CEUB | PIC

# SIMULAÇÃO COMPUTACIONAL SOBRE A TROCA DE ÓRBITAS DE UM SATÉLITE ARTIFICIAL

Professor(a) orientador(a): Dr. Ednardo Paulo Spaniol

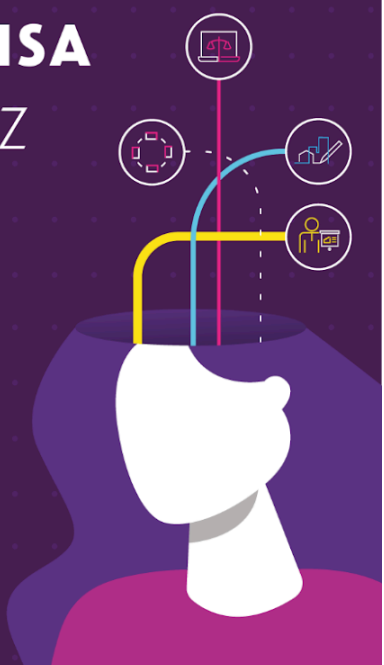
Alunos: Nícolas Tomaschewski Lopes

PROGRAMA DE
INICIAÇÃO CIENTÍFICA
PIC/CEUB

**RELATÓRIOS DE PESQUISA**

VOLUME 9 Nº 1- JAN/DEZ

**·2023·**



ISSN: 2595-4563

fapdf CNPq CEUB

**CENTRO UNIVERSITÁRIO DE BRASÍLIA - CEUB**
**PROGRAMA DE INICIAÇÃO CIENTÍFICA**

**NÍCOLAS TOMASCHEWSKI LOPES**

# SIMULAÇÃO COMPUTACIONAL SOBRE A TROCA DE ÓRBITAS DE UM SATÉLITE ARTIFICIAL

Relatório final de pesquisa de Iniciação Científica apresentado à Assessoria de Pós-Graduação e Pesquisa.

Orientação: Dr. Ednardo Paulo Spaniol

**BRASÍLIA**

**2024**

## AGRADECIMENTOS

Agradeço primeiramente à minha família por todo o apoio que me deram ao longo dos anos. Agradeço também ao meu professor Ednardo por todos os ensinamentos e orientações ao longo do desenvolvimento do trabalho, e ao CEUB por tornar possível a execução desse projeto. Além disso, agradeço a FAP-DF pelo suporte financeiro destinado a este estudo.

Este trabalho apresenta o desenvolvimento de simulações computacionais que auxiliam nos estudos sobre órbitas descritas por satélites artificiais, com foco na manobra de Hohmann, uma técnica fundamental para transferência eficiente de satélites entre órbitas circulares. Além das aplicações práticas na indústria aeroespacial, este artigo contribui para o desenvolvimento contínuo dos estudos sobre órbitas, um tema essencial e relevante da física, especialmente em sua área que lida com simulações computacionais. O objetivo central do trabalho é construir programas robustos e otimizados desenvolvidos utilizando a linguagem Python, com auxílio de suas bibliotecas, capazes de representar, com a maior precisão possível, a manobra, descrita matematicamente em detalhes ao longo dos tópicos do projeto. Durante o desenvolvimento, diversas ideias foram colocadas em prática, o que é útil para acompanhar as diferentes versões e a evolução do projeto. O estudo foi bem-sucedido em demonstrar a eficiência da manobra de Hohmann, bem como sua importância na indústria. A versão do código desenvolvida para simular as múltiplas trocas em uma única manobra merece atenção, pois se trata de uma prática real da indústria aeroespacial que o trabalho conseguiu contemplar com sucesso, além de apresentar exemplos de órbitas realmente utilizadas na prática. A pesquisa abrange diferentes versões da manobra de Hohmann, com foco especial em uma representação bidimensional do sistema, enquanto considera brevemente as possibilidades e limitações de estudos tridimensionais. Os resultados obtidos não apenas confirmam a eficiência da manobra de Hohmann, mas também destacam sua relevância prática na dinâmica orbital e sua aplicação na indústria aeroespacial moderna.

**Palavras-chave:** Simulação Computacional; Física Computacional; Manobra de Hohmann.

# 1. INTRODUÇÃO

Os satélites artificiais desempenham um papel fundamental na era em que vivemos. Por exemplo, as telecomunicações e a aviação dependem amplamente desses satélites para funcionarem adequadamente. Portanto, é de extrema importância estudar como posicionar e manter esses corpos em movimento em suas órbitas corretas. O conhecimento nesse campo é crucial para garantir a operação eficiente e confiável dos satélites, permitindo que eles cumpram suas funções vitais na comunicação global e na facilitação das atividades aeroespaciais. Neste sentido, se faz necessário o estudo teórico da dinâmica de satélites artificiais, ou seja, dos elementos orbitais clássicos e tipos de órbitas.

Em 1925, o engenheiro alemão Walter Hohmann dedicou-se ao estudo do problema da transferência de um veículo espacial entre duas órbitas circulares em um campo gravitacional newtoniano. Esse problema era de extrema importância para o desenvolvimento da exploração espacial, pois a transferência eficiente entre órbitas era essencial para economizar combustível e tempo nas missões espaciais. Hohmann desenvolveu uma solução matemática que ficou conhecida como "manobra de Hohmann", que consistia em realizar duas manobras de propulsão, uma para se colocar em uma órbita elíptica de transferência e outra para se inserir na órbita final desejada.

A manobra de Hohmann permitia que o veículo espacial aproveitasse o campo gravitacional para realizar a transferência entre as órbitas de forma mais eficiente. A primeira manobra era realizada no perigeu da órbita inicial, onde o veículo aumentava sua velocidade para se inserir em uma órbita elíptica de transferência. No apoastro dessa órbita, o veículo realizava a segunda manobra, diminuindo sua velocidade para se inserir na órbita final desejada. Essa estratégia de manobras proporcionava economia de energia, pois permitia que a maior parte do impulso necessário fosse obtida a partir da energia gravitacional.

O estudo de Hohmann foi um marco importante na exploração espacial, pois trouxe uma solução matemática para o problema da transferência entre órbitas circulares em um campo gravitacional newtoniano. Essa solução continua sendo

utilizada até os dias de hoje como base para planejar trajetórias de espaçonaves e satélites. Além disso, a manobra de Hohmann serviu de inspiração para o desenvolvimento de outras técnicas mais avançadas, como as manobras assistidas pela gravidade, que aproveitam a influência gravitacional de corpos celestes para impulsionar a espaçonave de forma ainda mais eficiente.

Neste trabalho, apresentamos uma fundamentação teórica detalhada sobre a manobra de Hohmann. Em seguida, são apresentados os códigos desenvolvidos em Python, acompanhados de exemplos das simulações realizadas. A conclusão discute os resultados obtidos, oferecendo uma análise crítica. Além da base teórica inicial, também desenvolvemos um estudo abrangente sobre órbitas e os formalismos lagrangiano e hamiltoniano, cujos detalhes estão disponíveis nos apêndices. O trabalho foi desenvolvido utilizando LaTex e todos os códigos foram desenvolvidos utilizando a linguagem de programação Python. Durante o avanço do projeto, inteligências artificiais generativas foram utilizadas para auxiliar no desenvolvimento dos códigos apresentados [1].

**OBJETIVOS**

Desenvolver uma análise detalhada da manobra de Hohmann, utilizando abordagens analíticas e computacionais. O projeto visa calcular e simular as trajetórias e variações de velocidade envolvidas na manobra, proporcionando uma compreensão aprofundada dos princípios que regem a transferência orbital eficiente entre duas órbitas circulares.

## 2. FUNDAMENTAÇÃO TEÓRICA

A manobra de Hohmann é uma técnica amplamente utilizada para transferir espaçonaves entre duas órbitas circulares com raios diferentes. Este procedimento é considerado energeticamente eficiente e envolve dois impulsos: um para transferir a espaçonave para uma órbita elíptica de transferência e outro para inseri-la na órbita circular final. Assim a diferença de velocidades no periastro e apoastro entre as órbitas circulares (inicial e final) e a órbita elíptica intermédia é exatamente o impulso necessário.

## 2.1 DESCRIÇÃO DA MANOBRA

O satélite inicia em uma órbita circular de raio $r_1$ e usa um impulso para se transferir para a órbita elíptica de transferência. O semi-eixo maior desta órbita elíptica é

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

Onde $r_2$ é o raio da órbita circular final desejada.

## 2.2 VELOCIDADE NAS ÓRBITAS

No intuito de obtermos as velocidades das órbitas que vamos considerar em nosso estudo devemos assumir algumas premissas. Em primeiro lugar, os corpos são considerados esféricos de maneira que pode-se assumir que as forças atuam no centro dos referidos corpos. Além disso, as únicas forças externas que atuam nos corpos são as forças gravitacionais. Neste contexto, vamos considerar a Lei da Gravitacional Universal proposta por Newton. O módulo da força gravitacional entre dois objetos de massas $m_1$ e $m_2$ é dada por:

$$F = G\frac{m_1 m_2}{r^2}$$

Onde $G$ é a constante gravitacional de valor $6,67\ X\ 10^{-11}\ Nm^2/kg^2$ e $r$ é a distância entre os dois corpos. Ao considerarmos a Lei da Gravitação Universal e a aceleração centrípeta pode-se escrever as velocidades nas órbitas circulares iniciais e finais, ou seja,

$$v_1 = \sqrt{\frac{GM}{r_1}},\ v_2 = \sqrt{\frac{GM}{r_2}}$$

Onde $GM$ é o produto da constante gravitacional universal e a massa do corpo central, que pode ser qualquer astro.

## 2.3 IMPULSO NECESSÁRIO

Os impulsos $\Delta v_1$ e $\Delta v_2$ necessários para a manobra são:

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

e

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

onde $\Delta v_1 = v_p - v_1$ e $\Delta v_2 = v_2 - v_a$ , ou seja, são as diferenças entre as velocidades no apoastro e periastro e aquelas das órbitas circulares.

## 2.4 DERIVAÇÃO DA VELOCIDADE TANGENCIAL NO PERIASTRO

Vamos deduzir a velocidade no periastro $v_p$ que é na verdade a velocidade tangencial $v_\theta$ , já que a velocidade radial é nula nesse ponto.

A taxa de variação angular $\theta^\bullet$ é obtida pela relação do momento angular por unidade de massa, ou seja,

$$h = \frac{L}{m} = r^2\theta^\bullet$$

O momento angular por unidade de massa $h$ é constante e dado por

$$h = \sqrt{GMp}$$

onde $p = a\left(1 - e^2\right)$ é o parâmetro da equação da órbita

$$r = \frac{p}{1+e\cos\theta}$$

e $a = \left(r_1 + r_2\right)/2$.

Portanto,

$$\theta^{\bullet} = \frac{h}{r^2} = \frac{\sqrt{GMp}}{r^2}$$

como a velocidade tangencial é

$$v_{\theta} = r\theta^{\bullet},$$

então

$$v_{\theta} = r\frac{\sqrt{GMp}}{r^2} = \frac{\sqrt{GMp}}{r}$$

No periastro, a distância radial $r$ é mínima e igual a:

$$r = r_1 = \frac{p}{1+e}$$

Assim, substituindo $r_1$ na expressão da velocidade tangencial acima, isto é,

$$v_{\theta} = \frac{\sqrt{GMp}}{r_1} = \sqrt{GMp}\frac{(1+e)}{\sqrt{p}}$$

Usando o fato de que pode-se escrever

$$e = \frac{r_2 - r_1}{r_1 + r_2}$$

chegamos a

$$v_0 = \frac{\sqrt{GMp}}{r_1}\sqrt{\frac{2r_2}{r_1+r_2}}.$$

Esta expressão reflete a velocidade no periastro, onde a energia cinética é maximizada a potencial gravitacional minimizada, por estar mais próxima do corpo central.

## 2.5 RESULTADOS MATEMÁTICOS DA MANOBRA

Observamos que os estudos de Hohmann produziram importantes resultados para o desenvolvimento da exploração espacial. As contas apresentadas na seção (2.3) podem ser reescritas de uma forma alternativa. Inicialmente o veículo precisa sofrer a primeira variação de velocidade para sair da órbita original. Essa variação de velocidade pode ser expressa da seguinte forma:

$$\Delta V = V_0 \left| \sqrt{\frac{2\left(\frac{R'}{R}\right)}{\frac{R'}{R}+1}} \right| - 1.$$

Onde $V_0$ simboliza a velocidade inicial do corpo, R e R' denotam os raios das órbitas inicial e final, respectivamente. Quando o veículo atinge o ponto intermediário da trajetória, um novo impulso é aplicado de forma tangencial ao movimento. Essa variação na velocidade é crucial para concluir a manobra, sendo expressa da seguinte maneira [2].

$$\Delta V' = V_0 \left| 1 - \sqrt{\frac{2}{\left(\frac{R'}{R}\right)+1}} \right| \sqrt{\frac{R}{R'}}.$$

Além disso, é possível encontrar o tempo necessário para que a manobra aconteça, isto é,

$$t = \frac{1}{2}\left(\frac{1+\frac{R'}{R}}{2}\right)^{\frac{3}{2}} T_1.$$

# 3. MÉTODO - CÓDIGO DESENVOLVIDO

Uma componente importante do trabalho são os códigos que automatizam os cálculos apresentados e geram as imagens analisadas. Para isso foi escolhida a linguagem Python e suas bibliotecas. Aqui estão todos os códigos escritos e o raciocínio por trás de sua construção:

## 3.1. PROTÓTIPO

Após todo o estudo do arcabouço teórico que envolve a manobra, uma primeira versão do código foi desenvolvida. Esse protótipo visa apenas explicar o conceito básico da manobra e realizar as contas necessárias para determinar o valor das variações de velocidade bem como o tempo necessário para finalizar a transferência. Para a realização dessas contas o usuário deve inserir a velocidade inicial do veículo, o raio da órbita que o objeto está descrevendo, o tempo necessário para completar uma volta e a órbita final que o veículo descreverá ao final da manobra.

Aqui está o código responsável por essa tarefa:

```python
import math

velocidade_inicial = float(input('Insira a velocidade inicial do veículo (em m/s): '))
orbita_inicial = float(input('Insira o raio da órbita inicial (em metros): '))
tempo_inicial = float(input('Insira o tempo necessário para o satélite completar uma volta completa ao redor da Terra (em segundos): '))
orbita_final = float(input('Insira o raio da órbita final (em metros): '))

print('O primeiro passo da manobra de Hohmann é dar um impulso ao veículo para que exista uma variação de velocidade')

modulo1 = math.sqrt((2 * (orbita_final / orbita_inicial)) / ((orbita_final / orbita_inicial) + 1)) - 1
deltav1 = velocidade_inicial * modulo1

print('Para que a manobra seja realizada com sucesso, o primeiro impulso dado ao veículo deve ser forte o suficiente para criar uma variação na velocidade de: {:.2f} m/s'.format(deltav1))

print('No momento em que o satélite se encontrar na metade da nova órbita, um segundo impulso deve ser dado, criando uma nova variação de velocidade.')

modulo2 = 1 - math.sqrt(2 / ((orbita_final / orbita_inicial) + 1))
deltav2 = velocidade_inicial * modulo2 * math.sqrt(orbita_inicial / orbita_final)

print('O valor dessa nova variação na velocidade deve ser igual a {:.2f} m/s'.format(deltav2))

print('Este segundo impulso faz com que o satélite atinja a órbita de destino.')
print('Também é possível determinar o tempo necessário para realizar a manobra em função do período da órbita inicial.')

tempo_total = 0.5 * ((1 + orbita_final / orbita_inicial) / 2) ** (3 / 2) * tempo_inicial

print('O tempo necessário para a realização dessa manobra de Hohmann é {:.2f} segundos.'.format(tempo_total))
```

Imagem 01: Código do Protótipo

## 3.2. ÓRBITA EM 2D - COORDENADAS CARTESIANAS

Em paralelo ao protótipo descrito acima foi desenvolvido um código que visa desenhar a trajetória estudada. Em um primeiro momento, essa figura foi expressa em coordenadas cartesianas e sem a possibilidade de interação com o usuário.

Aqui está o código responsável pelo desenho:

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Parâmetros
a1 = 7000  # Semi-eixo maior da órbita inicial em km
a2 = 42164  # Semi-eixo maior da órbita geostacionária em km
mu = 3.986 * 10**5  # Constante gravitacional da Terra em km^3/s^2

# Raio da órbita inicial e final
r1 = a1
r2 = a2

# Órbita de transferência
aTransfer = (r1 + r2) / 2

# Funções de posição
def r(t, a, e):
    return a * (1 - e**2) / (1 + e * np.cos(t))

def x(t, a, e):
    return r(t, a, e) * np.cos(t)

def y(t, a, e):
    return r(t, a, e) * np.sin(t)

# Excentricidades
e1 = 0  # Órbita inicial circular
eTransfer = (r2 - r1) / (r2 + r1)  # Órbita de transferência elíptica
e2 = 0  # Órbita final circular

# Trajetórias
trajectoryTransfer = np.array([[x(t, aTransfer, eTransfer), y(t, aTransfer, eTransfer)] for t in np.linspace(0, np.pi, 100)])
trajectoryFinal = np.array([[x(t, r2, e2), y(t, r2, e2)] for t in np.linspace(0, 2*np.pi, 200)])

# Criação da figura e eixos
fig, ax = plt.subplots()
ax.set_xlim(-a2, a2)
ax.set_ylim(-a2, a2)
ax.set_aspect('equal')
ax.plot(*zip(*trajectoryTransfer), color='blue')  # Trajetória de transferência (não precisa ser animada)
orbit_initial, = ax.plot([], [], 'o', color='red')
orbit_final, = ax.plot([], [], 'o', color='red')

# Desenhar órbitas
circle1 = plt.Circle((0, 0), r1, color='gray', fill=False)
circle2 = plt.Circle((0, 0), r2, color='gray', fill=False)
ax.add_patch(circle1)
ax.add_patch(circle2)

def init():
    orbit_initial.set_data([], [])
    orbit_final.set_data([], [])
    return orbit_initial, orbit_final

def update(frame):
    if frame < 200:
        t = 2 * np.pi * frame / 200
        orbit_initial.set_data(x(t, r1, e1), y(t, r1, e1))
        orbit_final.set_data([], [])
    elif frame < 300:
        t = np.pi * (frame - 200) / 100
        orbit_initial.set_data([], [])
        orbit_final.set_data(x(t, aTransfer, eTransfer), y(t, aTransfer, eTransfer))
    else:
        t = 2 * np.pi * (frame - 300) / 200
        orbit_initial.set_data([], [])
        orbit_final.set_data(x(t, r2, e2), y(t, r2, e2))
    return orbit_initial, orbit_final

# Criação da animação
ani = FuncAnimation(fig, update, frames=np.arange(0, 500), init_func=init, blit=True)

# Exibir a animação
plt.show()
```

Imagem 02: Código para Coordenadas Cartesianas

### 3.3. ÓRBITA EM 2D - COORDENADAS POLARES

Durante os estudos, foi observado que, devido a questões de simetria e apresentação das imagens, seria mais adequado representar o sistema em

coordenadas polares. Assim, o código original foi convertido para esse sistema de coordenadas. Além disso, foi implementada uma interface que solicita ao usuário a inserção dos seguintes dados: o raio da órbita inicial em quilômetros, o raio da órbita final em quilômetros, a velocidade inicial do veículo e o tempo necessário para o satélite completar uma volta ao redor da Terra.

Com as respostas fornecidas pelo usuário, o programa é capaz de calcular as duas variações de velocidade necessárias para a manobra e o tempo total envolvido, além de gerar uma figura ilustrativa do trajeto orbital. Além da conversão da imagem para coordenadas polares, foi possível integrar o trabalho desenvolvido no protótipo, resultando no que é apresentado a seguir.

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import tkinter as tk
from tkinter import simpledialog, messagebox
import math

# Função para solicitar os valores das órbitas
def solicitar_valores_orbitas():
    root = tk.Tk()
    root.withdraw()  # Esconder a janela principal
    a1 = simpledialog.askfloat("Valor da órbita inicial", "Insira o raio da órbita inicial em km:")
    a2 = simpledialog.askfloat("Valor da órbita final", "Insira o raio da órbita final em km:")
    velocidade_inicial = simpledialog.askfloat("Velocidade inicial", "Insira a velocidade inicial do veículo (em m/s):")
    tempo_inicial = simpledialog.askfloat("Tempo necessário", "Insira o tempo necessário para o satélite completar uma volta completa ao redor da Terra (em segundos):")

    modulo1 = math.sqrt((2 * (a2 / a1)) / ((a2 / a1) + 1)) - 1
    deltav1 = velocidade_inicial * modulo1

    modulo2 = 1 - math.sqrt(2 / ((a2 / a1) + 1))
    deltav2 = velocidade_inicial * modulo2 * math.sqrt(a1 / a2)

    tempo_total = 0.5 * ((1 + a2 / a1) / 2) ** (3 / 2) * tempo_inicial

    root.destroy()  # Fechar a janela

    messagebox.showinfo("Resultados", "Delta V1 = {:.2f}\nDelta V2 = {:.2f}\nTempo Total = {:.2f} segundos".format(deltav1, deltav2, tempo_total))

    return a1, a2

# Criação da janela para solicitar os valores das órbitas
def criar_janela():
    # Função para gerar a figura quando o botão for clicado
    def gerar_figura():
        a1, a2 = solicitar_valores_orbitas()
        if a1 is not None and a2 is not None:
            desenhar_figura(a1, a2)

    # Função para fechar a janela
    def sair():
        janela.destroy()

    # Criação da janela
    janela = tk.Tk()
    janela.title("Solicitar Valores de Órbitas")

    # Label e botão
    label = tk.Label(janela, text="Clique no botão para inserir os valores das órbitas.")
    label.pack(padx=10, pady=10)

    botao_inserir = tk.Button(janela, text="Inserir Dados", command=gerar_figura)
    botao_inserir.pack(padx=10, pady=5)

    botao_sair = tk.Button(janela, text="Sair", command=sair)
    botao_sair.pack(padx=10, pady=5)

    janela.mainloop()

# Desenhar a figura com os valores das órbitas inseridos
def desenhar_figura(a1, a2):
    # Raio da órbita inicial e final
    r1 = a1
    r2 = a2

    # Órbita de transferência
    aTransfer = (r1 + r2) / 2

    # Função de posição radial em coordenadas polares
    def r(t, a, e):
        return a * (1 - e**2) / (1 + e * np.cos(t))

    # Excentricidades
    e1 = 0  # Órbita inicial circular
    eTransfer = (r2 - r1) / (r2 + r1)  # Órbita de transferência elíptica
    e2 = 0  # Órbita final circular

    # Trajetórias
    theta_transfer = np.linspace(0, np.pi, 100)
    r_transfer = r(theta_transfer, aTransfer, eTransfer)

    theta_final = np.linspace(0, 2 * np.pi, 200)
    r_final = r(theta_final, r2, e2)

    # Criação da figura e eixos
    fig, ax = plt.subplots(subplot_kw={'projection': 'polar'}, figsize=(8, 8))
    fig.patch.set_facecolor('black')  # Define a cor de fundo da figura
    ax.set_facecolor('black')  # Define a cor de fundo dos eixos
    ax.set_ylim(0, a2)

    # Define a cor dos rótulos e dos ticks
    ax.tick_params(colors='white')  # Define a cor dos ticks
    ax.yaxis.label.set_color('white')  # Define a cor do rótulo do eixo y
    ax.xaxis.label.set_color('white')  # Define a cor do rótulo do eixo x
    ax.title.set_color('white')  # Define a cor do título

    # Adiciona o título com tamanho maior
    ax.set_title("Manobra de Hohmann em 2-D", fontsize=20)

    # Trajetórias
    ax.plot(np.linspace(0, 2 * np.pi, 100), [r1] * 100, color='blue', linestyle='--', label='Órbita inicial')  # Órbita inicial circular
    ax.plot(theta_transfer, r_transfer, color='orange', label='Órbita intermediária')  # Trajetória de transferência
    ax.plot(np.linspace(0, 2 * np.pi, 100), [r2] * 100, color='red', linestyle='--', label='Órbita final')  # Órbita final circular

    # Adiciona um ponto azul no centro para representar o planeta
    ax.plot(0, 0, 'bo', label='Planeta (Terra)')

    plt.show()

# Chamada para solicitar os valores das órbitas
criar_janela()
```

Imagem 03: Código para Coordenadas Polares

### 3.4. ÓRBITA EM 3D

Durante o desenvolvimento do trabalho foi cogitada a criação de uma simulação em três dimensões para ilustrar a manobra. Em seguida percebemos que isso não era necessário uma vez que não há, no nosso embasamento matemático, a necessidade de se trabalhar com uma terceira dimensão espacial. Uma versão inicial

foi feita com as exatas mesmas funcionalidades do código apresentado acima mas que ainda não mostra perfeitamente a interação em 3D:

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import tkinter as tk
from tkinter import simpledialog, messagebox
import math

# Função para solicitar os valores das órbitas
def solicitar_valores_orbitas():
    root = tk.Tk()
    root.withdraw()  # Esconder a janela principal
    a1 = simpledialog.askfloat("Valor da órbita inicial", "Insira o raio da órbita inicial em km:")
    a2 = simpledialog.askfloat("Valor da órbita final", "Insira o raio da órbita final em km:")
    velocidade_inicial = simpledialog.askfloat("Velocidade inicial", "Insira a velocidade inicial do veículo (em m/s):")
    tempo_inicial = simpledialog.askfloat("Tempo necessário", "Insira o tempo necessário para o satélite completar uma volta completa ao redor da Terra (em segundos):")

    modulo1 = math.sqrt(2 * (a2 / a1)) / ((a2 / a1) + 1)) - 1
    deltav1 = velocidade_inicial * modulo1

    modulo2 = 1 - math.sqrt(2 / ((a2 / a1) + 1))
    deltav2 = velocidade_inicial * modulo2 * math.sqrt(a1 / a2)

    tempo_total = 0.5 * ((1 + (a2 / a1)) / 2) ** (3 / 2) * tempo_inicial

    root.destroy()  # Fechar a janelaferra

    messagebox.showinfo("Resultados", "Delta V1 = {:.2f}\nDelta V2 = {:.2f}\nTempo Total = {:.2f} segundos".format(deltav1, deltav2, tempo_total))

    return a1, a2

# Criação da janela para solicitar os valores das órbitas
def criar_janela():
    # Função para gerar a figura quando o botão for clicado
    def gerar_figura():
        a1, a2 = solicitar_valores_orbitas()
        if a1 is not None and a2 is not None:
            desenhar_figura(a1, a2)

    # Função para fechar a janela
    def sair():
        janela.destroy()

    # Criação da janela
    janela = tk.Tk()
    janela.title("Solicitar Valores de Órbitas")

    # Label e botão
    label = tk.Label(janela, text="Clique no botão para inserir os valores das órbitas.")
    label.pack(padx=10, pady=10)

    botao_inserir = tk.Button(janela, text="Inserir Dados", command=gerar_figura)
    botao_inserir.pack(padx=10, pady=5)

    botao_sair = tk.Button(janela, text="Sair", command=sair)
    botao_sair.pack(padx=10, pady=5)

    janela.mainloop()

# Desenhar a figura com os valores das órbitas inseridos
def desenhar_figura(a1, a2):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Função de posição radial em coordenadas polares
    def r(t, a, e):
        return a * (1 - e**2) / (1 + e * np.cos(t))

    # Raio da órbita inicial e final
    r1 = a1
    r2 = a2

    # Órbita de transferência
    aTransfer = (r1 + r2) / 2

    # Excentricidades
    e1 = 0  # Órbita inicial circular
    eTransfer = (r2 - r1) / (r2 + r1)  # Órbita de transferência elíptica
    e2 = 0  # Órbita final circular

    # Trajetórias
    theta_transfer = np.linspace(0, np.pi, 100)
    r_transfer = r(theta_transfer, aTransfer, eTransfer)

    theta_final = np.linspace(0, 2 * np.pi, 200)
    r_final = r(theta_final, r2, e2)

    # Plot das órbitas
    ax.plot(r1 * np.cos(theta_final), r1 * np.sin(theta_final), 0, color='blue', label='Órbita Inicial')
    ax.plot(r_transfer * np.cos(theta_transfer), r_transfer * np.sin(theta_transfer), 0, color='orange', label='Órbita de Transferência')
    ax.plot(r2 * np.cos(theta_final), r2 * np.sin(theta_final), 0, color='red', label='Órbita Final')

    # Adiciona um ponto azul no centro para representar o planeta
    ax.scatter(0, 0, 0, color='blue', label='Planeta (Terra)')

    # Configurações adicionais
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Manobra de Hohmann em 3D')
    ax.legend()

    plt.show()

# Chamada para solicitar os valores das órbitas
criar_janela()
```

Imagem 04: Código para Modelo Tridimensional

### 3.5. MÚLTIPLAS TROCAS

Durante os estudos, percebemos que o maior desafio na execução de uma manobra orbital é a questão energética. Na prática, para economizar o máximo de

combustível possível, diversas manobras são realizadas, mesmo que isso implique em um tempo maior de execução. Em resposta a essa necessidade, o código descrito no item 4.3 foi aprimorado para suportar a simulação de várias manobras intermediárias. Para isso, foi introduzida uma nova interface em que o usuário é solicitado a inserir o número de manobras que deseja calcular, além de fornecer o raio da órbita inicial em quilômetros, o raio da órbita final em quilômetros, a velocidade inicial do veículo e o tempo necessário para o satélite completar uma volta ao redor da Terra. A seguir, apresentamos o código em Python que simula múltiplas trocas de órbita do veículo. Esta pode ser considerada a versão final do trabalho.

```python
import numpy as np
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import simpledialog, messagebox
import math

# Função para solicitar os valores iniciais
def solicitar_valores_iniciais():
    root = tk.Tk()
    root.withdraw()  # Esconder a janela principal
    num_manobras = simpledialog.askinteger("Número de Manobras", "Insira o número de manobras que deseja calcular:")
    a_inicial = simpledialog.askfloat("Órbita Inicial", "Insira o raio inicial do veículo em km:")
    a_final = simpledialog.askfloat("Órbita Final", "Insira o raio final do veículo em km:")
    velocidade_inicial = simpledialog.askfloat("Velocidade Inicial", "Insira a velocidade inicial do veículo (em m/s):")
    tempo_inicial = simpledialog.askfloat("Tempo Inicial", "Insira o tempo necessário para o satélite completar uma volta completa ao redor da Terra (em segundos):")
    root.destroy()  # Fechar a janela
    return num_manobras, a_inicial, a_final, velocidade_inicial, tempo_inicial

# Função para calcular os parâmetros de cada manobra
def calcular_manobras(num_manobras, a_inicial, a_final, velocidade_inicial, tempo_inicial):
    a_list = np.linspace(a_inicial, a_final, num_manobras + 1)
    orbitas = []
    for i in range(num_manobras):
        a1 = a_list[i]
        a2 = a_list[i + 1]

        modulo1 = math.sqrt((2 * (a2 / a1)) / ((a2 / a1) + 1)) - 1
        deltav1 = velocidade_inicial * modulo1

        modulo2 = 1 - math.sqrt(2 / ((a2 / a1) + 1))
        deltav2 = velocidade_inicial * modulo2 * math.sqrt(a1 / a2)

        tempo_total = 0.5 * ((1 + (a2 / a1)) / 2) ** (3 / 2) * tempo_inicial

        orbitas.append((a1, a2, deltav1, deltav2, tempo_total))

        # Exibir os resultados em uma caixa de diálogo
        messagebox.showinfo(
            f"Resultados da Manobra {i + 1}",
            f"Órbita Inicial: {a1:.2f} km\nÓrbita Final: {a2:.2f} km\nDelta V1: {deltav1:.2f} m/s\nDelta V2: {deltav2:.2f} m/s\nTempo Total: {tempo_total:.2f} segundos"
        )

    return orbitas

# Criação da janela para solicitar os valores das órbitas
def criar_janela():
    # Função para gerar a figura quando o botão for clicado
    def gerar_figura():
        num_manobras, a_inicial, a_final, velocidade_inicial, tempo_inicial = solicitar_valores_iniciais()
        orbitas = calcular_manobras(num_manobras, a_inicial, a_final, velocidade_inicial, tempo_inicial)
        desenhar_figura(orbitas)

    # Função para fechar a janela
    def sair():
        janela.destroy()

    # Criação da janela
    janela = tk.Tk()
    janela.title("Solicitar Valores de Órbitas")

    # Label e botão
    label = tk.Label(janela, text="Clique no botão para inserir os valores das órbitas.")
    label.pack(padx=10, pady=10)

    botao_inserir = tk.Button(janela, text="Inserir Dados", command=gerar_figura)
    botao_inserir.pack(padx=10, pady=5)

    botao_sair = tk.Button(janela, text="Sair", command=sair)
    botao_sair.pack(padx=10, pady=5)

    janela.mainloop()

# Desenhar a figura com os valores das órbitas inseridos
def desenhar_figura(orbitas):
    fig, ax = plt.subplots(subplot_kw={'projection': 'polar'}, figsize=(8, 8))
    fig.patch.set_facecolor('black')  # Define a cor de fundo da figura
    ax.set_facecolor('black')  # Define a cor de fundo dos eixos
    ax.set_ylim(0, max([a2 for a1, a2, deltav1, deltav2, tempo_total in orbitas]))

    # Define a cor dos rótulos e dos ticks
    ax.tick_params(colors='white')  # Define a cor dos ticks
    ax.yaxis.label.set_color('white')  # Define a cor do rótulo do eixo y
    ax.xaxis.label.set_color('white')  # Define a cor do rótulo do eixo x
    ax.title.set_color('white')  # Define a cor do título

    # Adiciona o título com tamanho maior
    ax.set_title("Múltiplas Trocas", fontsize=20)

    for i, (a1, a2, deltav1, deltav2, tempo_total) in enumerate(orbitas):
        r1 = a1
        r2 = a2
        aTransfer = (r1 + r2) / 2

        # Função de posição radial em coordenadas polares
        def r(t, a, e):
            return a * (1 - e**2) / (1 + e * np.cos(t))

        e1 = 0  # Órbita inicial circular
        eTransfer = (r2 - r1) / (r2 + r1)  # Órbita de transferência elíptica
        e2 = 0  # Órbita final circular

        theta_transfer = np.linspace(0, np.pi, 100)
        r_transfer = r(theta_transfer, aTransfer, eTransfer)

        theta_final = np.linspace(0, 2 * np.pi, 200)
        r_final = r(theta_final, r2, e2)

        # Trajetórias
        if i == 0:
            ax.plot(np.linspace(0, 2 * np.pi, 100), [r1] * 100, color='blue', linestyle='--', label='Órbita inicial')  # Órbita inicial circular
        else:
            ax.plot(np.linspace(0, 2 * np.pi, 100), [r1] * 100, color='white', linestyle='--')  # Outras órbitas iniciais

        ax.plot(theta_transfer, r_transfer, color='orange', label='Órbita intermediária' if i == 0 else "")  # Trajetória de transferência
        ax.plot(np.linspace(0, 2 * np.pi, 100), [r2] * 100, color='red', linestyle='--', label='Órbita final' if i == 0 else "")  # Órbita final circular

    # Adiciona um ponto azul no centro para representar o planeta
    ax.plot(0, 0, 'bo', label='Planeta (Terra)')
    ax.legend(loc='lower left')  # Define a posição da legenda
    plt.show()

# Chamada para solicitar os valores das órbitas
criar_janela()
```

Imagem 05: Código para Múltiplas Trocas

## 4. RESULTADOS E DISCUSSÃO - IMAGENS GERADAS

Para exemplificar os resultados gerados pelos códigos do item 3, foram realizadas três simulações distintas. A primeira simulação analisou a transição entre uma órbita baixa e uma órbita média, correspondente a altitudes de 1000 km a 10000 km. A segunda simulação abordou a transição entre uma órbita média e uma órbita geoestacionária, variando entre 10000 km e 36000 km. Por fim, a terceira simulação examinou a transição direta entre uma órbita baixa e uma órbita geoestacionária, cobrindo altitudes de 1000 km a 36000 km.

### 4.1. PROTÓTIPO

Como descrito no item 3.1, o objetivo do protótipo é simplesmente perguntar ao usuário parâmetros sobre a manobra, automatizar as contas e exibir o resultado. Por isso não existem imagens das três simulações propostas, apenas a captura de tela do terminal. Nota-se que quando o usuário respondeu às perguntas propostas o script realizou os códigos apresentados no item 2.5.

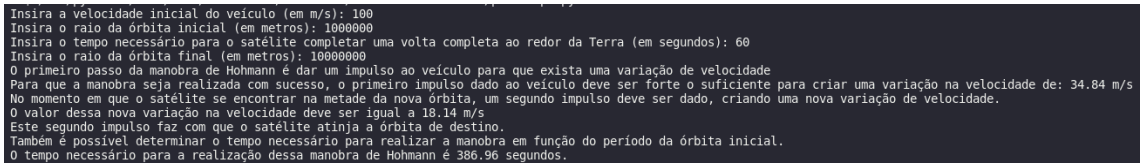



Imagem 06: Resultado do Protótipo

### 4.2. ÓRBITAS EM 2D - COORDENADAS CARTESIANAS

Neste ponto do trabalho a interação com o usuário ainda não havia sido desenvolvida. Por isso o código é capaz de gerar apenas uma simulação padrão. Só é possível alterar os parâmetros dessa simulação fazendo alterações no código. Na imagem tem-se as duas órbitas circulares em preto conectadas pela órbita intermediária em azul.

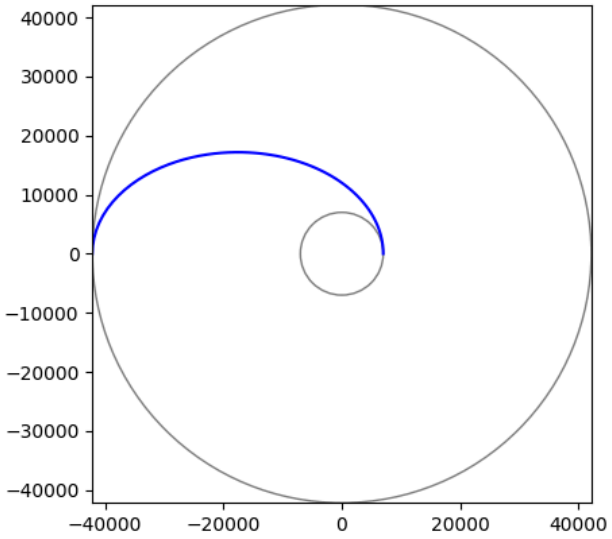



Imagem 07: Resultado do Modelo Cartesiano

### 4.3. ÓRBITAS EM 2D - COORDENADAS POLARES

Com a construção de uma simulação em coordenadas polares fica nítida a diferença entre a representação em cada tipo de coordenada. Por questões de apresentação a nível gráfico e facilidade nas operações matemáticas, apresentar esse tipo de sistema em coordenadas polares se mostra a melhor opção. Aqui já se tem mais claro as distâncias e raios das órbitas. A órbita azul no centro das figuras representa a trajetória inicial descrita pelo corpo e, em vermelho, temos representada a órbita alvo da manobra. A órbita intermediária que conecta as duas está expressa em amarelo.

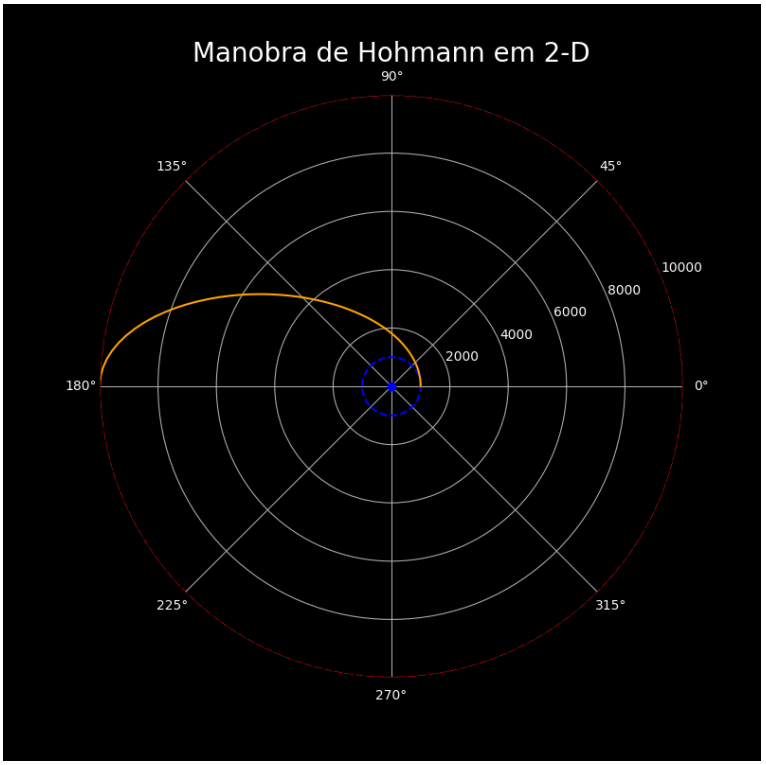



Imagem 08: Coordenadas Polares - Baixa para Média

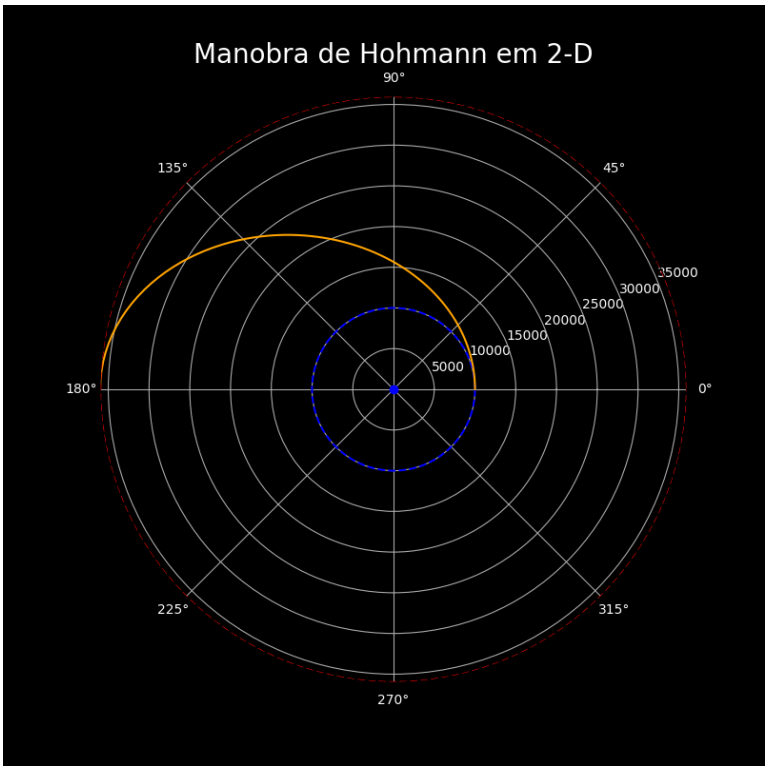



Imagem 09: Coordenadas Polares - Média para Geoestacionária

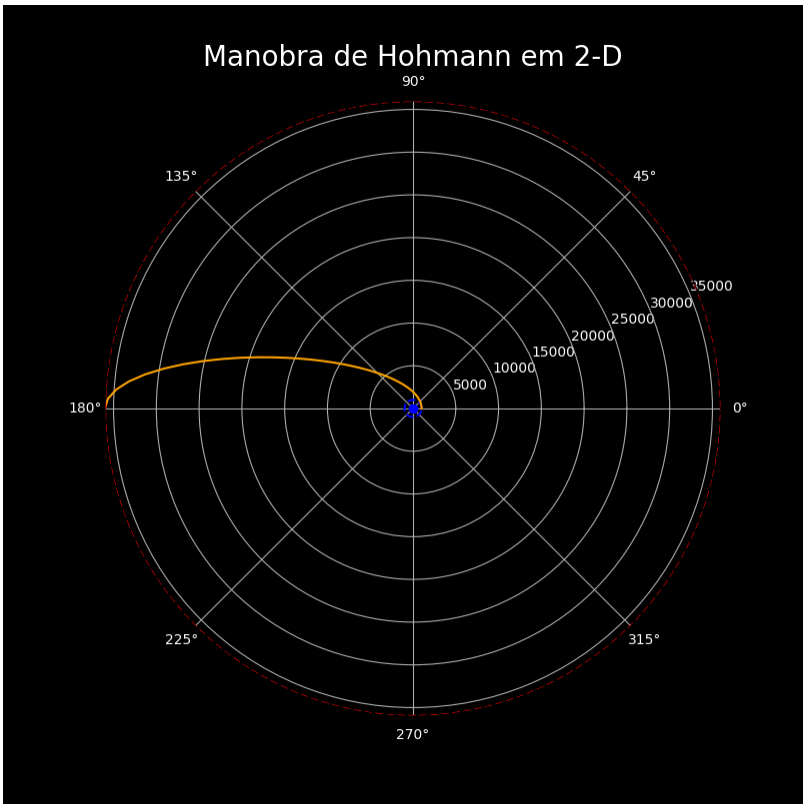



Imagem 10: Coordenadas Polares - Baixa para Geoestacionária

### 4.4. ÓRBITAS EM 3D

Esse código permite que o usuário rotacione a ilustração, entretanto, não é possível representar esse recurso em um artigo. Uma evolução proposta nesse ponto do trabalho foi a adição de uma legenda explicando os elementos da imagem. A representação tridimensional não foi realizada com êxito, já que o programa não leva em consideração variáveis intrínsecas a essa dimensão. O resultado gerado permite que a rotação bidimensional seja visualizada de diversos ângulos.

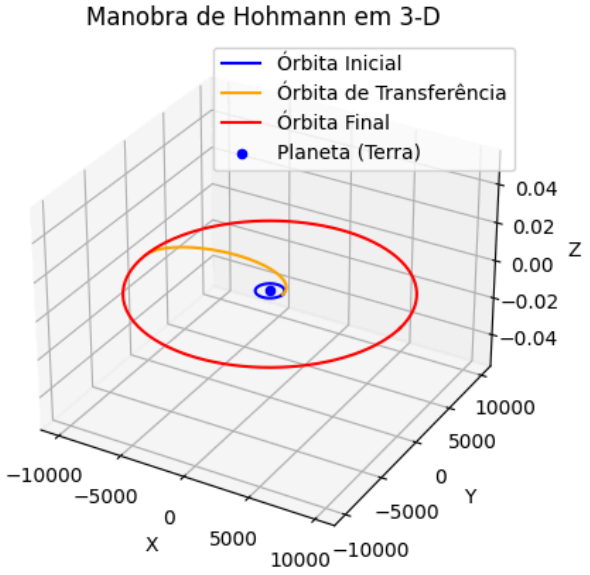



Imagem 11: Representação 3D - Baixa para Média

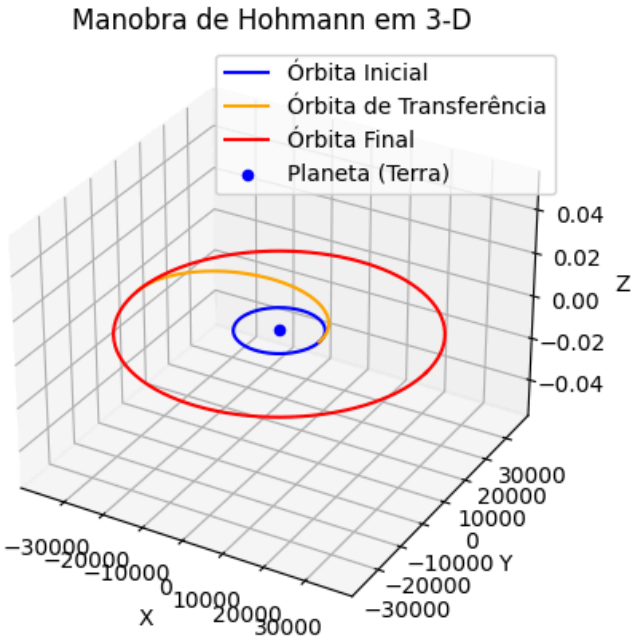



Imagem 12: Representação 3D - Média para Geoestacionária

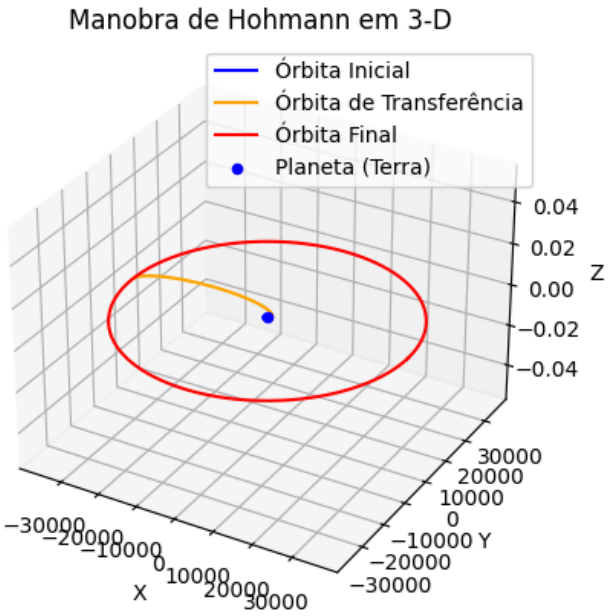



Imagem 13:Representação 3D - Baixa para Geoestacionária

### 4.5. MÚLTIPLAS TROCAS

Estas representações permitem que o usuário visualize múltiplas manobras em uma mesma imagem. As figuras contam com os elementos descritos acima, além de uma órbita branca representando as interações realizadas. Além das perguntas feitas ao usuário, necessárias para a realização dos cálculos, essa versão também solicita que o usuário insira o número de interações intermediárias a serem realizadas. Para todas as imagens geradas com esse código foram selecionadas duas manobras por interação.

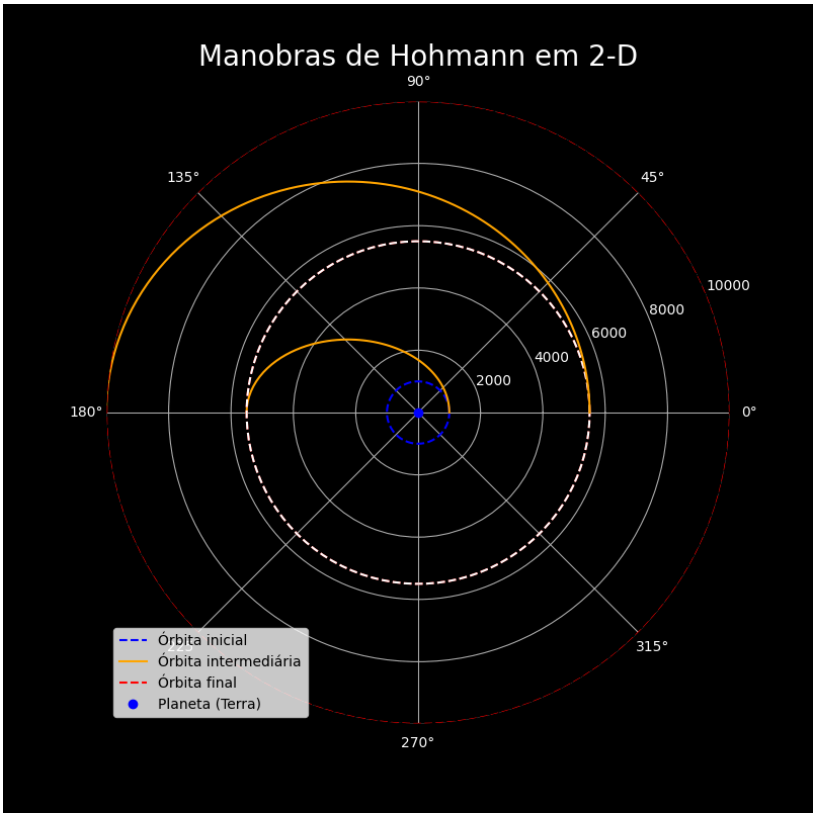



Imagem 14:Múltiplas Trocas - Baixa para Média

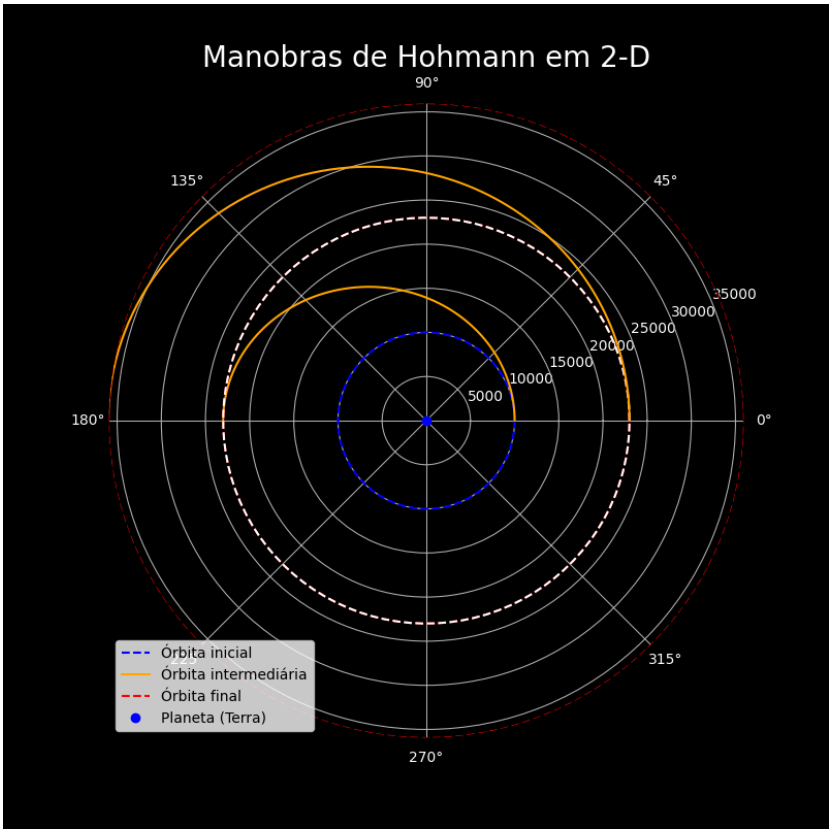



Imagem 15:Múltiplas Trocas - Média para Geoestacionária

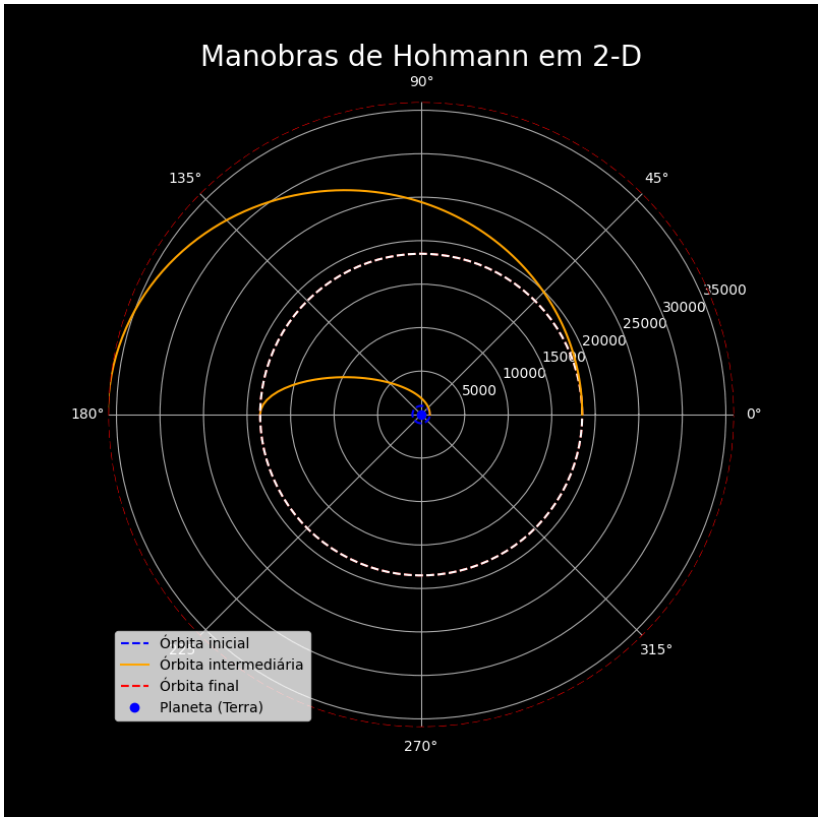



Imagem 16:Múltiplas Trocas - Baixa para Geoestacionária

### 4.6. RESULTADOS MATEMÁTICOS

As simulações geradas nos itens 4.3 e 4.4 apresentam o resultado dos cálculos matemáticos descritos da seguinte forma:

Para a realização desses cálculos será considerado um período de 3600 segundos e uma velocidade de 3000m/s.

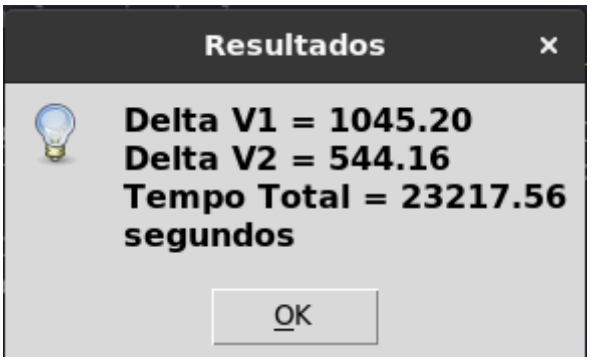



Imagem 17:Resultado - Baixa para Média

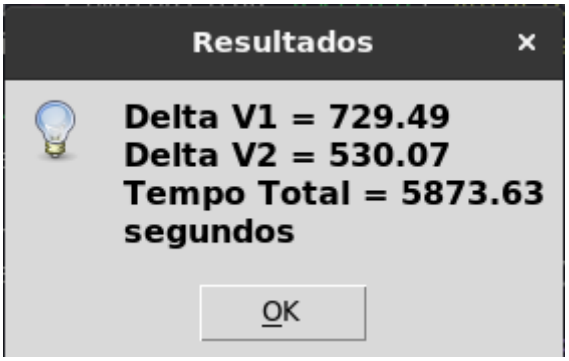



Imagem 18:Resultado - Média para Geoestacionária

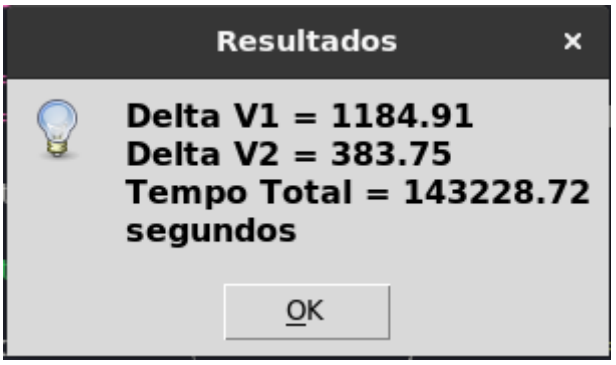



Imagem 19:Múltiplas Trocas - Baixa para Geoestacionária

Uma vez que o item 4.5 apresenta múltiplas interações, mostra-se necessária a existência de mais de uma caixa de resultado, uma para cada manobra.

## 5. CONSIDERAÇÕES FINAIS

Ao longo do projeto, foram desenvolvidas diversas simulações computacionais bem-sucedidas da manobra de Hohmann. Para tal, foram elaborados cinco códigos em Python, cada um projetado para modelar diferentes aspectos da manobra. Após a conclusão de cada código, a equipe realizou e armazenou as simulações correspondentes, assegurando a precisão e a validade dos resultados obtidos.

Foi possível demonstrar nas simulações a manobra sendo executada nas principais órbitas usadas na realidade aeroespacial. Em particular, o código que retrata múltiplas trocas orbitais mostrou-se próximo das práticas reais da indústria, oferecendo uma representação fiel das estratégias utilizadas para otimizar o uso de combustível em detrimento do tempo durante a execução dessas manobras.

O estudo da mecânica Lagrangiana e Hamiltoniana também foi realizado pois proporciona uma base teórica robusta para compreender as trocas de órbitas de satélites, oferecendo uma visão aprofundada das forças e energias envolvidas no processo. Ao utilizar essas abordagens, é possível modelar as trajetórias com maior precisão e prever o comportamento dinâmico dos satélites sob a influência de campos gravitacionais e outros fatores perturbadores. A aplicação desses princípios na análise de manobras orbitais, como a de Hohmann, permite não apenas otimizar o uso de combustível e tempo, mas também explorar soluções inovadoras para desafios complexos da engenharia espacial. Assim, a mecânica Lagrangiana e Hamiltoniana se revelam ferramentas essenciais para o desenvolvimento e a execução de estratégias eficientes de transferência orbital, contribuindo significativamente para a evolução das tecnologias aeroespaciais. A continuidade do projeto pode ser ampliada ao considerar as mecânicas Lagrangiana e Hamiltoniana como perspectivas futuras.

Desta forma, o trabalho realizado cumpriu seu objetivo central, fornecendo uma ferramenta robusta para a simulação da manobra de Hohmann e contribuindo para o estudo das dinâmicas orbitais.

# REFERÊNCIAS

[1] LAWDEN, D. Optical transfers between coplanar elliptical orbits. Journal of Gui-dence Control Dynamics, v.15, p. 788-791, jun. 1992.

[2] MORAES B. S. Estudo de algumas aquisições orbitais usando um propulsor a plasma do tipo hall com ímãs permanentes. 2008. 73 f. Dissertação (Mestrado em Física)-Universidade de Brasília, Brasília, 2008.

[3] TAYLOR J. R. Mecânica Clássica. Porto Alegre, Brasil: Bookman Editora, 2013. Edição em Português.

# APÊNDICE A - MECÂNICA LAGRANGIANA

A mecânica Lagrangiana é uma formulação da mecânica clássica baseada no princípio da menor ação, onde as equações de movimento são derivadas a partir de uma função chamada Lagrangiana, que geralmente é expressa como a diferença entre a energia cinética $T$ e a energia potencial $V$ do sistema: $L = T - V$. Ao invés de lidar diretamente com as forças, como na abordagem Newtoniana, a mecânica Lagrangiana descreve o movimento dos sistemas físicos utilizando coordenadas generalizadas e suas velocidades associadas. As equações de movimento são obtidas aplicando as equações de Euler-Lagrange, que derivam do princípio de Hamilton, afirmando que a trajetória real seguida por um sistema entre dois instantes é aquela que minimiza a ação $S$, definida como a integral da Lagrangiana ao longo do tempo. Esta formulação é poderosa para resolver problemas complexos, especialmente em sistemas com restrições ou em diferentes sistemas de coordenadas [3].

## A.1. DEMONSTRAÇÃO DA EQUAÇÃO DE EULER-LAGRANGE

A equação de Euler-Lagrange pode ser derivada a partir do princípio da ação mínima, que afirma que a trajetória que um sistema físico segue entre dois instantes de tempo $t_1$ e $t_2$ é aquela que minimiza a ação $S$, definida como:

$$S[q(t)] = \int_{t_1}^{t_2} L\left(q,\ \dot{q},\ t\right)dt,$$

onde:

- $q(t)$ são as coordenadas generalizadas do sistema,
- $\dot{q}(t) = \frac{dq}{dt}$ são as velocidades generalizadas,
- $t$ é o tempo.

### A.1.1 VARIAÇÃO DA AÇÃO

Considerando uma pequena variação $\delta q(t)$ na trajetória $q(t)$, a variação correspondente na ação é:

$$\delta S = S[q(t) + \delta q(t)] - S[q(t)].$$

Substituindo a definição de $S$:

$$\delta S = \int_{t_1}^{t_2} L\left(q + \delta q, \dot{q} + \delta\dot{q}, t\right)dt - \int_{t_1}^{t_2} L\left(q, \dot{q}, t\right)dt.$$

Podemos reescrever isso como:

$$\delta S = \int_{t_1}^{t_2} \left[L\left(q + \delta q, \dot{q} + \delta\dot{q}, t\right) - L\left(q, \dot{q}, t\right)\right]dt.$$

### A.1.2. EXPANSÃO DE TAYLOR DA LAGRANGIANA

Expandindo $L\left(q + \delta q, \dot{q} + \delta\dot{q}, t\right)$ em série de Taylor em torno de $q$ e $\dot{q}$, mantendo apenas os termos de primeira ordem:

$$L\left(q + \delta q, \dot{q} + \delta\dot{q}, t\right) = L\left(q, \dot{q}, t\right) + \frac{dL}{dq}\delta q + \frac{dL}{d\dot{q}}\delta\dot{q}$$

Desprezando os termos de ordem superior, a variação da ação $\delta S$ torna-se:

$$\delta S = \int_{t_1}^{t_2} \left(\frac{dL}{dq}\delta q + \frac{dL}{d\dot{q}}\delta\dot{q}\right)dt.$$

### INTEGRAÇÃO POR PARTES

O termo $\frac{dL}{d\dot{q}}\delta\dot{q}$ contém uma derivada de $q$ para simplificar fazemos uma integração por partes, onde $u = \frac{dL}{d\dot{q}}$ e $dv = \delta\dot{q}\,dt$:

$$\int_{t_1}^{t_2} \frac{dL}{d\dot{q}}\delta\dot{q}\,dt = \left[\frac{dL}{d\dot{q}}\delta q\right]_{t_1}^{t_2} - \int_{t_1}^{t_2} \frac{d}{dt}\left(\frac{dL}{d\dot{q}}\right)\delta q\,dt.$$

O primeiro termo $\left[\frac{dL}{d\dot{q}}\delta q\right]_{t_1}^{t_2}$ é uma integral de contorno e desaparece porque assumimos que a variação $\delta q(t)$ é nula nos extremos $t_1$ e $t_2$.

$$\delta q\left(t_1\right) = \delta q\left(t_2\right) = 0$$

Assim, a variação da ação reduz-se a:

$$\delta S = \int_{t_1}^{t_2} \left( \frac{dL}{dq} - \frac{d}{dt} \frac{dL}{dq^{\cdot}} \right) \delta q \, dt.$$

Para que a ação seja um extremo (mínima ou estacionária), a variação $\delta S$ deve ser zero para qualquer variação $\delta q(t)$. Como $\delta q(t)$ é arbitrária, isso implica que a expressão dentro do parênteses deve ser zero:

$$\frac{dL}{dq} - \frac{d}{dt} \frac{dL}{dq^{\cdot}} = 0$$

Esta é a equação de **Euler-Lagrange**:

$$\frac{d}{dt} \left( \frac{dL}{dq^{\cdot}} \right) - \frac{dL}{dq} = 0.$$

Essa equação descreve as equações de movimento do sistema em termos da Lagrangiana $L\left(q, \, q^{\cdot}, \, t\right)$.

## A.2. DETERMINANDO O MENOR CAMINHO ENTRE DOIS PONTOS

A distância diferencial é dada por:

$$ds^2 = dx^2 + dy^2$$

Assim, a distância elementar é:

$$ds = \sqrt{dx^2 + dy^2}$$

Considerando $dy = y'(x)dx$, temos:

$$ds = \sqrt{1 + y'^2} dx$$

O comprimento do caminho é então:

$$L = \int_{x_1}^{x_2} \sqrt{1 + y'^2} dx$$

Onde:

$$\sqrt{1 + y'^2} = f(y, y', x)$$

e:

$$d(y, y', x) = \left(1 + y'^2\right)^{\frac{1}{2}}$$

Aplicando a equação de Euler-Lagrange:

$$\frac{d}{dx}\left(\frac{df}{dy'}\right) - \frac{df}{dy} = 0$$

Onde $\frac{df}{dy} = 0$ e:

$$\frac{df}{dy'} = \frac{y'}{\left(1+y'^2\right)^{\frac{1}{2}}}$$

Derivando em relação a $x$:

$$\frac{d}{dx}\left(\frac{y'}{\left(1+y'^2\right)^{\frac{1}{2}}}\right) = 0$$

Logo, $\frac{y'}{\left(1+y'^2\right)^{\frac{1}{2}}}$ é uma constante, o que leva à conclusão de que a solução é uma reta:

$$y = mx + b$$

### A.3. CÁLCULO DA BRAQUISTÓCRONA

O problema da braquistócrona consiste em encontrar a curva mais rápida para um objeto deslizar sob a influência de um campo gravitacional uniforme, de um ponto A a um ponto B, onde B não está verticalmente abaixo de A. O termo "braquistócrona" vem do grego "brachistos" (o mais curto) e "chronos" (tempo), significando "o tempo mais curto"[3].

O tempo diferencial é dado por:

$$dt = \frac{ds}{v} = \frac{\sqrt{dx^2+dy^2}}{v}$$

Usando a conversão de energia:

$$\frac{1}{2}mv^2 = mgy$$

Ou

$$v = \sqrt{2gy}$$

Portanto:

$$dt = \frac{\sqrt{1+\left(\frac{dx}{dy}\right)^2}}{\sqrt{2gy}}dy$$

O tempo total é:

$$\Delta t = \frac{1}{\sqrt{2g}}\int \frac{\sqrt{1+x'^2}}{\sqrt{y}}dy$$

Para minimizar essa integral, aplicamos a equação de Euler-Lagrange para o integrando:

$$f\left(x,\ \dot{x},\ y\right) = \frac{\sqrt{1+x'^2}}{\sqrt{y}}$$

Derivando:

$$\frac{df}{dx} = 0, \frac{df}{dx'} = \frac{x'}{\sqrt{1+x'^2}\sqrt{y}}$$

e

$$\frac{x'}{\sqrt{1+x'^2}\sqrt{y}} = c$$

Sendo $c$ uma constante:

$$x' = \sqrt{\frac{y}{2a-y}}$$

Para encontrar $x$, resolvemos a integral:

$$x = \int \sqrt{\frac{y}{2a-y}}dy$$

A solução paramétrica em termos de θ é:

$$x(\theta) = a\theta - a\,sen\theta$$

e

$$y(\theta) = a(1 - cos\theta)$$

# APÊNDICE B - MECÂNICA HAMILTONIANA

A mecânica Hamiltoniana, introduzida por William Rowan Hamilton no século XIX, é uma reformulação elegante e poderosa da mecânica clássica. Diferente da mecânica Lagrangiana, que se baseia nas coordenadas generalizadas e suas velocidades associadas, a mecânica Hamiltoniana faz uso das coordenadas generalizadas e dos momentos conjugados, formando um conjunto de equações diferenciais de primeira ordem conhecidas como equações de Hamilton. Essa abordagem não só facilita a análise de sistemas complexos e conservativos, como também estabelece a ponte entre a mecânica clássica e a mecânica quântica, através da analogia entre o formalismo de Hamilton e o formalismo quântico [3].

## B.1. EQUAÇÕES DE HAMILTON PARA SISTEMAS UNIDIMENSIONAIS

Dado um sistema com coordenadas generalizadas $q$ e suas velocidades $q^{\bullet}$, a Lagrangiana $L$ é definida como:

$$L = L\left(q,\ q^{\bullet}\right) - U(q).$$

A Lagrangiana para um sistema conservativo com coordenadas naturais em uma dimensão pode ser escrita na forma geral

$$L = L\left(q,\ q^{\bullet}\right) = T - U = \frac{1}{2}A(q)q^{\bullet 2} - U(q).$$

A função Hamiltoniana $H$ é dada por:

$$H\left(q_i,\ p_i,\ t\right) = \sum_i p_i q_i^{\bullet} - L,$$

Onde os momentos conjugados $p_i$ são dados por:

$$p_i = \frac{dL}{dq_i^{\bullet}}.$$

Neste caso,

$$p = \frac{dL}{dq^{\bullet}} = \frac{dL}{dq^{\bullet}}\left[\frac{1}{2}A(q)q^{\bullet 2} - U(q)\right] = A(q)q^{\bullet}.$$

O que nos leva a

$$H = pq^{\bullet} - L = A(q)q^{\bullet}q^{\bullet} - \left[\frac{1}{2}A(q)q^{\bullet 2} - U(q)\right],$$

Ou seja,

$$H = A(q)q^{\bullet 2} - \frac{1}{2}A(q)q^{\bullet 2} + U(q) = \frac{1}{2}A(q)q^{\bullet 2} + U(q).$$

Então,

$$H = T + U$$

Que é a energia total do sistema. Por outro lado vimos que

$$p = \frac{dL}{dq^{\bullet}} \rightarrow q^{\bullet} = \frac{p}{A(q)} = q^{\bullet}(q, p)$$

Assim,

$$H = pq^{\bullet}(q, p) - L\left(qq^{\bullet}(q, p)\right)$$

Vamos calcular as derivadas de $H = H(q, p)$ utilizando a regra da cadeia, ou seja,

$$\frac{dH}{dq} = p\frac{dq^{\bullet}}{dq} - \left(\frac{dL}{dq} + \frac{dL}{dq^{\bullet}}\frac{dq^{\bullet}}{dq}\right),$$

ou ainda,

$$\frac{dH}{dq} = p\frac{dq^{\bullet}}{dq} - \frac{dL}{dq} - p\frac{dq^{\bullet}}{dq}.$$

O que nos leva a

$$\frac{dH}{dq} = -\frac{dL}{dq}.$$

Lembrando da equação de Euler-Lagrange

$$\frac{d}{dt}\left(\frac{dL}{dq^{\bullet}}\right) = \frac{dL}{dq}$$

Temos que:

$$\frac{dH}{dq} = -\frac{d}{dt}\left(\frac{dL}{dq^{\bullet}}\right) = -\frac{d}{dt}p = -p^{\bullet}.$$

Que é exatamente a primeira das duas equações de movimento de Hamilton.

$$p^{\bullet} - \frac{dH}{dq}$$

Para obtermos a segunda equação vamos tomar a derivada de $H$ em relação a $p$ . Note novamente que $H = pq^{\bullet}(q, p) - L\left(q, q^{\bullet}(q, p)\right)$.

$$\frac{dH}{dp} = \frac{d}{dp}\left(pq^{\bullet}(q, p)\right) - \frac{dL}{dq^{\bullet}}\frac{dq^{\bullet}}{dp}.$$

Tomando a derivada do primeiro termo e lembrando que $\frac{dL}{dq^{\bullet}} = p$, obtém-se

$$\frac{dH}{dp} = \frac{dp}{dp}q^{\bullet} + p\frac{dq^{\bullet}}{dp} - p\frac{dq^{\bullet}}{dp}.$$

Desta forma, escreve-se a segunda das equações de Hamilton:

$$q^{\bullet} = \frac{dH}{dp}.$$

# APÊNDICE C - ESTUDO SOBRE ELIPSES

## C.1. ELIPSES EM COORDENADAS CARTESIANAS

- Fixados os focos $F_1$ e $F_2$, o conjunto dos pontos $P$ tais que $PF_1 + PF_2 = 2a$ (constante) define uma elipse.
- Distância focal: $F_1F_2 = 2c$
- Excentricidade: $e = \frac{c}{a}$, onde $0 < e < 1$
- A relação entre os eixos da elipse é dada por $a^2 = b^2 + c^2$

A equação da elipse é:

$$\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} = 1$$

Onde $(x_0,\ y_0)$ é o centro da elipse.

Ao aumentar a excentricidade $e$, a distância entre os focos da elipse aumenta, tornando-a mais alongada. Por outro lado, ao diminuir $e$, a distância entre os focos diminui, tornando a elipse mais semelhante a uma circunferência.

## C.2. ELIPSES EM COORDENADAS POLARES

Para uma elipse em coordenadas polares, temos a relação:

$$d(P,\ F) = e \cdot d(P,\ S)$$

A equação da elipse é dada por:

$$r = e(d - x)$$

Ou de forma expandida

$$r = ed - er \cdot cos\theta$$

Que pode ser reorganizada como

$$r(1 + e \cdot cos\theta) = de$$

Portanto,

$$r(\theta) = \frac{de}{1+e \cdot cos\theta}$$

Onde $e$ deve necessariamente ser menor que 1. Interpretação geométrica de $d$: Para $\theta = \frac{\pi}{2}$ ou $\theta = 3\pi$, temos $r(\theta) = de$

$2de$ é a menor corda que passa pelo foco da elipse, chamada de **corda focal mínima** ou **latus rectum**.

Determinação do eixo maior ($2a$):

$$2a = r\theta + r\pi$$

Substituindo:

$$2a = \frac{de}{1+e \cdot cos\theta} + \frac{de}{1-e \cdot cos\pi}$$

Que resulta em:

$$2a = \frac{de-de^2+de+de^2}{1-e^2}$$

Portanto:

$$a = \frac{de}{1-e^2}$$

Onde:

$$1 - e^2 = \frac{b^2}{a^2}$$

E:

$$a = de\frac{a^2}{b^2}$$

Resultando em:

$$\frac{b^2}{a} = de$$

Nas órbitas temos:

$$r(\theta) = \frac{eL^2/cm^2}{1+e \cdot cos\theta}$$

Onde $e = \frac{c}{GM}$ e $d = \frac{L^2}{cm^2}$.