1+x^2}\,dx, & M = 0 \\ \dfrac{2}{1-x^2}\,dx, & M > 0 \end{cases}, d\tau = \frac{1}{T}\,dT \tag{3.25}$$

Подставляя эти соотношения в уравнение (3.22), после некоторых манипуляций мы можем получить следующие гибридные выражения для производных при $M = 0$:

$$\begin{aligned} \frac{d\tau}{d\xi} &= \frac{1+x}{T}\frac{dT}{dx} \\ \frac{d^2\tau}{d\xi^2} &= \frac{(1+x)^2}{T}\frac{d^2T}{dx^2} + \frac{d\tau}{d\xi} - \left(\frac{d\tau}{d\xi}\right)^2 \\ \frac{d^3\tau}{d\xi^3} &= \frac{(1+x)^3}{T}\frac{d^3T}{dx^3} + \left(\frac{d^2\tau}{d\xi^2} - \frac{d\tau}{d\xi} + \left(\frac{d\tau}{d\xi}\right)^2\right)\left(2 - \frac{d\tau}{d\xi}\right) + \frac{d^2\tau}{d\xi^2} - 2\frac{d\tau}{d\xi}\frac{d^2\tau}{d\xi^2} \end{aligned} \tag{3.26}$$

Эти выражения вычисляются последовательно на основе уравнений (3.22). Для случая $M = 0$ выполняются предельные соотношения:

$$\begin{aligned} \lim_{\xi\to\infty} \tau &= -\xi + \log(1 - \lambda\,|\,\lambda\,|) \\ \lim_{\xi\to-\infty} \tau &= -\frac{3}{2}\xi + \log\left(\frac{\pi}{4}\sqrt{2}\right) \end{aligned} \tag{3.27}$$

что описывает асимптотическое поведение времени полёта. Асимптоты имеют угловые коэффициенты -1 и $-\frac{3}{2}$ [8].

Для многократных оборотов $(M > 0)$ производные принимают вид:

$$\frac{d\tau}{d\xi} = \frac{1-x^2}{2T}\frac{dT}{dx}$$

$$\frac{d^2\tau}{d\xi^2} = \frac{(1-x^2)^2}{4T}\frac{d^2T}{dx^2} - x\frac{d\tau}{d\xi} - \left(\frac{d\tau}{d\xi}\right)^2$$

$$\frac{d^3\tau}{d\xi^3} = \frac{(1-x^2)^3}{8T}\frac{d^3T}{dx^3} - \left(\frac{d^2\tau}{d\xi^2} + x\frac{d\tau}{d\xi} + \left(\frac{d\tau}{d\xi}\right)^2\right)\left(2x + \frac{d\tau}{d\xi}\right)$$

$$-\frac{1-x^2}{2}\frac{d\tau}{d\xi} - x\frac{d^2\tau}{d\xi^2} - 2\frac{d\tau}{d\xi}\frac{d^2\tau}{d\xi^2} \tag{3.28}$$

опять же, это можно вычислить каскадно, и можно получить следующее асимптотическое поведение:

$$\lim_{\xi\to-\infty}\tau = \log\left(\frac{\pi + M\pi}{8}\right) - \frac{3}{2}\xi$$

$$\lim_{\xi\to\infty}\tau = \log\left(\frac{M\pi}{8}\right) + \frac{3}{2}\xi \quad , \tag{3.29}$$

что указывает на симметричные асимптоты с угловыми коэффициентами $\pm\frac{3}{2}$ [8].

ГЛАВА 3.
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МАНЕВРА

3.1 Основные компоненты модели

Программная реализация модели включает несколько ключевых компонентов, каждый из которых выполняет свою роль в симуляции межпланетного перелета:

- Данные о планетах: для получения актуальных данных о планетах используются данные, предоставляемые сервисом NASA JPL Horizons. Эти данные включают координаты и скорости планет в различные моменты времени, что позволяет моделировать их движение в пространстве.
- Моделирование гравитационных полей: для точного расчета траекторий космических аппаратов необходимо учитывать гравитационные поля планет. В программе используется выводы из второй главы для

вычисления гравитационных сил, действующих на космический аппарат со стороны Солнца и планет.

- Поиск оптимальной траектории летательного аппарата: программа перебирает всевозможные начальные параметры, такие как начальная скорость аппарата и дата начала полета. Проверяет возможность совершения одного или нескольких маневров вокруг планет.

- Моделирование траектории летательного аппарата: модель включает его начальные параметры, такие как масса аппарата, начальные координаты и начальный вектор скорости. Программа рассчитывает изменение положения и скорости аппарата под воздействием гравитационных сил планет и Солнца.

- Визуализация данных: для наглядного представления результатов моделирования используется библиотека «matplotlib» и «mplot3d» для двумерного и трехмерного случая соответственно. Визуализация включает графики зависимости скорости аппарата от времени, а также анимацию, показывающую движение космического аппарата и планет по их траекториям.

Выбор языка программирования Python для реализации модели обусловлен его простотой и читаемостью, что облегчает разработку и отладку кода. Богатая экосистема библиотек, таких как «numpy», «scipy» и «matplotlib», предоставляет мощные инструменты для научных вычислений и визуализации данных, что делает его идеальным для данного проекта.

3.2 Маневры ускорения и торможения

Процесс моделирования начинается с задания начальных параметров миссии: даты старта, начальных координат и скорости космического аппарата. Программа последовательно вычисляет движение аппарата, интегрируя уравнения движения с учетом гравитационных воздействий Солнца и планет. На каждом временном шаге обновляются координаты и векторы скорости, одновременно собирая данные для последующего анализа и визуализации. Этот подход позволяет исследовать различные сценарии перелетов путем изменения начальных условий и параметров траектории. Для верификации программы были реализованы два базовых сценария:

Первый сценарий демонстрирует, как при прохождении за Юпитером (рисунок 3.1в) аппарат получает значительное ускорение за счет гравитационного маневра, что видно по характерному росту скорости на рисунке 3.1а. Этот эффект обусловлен передачей орбитальной энергии планеты космическому аппарату, когда траектория пролета образует гиперболическую

кривую в гравитационном поле Юпитера. В рамках данного сценария космический аппарат запускается с Земли в начале сентября 1977 года и попадает в гравитационное поле Юпитера в начале апреля 1979 года. Маневр показал, как космический аппарат может ускоряться за счет гравитационного поля планет.

Во втором сценарии исследовано влияние изменения даты запуска космического аппарата на его траекторию и возможность выполнения гравитационного маневра для торможения. Для этого сценария дата запуска была сдвинута на один день вперед, при этом все остальные параметры остались неизменными. В результате траектория аппарата смещается таким образом, что он проходит перед Юпитером (рисунок 3.1г), что отличается от первого сценария, где аппарат проходил за планетой. Прохождение впереди планеты позволяет аппарату снизить свою скорость за счет гравитационного воздействия Юпитера, что является противоположным эффектом по сравнению с маневром ускорения. Физически это объясняется тем, что при переднем пролете относительно направления орбитального движения планеты аппарат теряет кинетическую энергию, передавая ее Юпитеру.

Оба сценария, результаты которых визуализированы на рисунке 3.1, подтверждают корректность математической модели и программной реализации. Различия в динамике скорости (рисунки 3.1а-б) и геометрии траекторий (рисунки 3.1в-г) демонстрируют ключевой принцип гравитационных маневров: направление облета планеты относительно ее орбитального движения определяет, получит аппарат ускорение или замедление. Полученные результаты полностью соответствуют теоретическим предсказаниям, описанным в разделе 1.2, и подтверждают возможность использования программы для проектирования сложных межпланетных миссий.

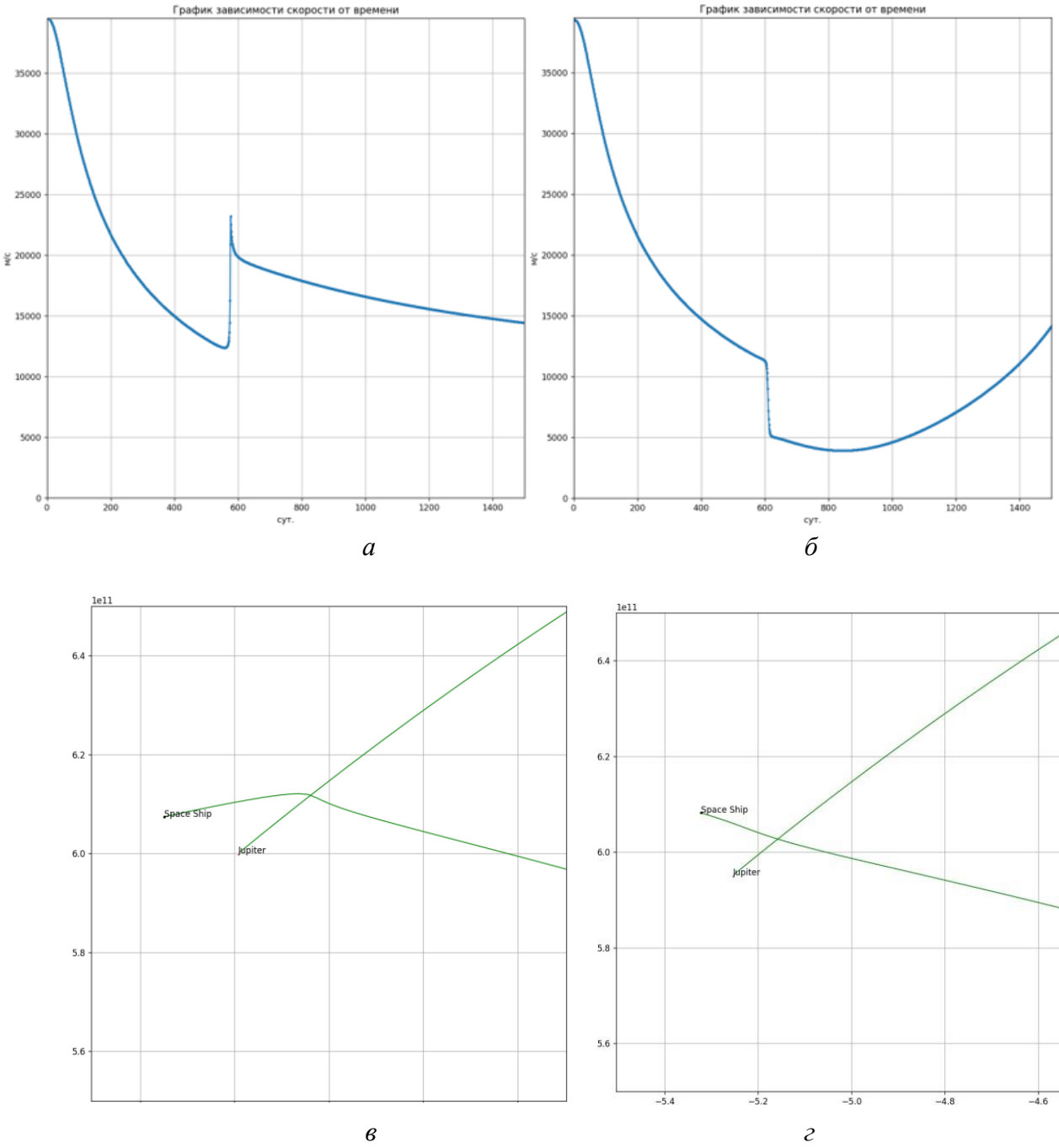


Рисунок 3.1 – Изменение скорости и траекторий при гравитационных маневрах. а – скорость ЛА для маневра ускорения; б – скорость ЛА для маневра торможения; в – траектории для маневра ускорения; г – траектории для маневра торможения

3.3 Метод Иззо решения задачи Ламберта

Ключевым этапом расчёта оптимальной траектории является решение задачи Ламберта, как основы для планирования переходов между орбитами планет. Её решение позволяет находить возможные траектории между двумя

точками пространства за заданное время и играет ключевую роль в построении межпланетных маршрутов.

Решатель Ламберта определяется как процедура, возвращающая для гравитационного поля с параметром μ все возможные векторы скорости v_1 и v_2, соответствующие кеплеровским орбитам, соединяющим r_1и r_2 за время перелёта T*. Основными компонентами такого алгоритма являются: выбор переменной для итераций; итерационный метод; начальное приближение для итерационного метода; методология восстановления векторов скорости v_1 и v_2 из значения, полученного в результате итераций [8].

Рассмотрим наиболее современный и быстрый алгоритм решения данной задачи, представленный в 2014 году Дарио Иззо. В данном алгоритме: итерации проводятся по переменной Ланкастера-Бланшара x; используется схема итераций Хаусхолдера; начальное приближение строится на основе аналитического анализа формы кривых в плоскости $\tau - \xi$; векторы скорости восстанавливаются по методологии Гудинга (1990) [8].

Одним из ключевых отличий предложенного решателя от предыдущих работ является использование итерационной схемы Хаусхолдера для нахождения корня уравнения времени полёта $T(x) - T* = 0$. Итерации Хаусхолдера редко применяются в подобных задачах, так как вычисление производных высокого порядка требует дополнительных вычислительных ресурсов. Однако в данном случае, благодаря использованию уравнений (3.22) для расчёта производных, метод демонстрирует значительную эффективность.

Итерации реализованы в следующей форме:

$$x_{n+1} = x_n - f(x_n)\frac{f'^2(x_n) - f(x_n)f''(x_n)/2}{f'(x_n)(f'^2(x_n) - f(x_n)f''(x_n)) + f'''(x_n)f^2(x_n)/6}$$

где $f(x) = T(x) - T*$, а f', f'', f''' обозначают первую, вторую и третью производные $f(x)$.

Начальное приближение для переменной x генерируется на основе введённых переменных $\tau - \xi$ и значений T_0 и T_1, вычисленных из уравнений (3.19) и (3.21). Приближение получается инвертированием линейной аппроксимации кривых времени полёта: $\tau = c\xi + d$, где коэффициенты c и d выбираются в зависимости от значений τ и M [8].

- Для случая одного оборота ($M = 0$):

При больших значениях τ коэффициент $c = -1.0$; при малых значениях τ коэффициент $c = -3/2$, что соответствует асимптотическому поведению,

описанному в уравнении (3.27). Используется кусочно-линейная аппроксимация:

$$\tau = \begin{cases} -\frac{3}{2}\xi + \tau_0, & T \geq T_0, \\ -\xi + \log 2 + \tau_1, & T < T_1, \\ \tau_0 + \frac{\xi}{\log 2}(\tau_1 - \tau_0), & T_1 < T < T_0, \end{cases}$$

где $\tau_1 = logT_1, \tau_0 = logT_0$. Линии проходят через точки (x_0, T_0) и (x_1, T_1), сохраняя асимптотики. Инвертирование даёт начальное приближение для ξ_0:

$$\xi_0 = \begin{cases} \frac{2}{3}(\tau_0 - \tau), & T \geq T_0, \\ \log 2 + \tau_1 - \tau, & T < T_1, \\ \frac{\tau - \tau_0}{\tau_1 - \tau_0} \log 2, & T_1 < T < T_0. \end{cases}$$

Преобразуя обратно в плоскость x $-$ T, получаем выражения для x_0:

$$x_0 = \begin{cases} \left(\frac{T_0}{T}\right)^{\frac{2}{3}} - 1, & T \geq T_0, \\ 2\frac{T_1}{T} - 1, & T < T_1, \\ \left(\frac{T_0}{T}\right)^{\log_2\left(\frac{T_1}{T_0}\right)} - 1, & T_1 < T < T_0. \end{cases}$$

Для улучшения точности при $T < T_1$ используется уравнение (3.23):

$$x_0 = \begin{cases} \left(\frac{T_0}{T}\right)^{\frac{2}{3}} - 1, & T \geq T_0, \\ \frac{5}{2}\frac{T_1(T-T)}{T(1-\lambda^5)} + 1, & T < T_1, \\ \left(\frac{T_0}{T}\right)\log_2\left(\frac{T}{T_0}\right) - 1, & T_1 < T < T_0. \end{cases}$$

где учтены производные и значения в $x = 1$ и x $= \infty$ [8].

- Для случая множества оборотов ($M > 0$):

Существует два возможных значения x, требующих двух начальных приближений. Они строятся инвертированием асимптотик из уравнения (3.29):

$$\xi_{0l} = \frac{2}{3}\left(\log\frac{\pi + M\pi}{8} - \tau\right),$$

$$\xi_{0r} = \frac{2}{3}\left(\tau - \log\frac{M\pi}{8}\right).$$

Преобразование обратно в плоскость $x - T$ даёт:

$$x_{0l} = \frac{\left(\frac{(M+1)\pi}{8T}\right)^{\frac{2}{3}} - 1}{\left(\frac{(M+1)\pi}{8T}\right)^{\frac{2}{3}} + 1},$$

$$x_{0r} = \frac{\left(\frac{8T}{M\pi}\right)^{\frac{2}{3}} - 1}{\left(\frac{8T}{M\pi}\right)^{\frac{2}{3}} + 1}.$$

Приведенные выше выражения хорошо аппроксимируют кривые времени полета при $|x| \to 1$. Не требуют вычисления T_{min} или x_{min} (минимум кривой времени полета и ее экстремальное значение), что исключает дополнительные итерации [8].

Для тестирования производительности нового алгоритма сначала оценивается его точность. Рассматривается случай с заданным M: случайным образом выбираются $\lambda \in [-0.999,0.999]$ и $\mathrm{x}_{true} = [-0.99,3]$ (или $x_{true} = [-0.999,0,999]$ для $M > 0$), после чего вычисляется соответствующее время полёта T. Затем с использованием итераций Хаусхолдера и начального приближения определяется значение x. Итерации останавливаются, когда разность между значениями x на двух последовательных шагах становится меньше 10^{-5} (или 10^{-8} для $M > 0$). Для каждого теста фиксируются количество итераций и ошибка $\epsilon = |x_{true} - x|$. В подавляющем большинстве случаев ошибка $\epsilon < 10^{-13}$. Среднее количество итераций для $M = 0$ составляет 2.1, для $M > 0$ — 3.3 [8].

3.4 Анализ результатов на примере траектории «Новые горизонты»

Рассмотрим трехмерное моделирование миссии «Новые горизонты» с использованием разработанной программы. В ходе исследования был проанализирован период запуска с 1 декабря 2005 года по 1 марта 2006 года. В результате был найден оптимальный маршрут, который состоялся 19 января 2006 года. Аппарат получил рекордную стартовую скорость, однако с удалением от Солнца она стала уступать скоростям «Вояджера-1» и «Вояджера-2». Причиной этого является то, что «Вояджеры» использовали несколько гравитационных маневров, тогда как «Новые горизонты» — только один. Это наглядно демонстрирует принципиальную разницу в подходах к

проектированию межпланетных миссий.

В конце февраля 2007 года летательный аппарат успешно выполнил гравитационный манёвр возле Юпитера, увеличив скорость на 3,85 км/с и изменил наклон орбиты на 2,5° к плоскости эклиптики. Это было необходимо для точного выхода на орбиту Плутона, которая характеризуется значительным наклоном. За манёвром последовал восьмилетний перелёт, кульминацией которого стало максимальное сближение с Плутоном в начале июля 2015 года. Программа точно предсказала скорость сближения с Плутоном (14,89 км/с), а также ежегодное удаление аппарата от Солнца после пролёта — около 3 астрономических единиц (рисунок 3.2).

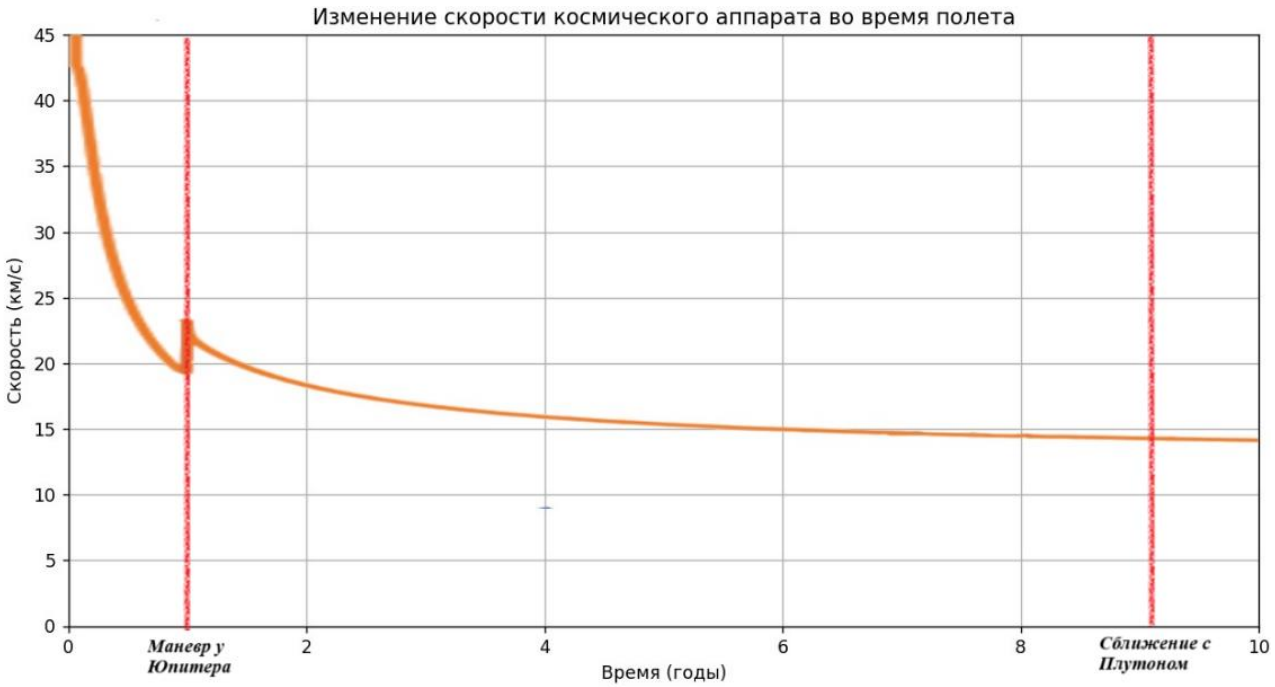


Рисунок 3.2 – График зависимости скорости от времени аппарата «Новые горизонты»

Разработанная модель продемонстрировала высокую точность расчётов на примере современной межпланетной миссии, что позволяет рассматривать её как эффективный инструмент для моделирования высокоскоростных космических перелётов. Сопоставление результатов симуляции с реальными данными подтвердило надёжность и достоверность методики (рисунок 3.3).

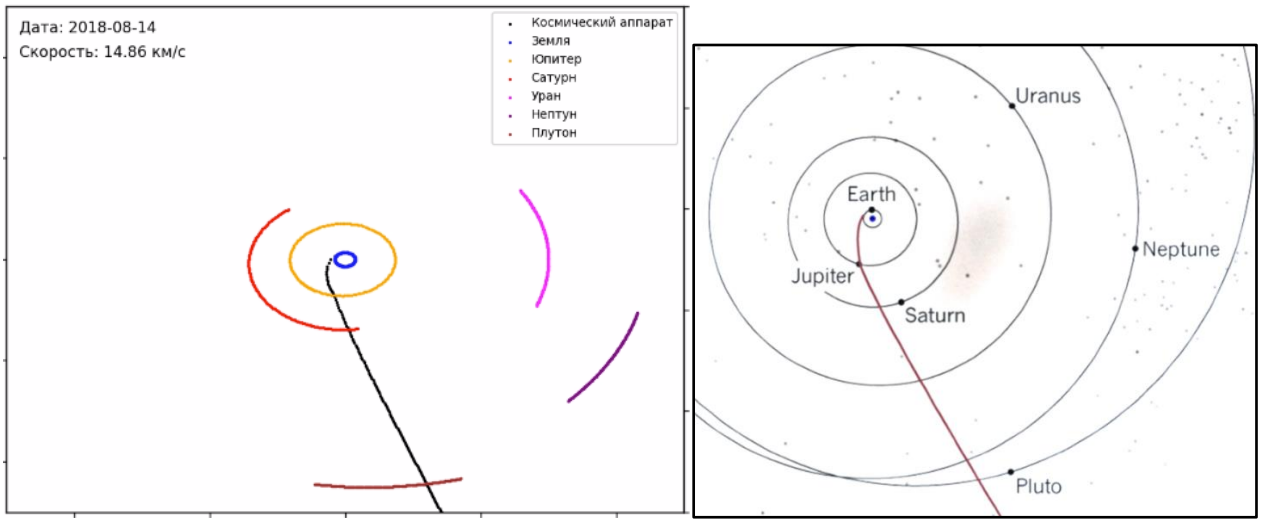


Рисунок 3.3 – Построенная программой и полученная из источника [12] траектория полета аппарата «Новые горизонты»

3.5 Анализ результатов на примере «Вояджер 2»

Моделирование в трехмерном пространстве миссии "Вояджер-2" стало ключевым этапом проверки работоспособности и точности разработанной программы. Воспроизведение этого уникального маршрута позволило комплексно протестировать алгоритмы расчета сложных гравитационных маневров.

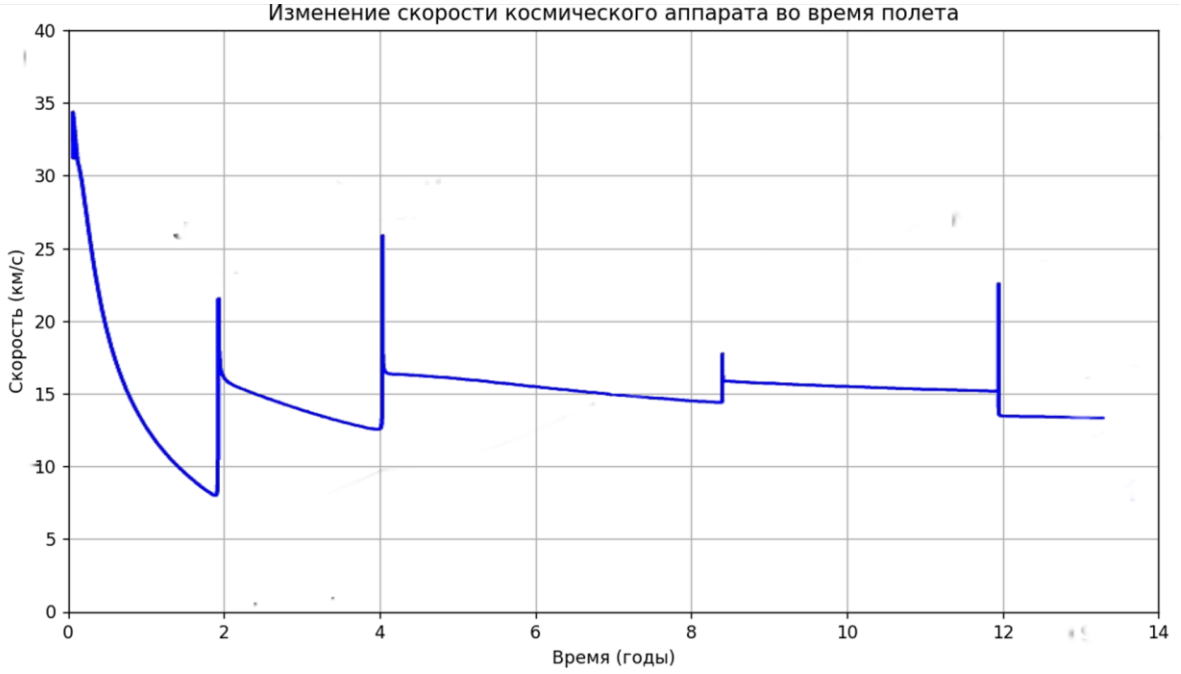


Рисунок 3.4 – График зависимости скорости от времени аппарата «Вояджер-2»

Важным этапом исследования стал анализ чувствительности миссии к параметрам запуска. Программа выяснила, что «окно запуска» для успешного выполнения всех запланированных маневров было чрезвычайно узким, так как смещение старта всего на 10 дней делало невозможным гравитационный маневр у Урана из-за изменения взаимного положения планет.

Был найден один из наиболее оптимальных маршрутов, который начал свой путь 21 августа 1977 года. После почти двухлетнего полета в середине июля 1979 года аппарат выполнил гравитационный маневр возле Юпитера, увеличив свою скорость более чем на 10 км/с (рисунок 3.4). Дальше аппарат продолжил свое путешествие длиной в 2 года и совершил маневр возле Сатурна. После этого начался самый длительный и сложный этап миссии — пятилетний перелет к Урану. Сблизился с ним «Вояджер» в конце января 1986 года. Последним маневром этого космического путешествия стал маневр возле Нептуна в конце августа 1989 года. После этих маневров летательный аппарат продолжил свой вечный полет в межзвездное пространство. Как видно на Рисунке 3.5, программа с высокой точностью воспроизвела реальную траекторию полета аппарата.

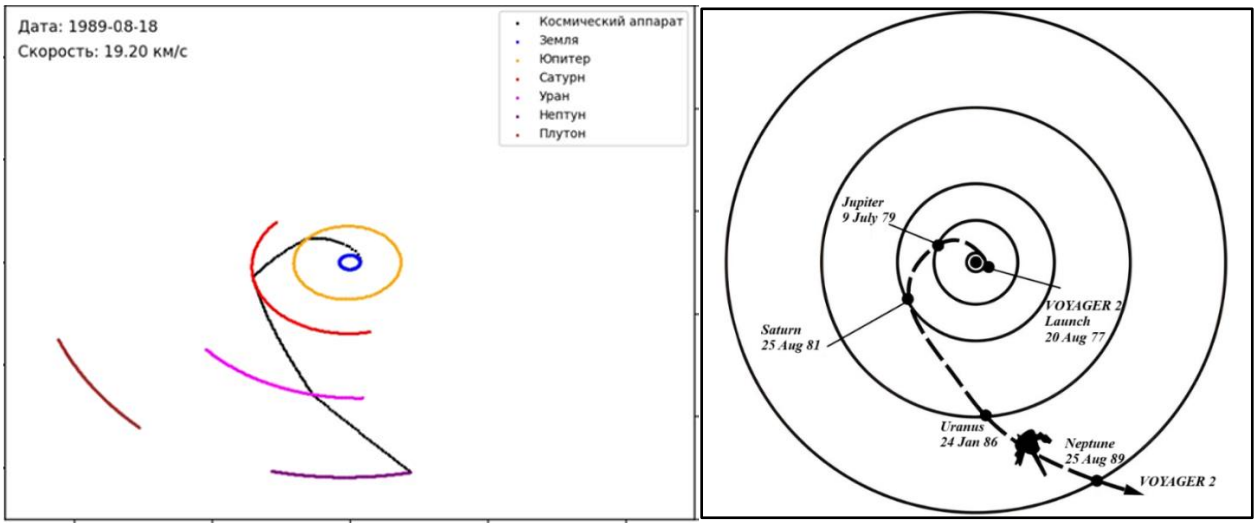


Рисунок 3.5 – Построенная программой и полученная из источника [12\] траектория полета аппарата «Вояджер 2»

Полученные результаты доказывают, что разработанная программа не только точно воспроизводит траекторию полета, но и корректно рассчитывает ключевые параметры миссии. Точность расчетов позволяет использовать данный инструмент для проектирования будущих межпланетных миссий с использованием гравитационных маневров.

3.6 Моделирование будущего аналога миссии «Вояджер 2»

Проведя анализ на длительном временном промежутке, программа выявила удивительный факт того, что следующая возможность повторения «Великого пути» с последовательными гравитационными маневрами у Юпитера, Сатурна, Урана и Нептуна представится лишь в 2156 году. Это подчеркивает уникальность исходной миссии «Вояджер-2» и редчайшее положение планет, которое она использовала (рисунок 3.6).

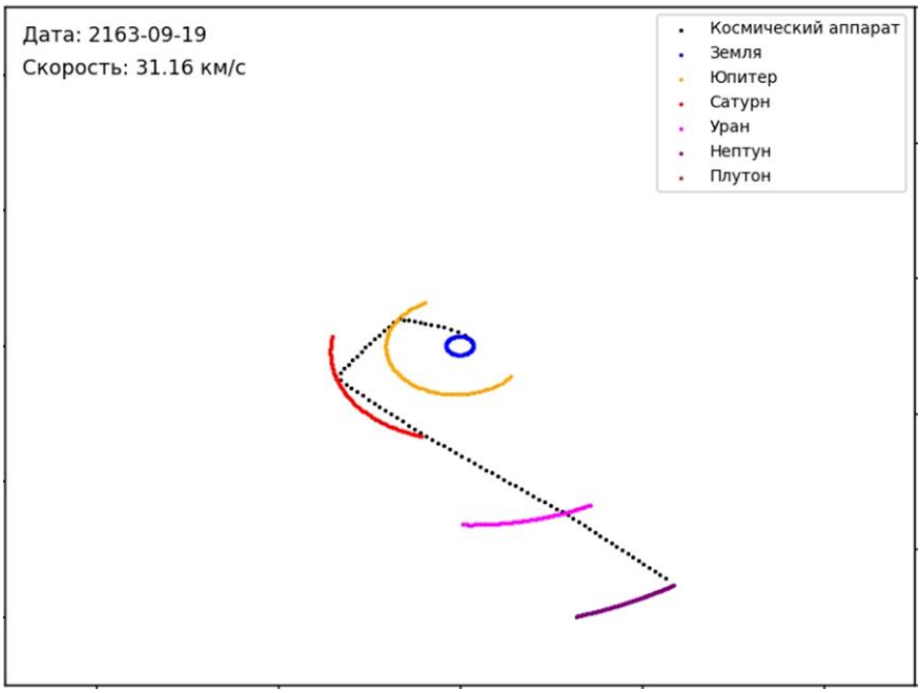


Рисунок 3.6 – Построенная траектория будущего аналога миссии «Вояджер-2»

Смоделированный маршрут демонстрирует существенное сокращение времени перелета между планетами, который будет возможен лишь благодаря будущим технологиям. Начнет свое путешествие аппарат в конце октября 2156 года. Если оригинальный «Вояджер-2» достиг Юпитера почти за два года, то будущий аппарат преодолеет этот участок всего за один год, прибыв к «газовому гиганту» уже в начале ноября 2157 года. Еще более впечатляющая разница наблюдается на участке до Сатурна, где вместо двух лет в будущем потребуется лишь 1 год и 1 месяц. Приблизится аппарат к планете в начале декабря 2159 года.

Не обошлось без значительного сокращения и на самом длинном участке

маршрута. Оригинальной миссии потребовалось 4,5 года для достижения Урана после Сатурна, тогда как будущий аппарат преодолеет это расстояние за два года в конце февраля 2162 года. Финальный рывок к Нептуну, который у «Вояджера» занял 3,5 года, в будущей миссии будет выполнен за рекордные 1,5 года и пролетит мимо «ледяного гиганта» в конце сентября 2163 года.

Общее время выполнения всей миссии сокращается с 12 до 7 лет. Это стало возможным благодаря комбинации факторов: различиям в положении планет, более мощным стартовым ускорителям и оптимизированным траекториям полета. Однако стоит отметить, что такие высокие скорости предъявляют исключительные требования к конструкции аппарата и его теплозащите при прохождении близко к планетам-гигантам.

ЗАКЛЮЧЕНИЕ

Проведённое исследование подтверждает роль гравитационных маневров как высокоэффективного инструмента для межпланетных перелетов. В рамках работы была успешно разработана и реализована комплексная математическая модель, интегрирующая расчет гиперболических траекторий вблизи планет с решением задачи Ламберта для глобального планирования оптимальных траекторий между планетами. Программная реализация модели на Python с использованием современных алгоритмов обеспечила высокую точность расчетов.

Верификация модели на примере исторических миссий «Вояджер-2» и «Новые горизонты» продемонстрировала её высокую надежность. Программа точно воспроизвела ключевые параметры полета, включая изменения скорости и направления после маневров. Анализ сценариев ускорения и торможения подтвердил теоретические принципы, лежащие в основе гравитационных маневров, и корректность их программной реализации.

Разработанный программный инструмент предоставляет мощные возможности для моделирования, визуализации и оптимизации сложных межпланетных миссий. Его применение позволяет не только анализировать исторические траектории с высокой точностью, но и проектировать будущие перелеты, как это было продемонстрировано на примере прогнозируемого аналога миссии «Вояджер-2».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гравитационный маневр [Электронный ресурс] // Википедия. – URL: https://ru.wikipedia.org/wiki/Гравитационный_маневр (дата обращения: 30.12.2024).

2. Левантовский, В. И. Механика космического полета в элементарном изложении / В. И. Левантовский. – Москва : Наука, 1980. – 512 с.

3. Платонов, И. В. Разработка двигательной установки на базе двигателей малой тяги и схемы полёта космических аппаратов к центру солнечной системы / И. В. Платонов, А. В. Симонов // Сибирский журнал науки и технологий. – 2018. – С. 45–52.

4. Синицын, А. А. Гравитационные маневры при осуществлении марсианской пилотируемой экспедиции с электроракетной двигательной установкой / А. А. Синицын // Инженерный журнал: наука и инновации. – 2019. – Вып. 8. – С. 1–15.

5. Суханов, А. А. Астродинамика / А. А. Суханов. — Москва : ИКИ РАН, 2010. — 204 с. — (Механика, управление, информатика).

6. Bombardelli, C. Approximate analytical solution of the multiple revolution Lambert’s targeting problem / C. Bombardelli, J. Roa, J. Gonzalo // Advances in the Astronautical Sciences. – 2016. – Vol. 158. – P. 1–14.

7. Gonzalez, A. Astrodynamics and Space Missions [Электронный ресурс] / A. Gonzalez // Space Engineering Podcast. – URL: https://space-engineering-podcast.simplecast.com/ (дата обращения: 11.01.2025).

8. Izzo, D. Revisiting Lambert's Problem / D. Izzo // ACT Technical Report. – 2014. – P. 1–15. – (ESA Technical Report).

9. Lancaster, E. R. A unified form of Lambert’s theorem / E. R. Lancaster, R. C. Blanchard // NASA technical note. – 1969. – P. 1–20. – (NASA TN D-5368).

10. NASA - Eyes On The Solar System [Электронный ресурс]. – URL: https://eyes.nasa.gov/apps/solar-system/ (дата обращения: 21.01.2025).

11. NASA - Planetary Fact Sheet [Электронный ресурс]. – URL: https://nssdc.gsfc.nasa.gov/planetary/factsheet/ (дата обращения: 14.02.2025).

12. NASA - Science Missions [Электронный ресурс]. – URL: https://science.nasa.gov/science-missions/ (дата обращения: 18.01.2025).

ПРИЛОЖЕНИЕ А

```
from planets import Planet
from spaceShips import SpaceShip
from data import *
import plotly.graph_objects as go
from math import sqrt
PLANET_COLORS = {
    "Sun": "yellow",
    "Mercury": "darkgray",
    "Venus": "orange",
    "Earth": "blue",
    "Mars": "red",
    "Jupiter": "brown",
    "Saturn": "goldenrod",
    "Uranus": "lightblue",
    "Neptune": "darkblue"
}
def initialize_planets(start):
    planets = {
        "Sun": Planet("Sun", 10, Ms, start=start),
        "Mercury": Planet("Mercury", 199, Mq, start=start),
        "Venus": Planet("Venus", 299, Mv, start=start),
        "Earth": Planet("Earth", 399, Me, start=start),
        "Mars": Planet("Mars", 499, Mm, start=start),
        "Jupiter": Planet("Jupiter", 599, Mj, start=start),
        "Saturn": Planet("Saturn", 699, Msa, start=start),
        "Uranus": Planet("Uranus", 799, Mu, start=start),
        "Neptune": Planet("Neptune", -32, Mn, start=start),
    }
    return planets
if __name__ == "__main__":
    start = "1977-08-20"
    planets = initialize_planets(start)
    sun = planets["Sun"]
    earth = planets["Earth"]
    vCurrent = sqrt((earth.vx[0] / daysec) ** 2 + (earth.vy[0] / daysec) ** 2 +
(earth.vz[0] / daysec) ** 2)
    vDesired = vCurrent + 16200
    vPercent = vDesired / vCurrent
```

```
        print(f"Орбитальная скорость Земли: {vCurrent:.2f} м/с")
        print(f"Начальная скорость Вояджера-2: {vDesired:.2f} м/s")
        print(f"Процент увеличения скорости: {vPercent:.2f}")
        ship = SpaceShip(721.9, sun, planets["Mercury"], planets["Venus"], earth,
planets["Mars"], planets["Jupiter"],
                         planets["Saturn"], planets["Uranus"], planets["Neptune"], daysec,
vPercent)
        ship.xlist = ship.xlist[::10]
        ship.ylist = ship.ylist[::10]
        ship.zlist = ship.zlist[::10]
        ship_vlist = ship.vlist[::10]
        for planet in planets.values():
            planet.x = planet.x[::10]
            planet.y = planet.y[::10]
            planet.z = planet.z[::10]
        fig = go.Figure()
        for name, planet in planets.items():
            fig.add_trace(go.Scatter3d(
                x=planet.x,
                y=planet.y,
                z=planet.z,
                mode='lines',
                line=dict(width=2, color=PLANET_COLORS.get(name, "gray")),
                name=f"{name} Orbit"
            ))
        for name, planet in planets.items():
            fig.add_trace(go.Scatter3d(
                x=[planet.x[0]],
                y=[planet.y[0]],
                z=[planet.z[0]],
                mode='markers',
                marker=dict(size=5, color=PLANET_COLORS.get(name, "gray")),
                name=f"{name} Position"
            ))
        fig.add_trace(go.Scatter3d(
            x=[ship.xlist[0]],
            y=[ship.ylist[0]],
            z=[ship.zlist[0]],
            mode='markers',
            marker=dict(size=5, color='red'),
```

```
        name="SpaceShip"
    ))
    frames = []
    num_frames = len(earth.x)
    for frame_idx in range(num_frames):
        frame_data = []
        for name, planet in planets.items():
            frame_data.append(go.Scatter3d(
                x=planet.x,
                y=planet.y,
                z=planet.z,
                mode='lines',
                line=dict(width=2, color=PLANET_COLORS.get(name, "gray")),
                name=f"{name} Orbit"
            ))
        for name, planet in planets.items():
            frame_data.append(go.Scatter3d(
                x=[planet.x[frame_idx]],
                y=[planet.y[frame_idx]],
                z=[planet.z[frame_idx]],
                mode='markers',
                marker=dict(size=5, color=PLANET_COLORS.get(name, "gray")),
                name=f"{name} Position"
            ))
        frame_data.append(go.Scatter3d(
            x=ship.xlist[:frame_idx + 1],
            y=ship.ylist[:frame_idx + 1],
            z=ship.zlist[:frame_idx + 1],
            mode='lines+markers',
            line=dict(width=1, color='red'),
            marker=dict(size=2, color='black'),
            name="SpaceShip Trajectory"
        ))
        frame_data.append(go.Scatter3d(
            x=[0], y=[0], z=[0],
            mode="text",
            text=[f"Date: {earth.d[frame_idx]}<br>Speed:
{ship_vlist[frame_idx]:.2f} m/s"],
            textposition="top center",
            name="Info"
```

```
            ))
            frames.append(go.Frame(data=frame_data, name=str(frame_idx)))
        fig.frames = frames
        fig.update_layout(
            scene=dict(
                xaxis=dict(title='X (м)', range=[-10 * AU, 10 * AU]),
                yaxis=dict(title='Y (м)', range=[-10 * AU, 10 * AU]),
                zaxis=dict(title='Z (м)', range=[-2 * AU, 2 * AU])
            ),
            title='3D-анимация орбит планет и траектории космического
аппарата',
            updatemenus=[{
                'type': 'buttons',
                'buttons': [
                    {
                        'label': 'Play',
                        'method': 'animate',
                        'args': [None, {'frame': {'duration': 50, 'redraw': True},
'fromcurrent': True}]
                    },
                    {
                        'label': 'Pause',
                        'method': 'animate',
                        'args': [[None], {'frame': {'duration': 0, 'redraw': True}, 'mode':
'immediate'}]
                    }
                ]
            }]
        )
        fig.show()
```

ПРИЛОЖЕНИЕ Б

```
from __future__ import annotations

from astroquery.jplhorizons import Horizons

import logging

import json
```

```
        import os
        logging.basicConfig(level=logging.INFO, format="%(asctime)s
[%(levelname)s] %(message)s")
        class Planet:
            AU2meters: int = 149597870700
            step: str = "1d"
            cache_dir: str = "cache"
            def __str__(self):
                return f"{self.name} with ID {self.id}"
            def getVectors(self):
                cache_file = os.path.join(self.cache_dir,
f"{self.name}_{self.start}_{self.stop}_{self.step}.json")
                if os.path.exists(cache_file):
                    logging.info(f"Чтение данных из кэша для планеты {self.name}...")
                    with open(cache_file, "r") as f:
                        data = json.load(f)
                        self.x = data["x"]
                        self.y = data["y"]
                        self.z = data["z"]
                        self.vx = data["vx"]
                        self.vy = data["vy"]
                        self.vz = data["vz"]
                        self.d = data["datetime"]
                    logging.info(f"Данные для планеты {self.name} успешно загружены
из кэша.")
                    return
                try:
                    logging.info(f"Запрос данных для планеты {self.name} (ID:
```

```
{self.id})...")
                obj = Horizons(
                    id=self.id,
                    location=None,
                    epochs={'start': self.start, 'stop': self.stop, 'step': self.step}
                )
                eph = obj.vectors()
                self.x = [a * self.AU2meters for a in eph['x']]
                self.y = [b * self.AU2meters for b in eph['y']]
                self.z = [c * self.AU2meters for c in eph['z']]
                self.vx = [d * self.AU2meters for d in eph['vx']]
                self.vy = [e * self.AU2meters for e in eph['vy']]
                self.vz = [f * self.AU2meters for f in eph['vz']]
                self.d = [g.split(" ")[1] for g in eph["datetime_str"]]
                logging.info(f"Сохранение данных для планеты {self.name} в
кэш...")
                os.makedirs(self.cache_dir, exist_ok=True)
                with open(cache_file, "w") as f:
                    json.dump({
                        "x": self.x,
                        "y": self.y,
                        "z": self.z,
                        "vx": self.vx,
                        "vy": self.vy,
                        "vz": self.vz,
                        "datetime": self.d
                    }, f)
```

```
                logging.info(f"Данные для планеты {self.name} успешно сохранены в кэш.")
            except Exception as e:
                logging.error(f"Ошибка при получении данных для планеты {self.name}: {e}")
                raise
        def __init__(self, name, num, mass, start="1977-08-20", stop="1989-10-02", auto_load=True):
            self.name = name
            self.id = num
            self.mass = mass
            self.start = start
            self.stop = stop
            if auto_load:
                self.getVectors()
```

ПРИЛОЖЕНИЕ В

```
from __future__ import annotations
from astroquery.jplhorizons import Horizons
from data import *
import logging
from math import sqrt
logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s")
class SpaceShip:
    xv, yv, zv = 0, 0, 0  # начальные скорости
    xlist, ylist, zlist, vlist = [], [], [], []  # списки координат и скорости
    dt = 86400  # временной шаг = 1 день
```

```
        def computeGravity(self, planets, position):
            hx, hy, hz = position
            gx, gy, gz = 0, 0, 0
            for planet in planets:
                try:
                    gravconst = G * planet.mass
                    rx = hx - planet.x[self.c]
                    ry = hy - planet.y[self.c]
                    rz = hz - planet.z[self.c]
                    r3 = (rx**2 + ry**2 + rz**2)**1.5  # Куб расстояния
                    if r3 == 0:
                        logging.warning(f"Деление на ноль при расчете гравитации для
планеты {planet.name}")
                        continue
                    gx -= gravconst * rx / r3
                    gy -= gravconst * ry / r3
                    gz -= gravconst * rz / r3
                except IndexError as e:
                    logging.error(f"Ошибка доступа к данным планеты {planet.name}:
{e}")
                    continue
            return gx, gy, gz
        def rungeKuttaStep(self, planets):
            x, y, z = self.x, self.y, self.z
            vx, vy, vz = self.xv, self.yv, self.zv
            dt = self.dt
            def accel(position):
```

```
                return self.computeGravity(planets, position)
            k1vx, k1vy, k1vz = accel((x, y, z))
            k1x, k1y, k1z = vx, vy, vz
            k2vx, k2vy, k2vz = accel((x + 0.5 * k1x * dt, y + 0.5 * k1y * dt, z + 0.5 *
k1z * dt))
            k2x, k2y, k2z = vx + 0.5 * k1vx * dt, vy + 0.5 * k1vy * dt, vz + 0.5 *
k1vz * dt
            k3vx, k3vy, k3vz = accel((x + 0.5 * k2x * dt, y + 0.5 * k2y * dt, z + 0.5 *
k2z * dt))
            k3x, k3y, k3z = vx + 0.5 * k2vx * dt, vy + 0.5 * k2vy * dt, vz + 0.5 *
k2vz * dt
            k4vx, k4vy, k4vz = accel((x + k3x * dt, y + k3y * dt, z + k3z * dt))
            k4x, k4y, k4z = vx + k3vx * dt, vy + k3vy * dt, vz + k3vz * dt
            self.x += (k1x + 2 * k2x + 2 * k3x + k4x) * dt / 6
            self.y += (k1y + 2 * k2y + 2 * k3y + k4y) * dt / 6
            self.z += (k1z + 2 * k2z + 2 * k3z + k4z) * dt / 6
            self.xv += (k1vx + 2 * k2vx + 2 * k3vx + k4vx) * dt / 6
            self.yv += (k1vy + 2 * k2vy + 2 * k3vy + k4vy) * dt / 6
            self.zv += (k1vz + 2 * k2vz + 2 * k3vz + k4vz) * dt / 6
        def checkProximity(self, target, threshold):
            if len(self.xlist) == 0 or len(target.x) == 0:
                logging.error(f"Недостаточно данных для расчета траекторий:
xlist={len(self.xlist)}, target.x={len(target.x)}")
                return False
            min_len = min(len(self.xlist), len(target.x))
            logging.info(f"Минlen: {min_len}")
            distances = [
```

```
                sqrt((self.xlist[i] - target.x[i]) ** 2 +
                    (self.ylist[i] - target.y[i]) ** 2 +
                    (self.zlist[i] - target.z[i]) ** 2)
                for i in range(min_len)
            ]
            min_distance = min(distances)
            logging.info(f"Минимальное расстояние до {target.name}:
{min_distance / 1e3:.2f} км")
            self.xlist.clear()
            self.ylist.clear()
            self.zlist.clear()
            self.vlist.clear()
            return min_distance <= threshold
        def __init__(self, mass, sun, mercury, venus, earth, mars, jupiter, saturn,
uranus, neptune, timestep, velocity=1):
            logging.info("Инициализация космического аппарата...")
            self.earth = earth
            self.x = self.earth.x[0]
            self.y = self.earth.y[0]
            self.z = self.earth.z[0]
            if earth.vx is None or earth.vy is None or earth.vz is None:
                raise ValueError("Данные для скорости Земли не загружены.
Вызывайте getVectors() для Земли.")
            self.xv = self.earth.vx[0] / timestep * velocity
            self.yv = self.earth.vy[0] / timestep * velocity
            self.zv = self.earth.vz[0] / timestep * velocity
            self.mass = mass
```

```
        self.dt = timestep
        self.c = 0
        planets = [sun, mercury, venus, mars, jupiter, saturn, uranus, neptune]
        for planet in planets:
            if len(planet.x) != len(planet.y) or len(planet.x) != len(planet.z):
                raise ValueError(
                    f"Длины массивов координат планеты {planet.name} не
совпадают!"
                )
        for step in range(len(earth.x)):
            self.rungeKuttaStep(planets)
            self.xlist.append(self.x)
            self.ylist.append(self.y)
            self.zlist.append(self.z)
            self.vlist.append((self.xv**2 + self.yv**2 + self.zv**2)**0.5)
            self.c += 1
            #logging.info(f"Шаг {step + 1}/{len(earth.x)} завершён.")
        logging.info("Инициализация завершена.")
```

ПРИЛОЖЕНИЕ Г

```
G       = 6.67e-11
Ms      = 1.9885e30      # sun
Mq      = 0.3301e24      # mercury
Mv      = 4.8673e24      # venus
Me      = 5.9722e24      # earth
Mm      = 0.64169e24     # mars
```

```
Mj          = 1898.13e24        # jupiter
Msa         = 568.32e24         # saturn
Mu          = 86.811e24         # uranus
Mn          = 102.409e24        # neptune
AU          = 149597870700      # a.e.
daysec      = 24.0*60*60
```

ПРИЛОЖЕНИЕ Д

```
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from jplephem.spk import SPK
from fourthRungeKutta import computeRk4
import numpy as np
KERNEL = SPK.open('de440.bsp')
MASS_SUN = 1.989e30
def julian_to_date(julian_date):
    jd = julian_date - 2440587.5
    return datetime(1970, 1, 1) + timedelta(days=jd)
def calculate_planet_positions(julian_time):
    positions = {}
    for planet_id, name in [(3, 'earth'), (5, 'jupiter'), (6, 'saturn'),
                            (7, 'uranus'), (8, 'neptune'), (9, 'pluto')]:
        pos_km = KERNEL[0, planet_id].compute(julian_time)
        positions[name] = pos_km * 1000  # Переводим в метры
    return positions
def update_trajectory(xCraft, yCraft, zCraft, vxCraft, vyCraft, vzCraft,
```

```
                timestep, mass_sun=MASS_SUN):
    k = computeRk4(xCraft, yCraft, zCraft, 0, 0, 0,
                   vxCraft, vyCraft, vzCraft, mass_sun, timestep)
    xCraft += (timestep / 6) * (k[0] + 2 * k[6] + 2 * k[12] + k[18])
    yCraft += (timestep / 6) * (k[1] + 2 * k[7] + 2 * k[13] + k[19])
    zCraft += (timestep / 6) * (k[2] + 2 * k[8] + 2 * k[14] + k[20])
    vxCraft += (timestep / 6) * (k[3] + 2 * k[9] + 2 * k[15] + k[21])
    vyCraft += (timestep / 6) * (k[4] + 2 * k[10] + 2 * k[16] + k[22])
    vzCraft += (timestep / 6) * (k[5] + 2 * k[11] + 2 * k[17] + k[23])
    return xCraft, yCraft, zCraft, vxCraft, vyCraft, vzCraft
def simulate_leg(xCraft, yCraft, zCraft, vxCraft, vyCraft, vzCraft,
                 t_start, t_end, timestep, julian_time,
                 trajectories, velocities):
    t = t_start
    while t < t_end:
        xCraft, yCraft, zCraft, vxCraft, vyCraft, vzCraft = update_trajectory(
            xCraft, yCraft, zCraft, vxCraft, vyCraft, vzCraft, timestep)
        updated_julian_time = julian_time + (t / 86400.0)
        planet_positions = calculate_planet_positions(updated_julian_time)
        for planet, pos in planet_positions.items():
            trajectories[f'x{planet.capitalize()}Trajectory'].append(pos[0])
            trajectories[f'y{planet.capitalize()}Trajectory'].append(pos[1])
            trajectories[f'z{planet.capitalize()}Trajectory'].append(pos[2])
        trajectories['xTrajectory'].append(xCraft[0])
        trajectories['yTrajectory'].append(yCraft[0])
        trajectories['zTrajectory'].append(zCraft[0])
        velocities.append(vxCraft**2 + vyCraft**2 + vzCraft**2)
```

```
            t += timestep
        return xCraft, yCraft, zCraft
    def Plotting(xCraftStart, yCraftStart, zCraftStart,
                 xVel_leg1, yVel_leg1, zVel_leg1, tof_leg1,
                 xVel_leg2, yVel_leg2, zVel_leg2, tof_leg2,
                 xVel_leg3, yVel_leg3, zVel_leg3, tof_leg3,
                 xVel_leg4, yVel_leg4, zVel_leg4, tof_leg4,
                 timestep, julianTime):
        trajectories = {
            'xTrajectory': [], 'yTrajectory': [], 'zTrajectory': [],
            'xEarthTrajectory': [], 'yEarthTrajectory': [], 'zEarthTrajectory': [],
            'xJupiterTrajectory': [], 'yJupiterTrajectory': [], 'zJupiterTrajectory': [],
            'xSaturnTrajectory': [], 'ySaturnTrajectory': [], 'zSaturnTrajectory': [],
            'xUranusTrajectory': [], 'yUranusTrajectory': [], 'zUranusTrajectory': [],
            'xNeptuneTrajectory': [], 'yNeptuneTrajectory': [], 'zNeptuneTrajectory':
[],
            'xPlutoTrajectory': [], 'yPlutoTrajectory': [], 'zPlutoTrajectory': []
        }
        velocities = []
        xCraft, yCraft, zCraft = xCraftStart, yCraftStart, zCraftStart
        legs = [
            (xVel_leg1, yVel_leg1, zVel_leg1, tof_leg1),
            (xVel_leg2, yVel_leg2, zVel_leg2, tof_leg2),
            (xVel_leg3, yVel_leg3, zVel_leg3, tof_leg3),
            (xVel_leg4, yVel_leg4, zVel_leg4, tof_leg4)
        ]
        t_current = 0
```

```
for i, (vx, vy, vz, tof) in enumerate(legs):
    if tof > 0:  # Пропускаем пустые участки
        t_start = t_current
        t_end = t_start + tof * 86400
        xCraft, yCraft, zCraft = simulate_leg(
            xCraft, yCraft, zCraft, vx, vy, vz,
            t_start, t_end, timestep, julianTime,
            trajectories, velocities
        )
        t_current = t_end
plt.ion()
fig = plt.figure(figsize=(8, 6))
plt.xlim([-5e12, 5e12])
plt.ylim([-5e12, 5e12])
ax1 = plt.gca()
ax2 = ax1.twinx()
date_text = plt.text(0.02, 0.95, '', transform=ax1.transAxes, fontsize=10)
speed_text = plt.text(0.02, 0.90, '', transform=ax1.transAxes, fontsize=10)
colors = {
    'Trajectory': 'black',
    'Earth': 'blue',
    'Jupiter': 'orange',
    'Saturn': 'red',
    'Uranus': 'magenta',
    'Neptune': 'purple',
    'Pluto': 'brown'
}
```

```
    for i in range(0, len(trajectories['xTrajectory']), 30):
        current_date = julian_to_date(julianTime + (i * timestep / 86400.0))
        date_text.set_text(f'Дата: {current_date.strftime("%Y-%m-%d")}')
        if i < len(velocities):
            current_speed = np.sqrt(velocities[i]) / 1000
            speed_text.set_text(f'Скорость: {float(current_speed):.2f} км/с')
        for name, color in colors.items():
            if name == 'Trajectory':
                ax1.scatter(trajectories[f'x{name}'][i],
                            trajectories[f'y{name}'][i],
                            s=1, c=color, label='Космический аппарат')
            else:
                ax1.scatter(trajectories[f'x{name}Trajectory'][i],
                            trajectories[f'y{name}Trajectory'][i],
                            s=1, c=color, label=name)
        if i == 0:
            ax1.legend(loc='upper right', fontsize=8)
        plt.draw()
        plt.pause(0.1)
```

ПРИЛОЖЕНИЕ Е

```
import time
import numpy as np
from numpy import cross, pi
from numpy.linalg import norm
from scipy.special import hyp2f1
```

```
def solve_lambert(
    mu,
    r1,
    r2,
    tof,
    M=0,
    maxiter=35,
    atol=1e-5,
    rtol=1e-7,
):
    # Хорда
    c = r2 - r1
    c_norm, r1_norm, r2_norm = norm(c), norm(r1), norm(r2)
    # Полупериметр
    s = (r1_norm + r2_norm + c_norm) * 0.5
    # Единичные векторы
    i_r1, i_r2 = r1 / r1_norm, r2 / r2_norm
    i_h = cross(i_r1, i_r2)
    i_h = i_h / norm(i_h)
    # Геометрия задачи
    ll = np.sqrt(1 - min(1.0, c_norm / s))
    # Вычисление фундаментальных тангенциальных направлений
    if i_h[2] < 0:
        ll = -ll
        i_t1, i_t2 = cross(i_r1, i_h), cross(i_r2, i_h)
    else:
        i_t1, i_t2 = cross(i_h, i_r1), cross(i_h, i_r2)
```

```
        # Безразмерное время полета
        T = np.sqrt(2 * mu / s ** 3) * tof
        # Поиск решений
        x, y, numiter, tpi = _find_xy(ll, T, M, maxiter, atol, rtol)
        # Реконструкция
        gamma = np.sqrt(mu * s / 2)
        rho = (r1_norm - r2_norm) / c_norm
        sigma = np.sqrt(1 - rho ** 2)
        # Вычисление радиальных и тангенциальных компонентов
        V_r1, V_r2, V_t1, V_t2 = _reconstruct(x, y, r1_norm, r2_norm, ll, gamma,
rho, sigma)
        # Решение для начальной и конечной скорости
        v1 = V_r1 * (r1 / r1_norm) + V_t1 * i_t1
        v2 = V_r2 * (r2 / r2_norm) + V_t2 * i_t2
        return v1, v2
    def _reconstruct(x, y, r1, r2, ll, gamma, rho, sigma):
        # Реконструкция векторов скорости.
        V_r1 = gamma * ((ll * y - x) - rho * (ll * y + x)) / r1
        V_r2 = -gamma * ((ll * y - x) + rho * (ll * y + x)) / r2
        V_t1 = gamma * sigma * (y + ll * x) / r1
        V_t2 = gamma * sigma * (y + ll * x) / r2
        return [V_r1, V_r2, V_t1, V_t2]

    def _find_xy(ll, T, M, maxiter, atol, rtol):
        # Вычисляет все x, y для заданного числа оборотов.
        assert abs(ll) < 1
        M_max = np.floor(T / pi)
```

```
        T_00 = np.arccos(ll) + ll * np.sqrt(1 - ll ** 2)
        if T < T_00 + M_max * pi and M_max > 0:
            _, T_min = _compute_T_min(ll, M_max, maxiter, atol, rtol)
            if T < T_min:
                M_max -= 1
        if M > M_max:
            raise ValueError("Нет допустимого решения, попробуйте уменьшить
M!")
        x_0 = _initial_guess(T, ll, M)
        x, numiter, tpi = _householder(x_0, T, ll, M, atol, rtol, maxiter)
        y = _compute_y(x, ll)
        return x, y, numiter, tpi
    def _compute_y(x, ll):
        # Вычисляет y.
        return np.sqrt(1 - ll ** 2 * (1 - x ** 2))
    def _compute_psi(x, y, ll):
        # Вычисляет вспомогательный угол psi.
        if -1 <= x < 1:
            return np.arccos(x * y + ll * (1 - x ** 2))
        elif x > 1:
            return np.arcsinh((y - x * ll) * np.sqrt(x ** 2 - 1))
        else:
            return 0.0
    def _tof_equation(x, T0, ll, M):
        # Уравнение времени полета.
        return _tof_equation_y(x, _compute_y(x, ll), T0, ll, M)
    def _tof_equation_y(x, y, T0, ll, M):
```

```
    # Уравнение времени полета с предварительно вычисленным y.
    if M == 0 and np.sqrt(0.6) < x < np.sqrt(1.4):
        eta = y - ll * x
        S_1 = (1 - ll - x * eta) * 0.5
        Q = 4 / 3 * hyp2f1(3, 1, 5 / 2, S_1)
        T_ = (eta ** 3 * Q + 4 * ll * eta) * 0.5
    else:
        psi = _compute_psi(x, y, ll)
        T_ = np.divide(
            np.divide(psi + M * pi, np.sqrt(np.abs(1 - x ** 2))) - x + ll * y,
            (1 - x ** 2),
        )
    return T_ - T0
def _tof_equation_p(x, y, T, ll):
    # Производная уравнения времени полета.
    return (3 * T * x - 2 + 2 * ll ** 3 * x / y) / (1 - x ** 2)
def _tof_equation_p2(x, y, T, dT, ll):
    # Вторая производная уравнения времени полета.
    return (3 * T + 5 * x * dT + 2 * (1 - ll ** 2) * ll ** 3 / y ** 3) / (1 - x ** 2)
def _tof_equation_p3(x, y, _, dT, ddT, ll):
    # Третья производная уравнения времени полета.
    return (7 * x * ddT + 8 * dT - 6 * (1 - ll ** 2) * ll ** 5 * x / y ** 5) / (
        1 - x ** 2
    )
def _compute_T_min(ll, M, maxiter, atol, rtol):
    # Вычисляет минимальное T.
    if ll == 1:
```

```
        x_T_min = 0.0
        T_min = _tof_equation(x_T_min, 0.0, ll, M)
    else:
        if M == 0:
            x_T_min = np.inf
            T_min = 0.0
        else:
            x_i = 0.1
            T_i = _tof_equation(x_i, 0.0, ll, M)
            x_T_min = _halley(x_i, T_i, ll, atol, rtol, maxiter)
            T_min = _tof_equation(x_T_min, 0.0, ll, M)
    return [x_T_min, T_min]
def _initial_guess(T, ll, M):
    # Начальное приближение.
    if M == 0:
        T_0 = np.arccos(ll) + ll * np.sqrt(1 - ll ** 2) + M * pi
        T_1 = 2 * (1 - ll ** 3) / 3
        if T >= T_0:
            x_0 = (T_0 / T) ** (2 / 3) - 1
        elif T < T_1:
            x_0 = 5 / 2 * T_1 / T * (T_1 - T) / (1 - ll ** 5) + 1
        else:
            x_0 = (T_0 / T) ** (np.log2(T_1 / T_0)) - 1


        return x_0
    else:
        x_0l = (((M * pi + pi) / (8 * T)) ** (2 / 3) - 1) / (
```

```
            ((M * pi + pi) / (8 * T)) ** (2 / 3) + 1
        )
        x_0r = (((8 * T) / (M * pi)) ** (2 / 3) - 1) / (
            ((8 * T) / (M * pi)) ** (2 / 3) + 1
        )
        return x_0l  # Всегда используем нижний путь
def _halley(p0, T0, ll, atol, rtol, maxiter):
    # Находит минимум уравнения времени полета методом Халли.
    for ii in range(1, maxiter + 1):
        y = _compute_y(p0, ll)
        fder = _tof_equation_p(p0, y, T0, ll)
        fder2 = _tof_equation_p2(p0, y, T0, fder, ll)
        if fder2 == 0:
            raise RuntimeError("Производная равна нулю")
        fder3 = _tof_equation_p3(p0, y, T0, fder, fder2, ll)
        p = p0 - 2 * fder * fder2 / (2 * fder2 ** 2 - fder * fder3)
        if abs(p - p0) < rtol * np.abs(p0) + atol:
            return p
        p0 = p
    raise RuntimeError("Не удалось достичь сходимости")
def _householder(p0, T0, ll, M, atol, rtol, maxiter):
    # Находит нуль уравнения времени полета методом Хаусхолдера.
    tic = time.perf_counter()
    for numiter in range(1, maxiter + 1):
        y = _compute_y(p0, ll)
        fval = _tof_equation_y(p0, y, T0, ll, M)
        T = fval + T0
```

```
    fder = _tof_equation_p(p0, y, T, ll)
    fder2 = _tof_equation_p2(p0, y, T, fder, ll)
    fder3 = _tof_equation_p3(p0, y, T, fder, fder2, ll)
    p = p0 - fval * (
        (fder ** 2 - fval * fder2 / 2)
        / (fder * (fder ** 2 - fval * fder2) + fder3 * fval ** 2 / 6)
    )
    if abs(p - p0) < rtol * np.abs(p0) + atol:
        tac = time.perf_counter()
        tpi = (tac - tic) / numiter

        return p, numiter, tpi
    p0 = p
raise RuntimeError("Не удалось достичь сходимости")
```