UNIVERSIDADE FEDERAL DO ABC

São Bernardo do Campo

Relatório 2 - Localização de Um Ponto e *Ground Track*

Disciplina: Laboratório de Guiagem, Navegação e Controle
Docente: Prof. Dr. Leandro Baroni

Arthur Sarri Binelli RA: 21043816
João Victor Pinho Pizoni RA: 11201920704
Rebeca Sales Ribeiro Alves RA: 112019207408

8 de julho de 2025

Sumário

Lista de Figuras

Capítulo 1

Introdução

Para determinar com precisão a localização de um ponto na superfície terrestre, é fundamental compreender os sistemas de coordenadas e o tempo envolvidos. Essa capacidade é essencial não apenas para fins científicos e geodésicos, mas também para a navegação, observação da Terra, telecomunicações e uma ampla gama de aplicações espaciais. A importância dessa habilidade reside na possibilidade de relacionar qualquer ponto na Terra com uma posição específica em um sistema de referência, permitindo que um veículo espacial seja monitorado ou posicionado de forma precisa a partir de dados orbitais.

Nesse contexto, a projeção da trajetória de um veículo espacial sobre a superfície da Terra, quando representada em um plano bidimensional, é chamada de ground track. Embora frequentemente confundidos, os conceitos de órbita e ground track são distintos. A órbita refere-se ao movimento tridimensional do veículo espacial ao redor de um corpo central, como a Terra. Já o ground track corresponde à projeção bidimensional dessa trajetória sobre a superfície do corpo central, permitindo a visualização do caminho que o satélite "desenha"sobre o planeta à medida que se desloca[1].

O estudo do ground track é de extrema relevância no planejamento e na operação de missões espaciais, pois possibilita a determinação de quando e onde um satélite será visível a partir da superfície terrestre. Essa análise depende diretamente dos parâmetros orbitais do veículo espacial, os quais variam conforme o tipo de missão, os objetivos operacionais e o tipo de órbita selecionada. Assim, ao conhecer a órbita, é possível prever o ground track, que por sua vez permite determinar quais regiões da Terra serão sobrevoadas e em que momentos ocorrerá essa visualização.

Na prática, a escolha da órbita de um veículo espacial influencia diretamente o seu ground track e, consequentemente, sua eficiência operacional. Por exemplo, uma órbita geoestacionária mantém o satélite fixo em relação a um ponto na superfície da Terra, eliminando a necessidade de correções contínuas de apontamento para antenas de comunicação. Já outros tipos de órbitas, como as órbitas polares ou heliossíncronas, são utilizadas para cobrir amplamente a superfície terrestre com diferentes ground tracks ao longo do tempo, conforme será discutido na Seção 2.3.

1.1 Objetivos

Este relatório tem como objetivos principais a determinação da localização de 3 pontos em coordenadas geocêntricas inerciais, a partir de coordenadas cartesianas geográficas, e a obtenção do ground track de um veículo espacial em órbita terrestre, com base em condições iniciais de posição e velocidade. Adicionalmente, busca-se analisar a influência dos parâmetros orbitais na configuração do ground track, de modo a compreender sua contribuição para a trajetória projetada do corpo orbital.

Seguem abaixo os pontos analisados.

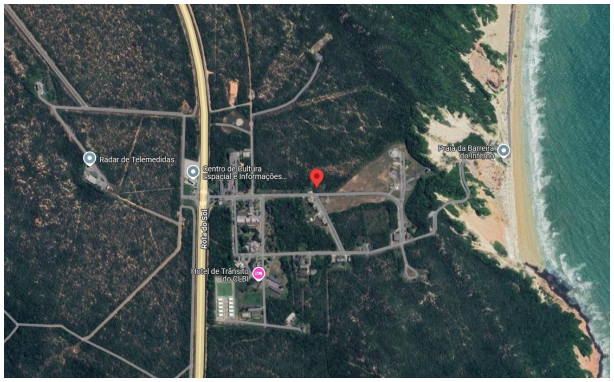

Figura 1.1: Centro de Cultura Espacial e Informações Turísticas

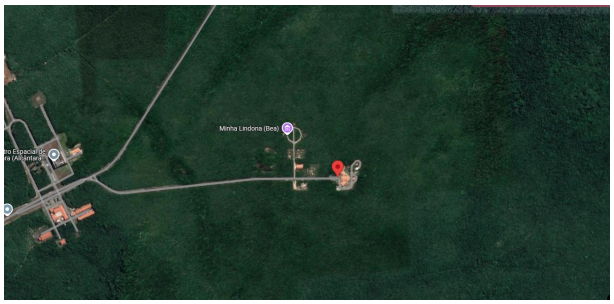

Figura 1.2: Centro de Lançamento de Alcântara (CLA)

Figura 1.3: UFABC Campus SBC

Capítulo 2

Fundamentação Teórica

2.1 Sistemas de Tempo

A definição precisa do tempo é fundamental para a descrição do movimento dos corpos celestes e para a navegação astronômica e espacial. Para esse fim, diversos sistemas de tempo foram desenvolvidos ao longo da história, com base em referências astronômicas observáveis, como o movimento aparente da esfera celeste.

A esfera celeste é uma construção teórica que representa a abóbada celeste como uma esfera de raio arbitrário centrada na Terra, sobre a qual se projetam os astros. A rotação da Terra em torno de seu próprio eixo, no sentido de oeste para leste, causa a impressão de que os astros realizam um movimento de leste para oeste ao longo do dia — fenômeno conhecido como movimento diurno.

Esse movimento aparente permite a definição de escalas de tempo baseadas em fenômenos astronômicos, como o tempo solar verdadeiro e o tempo sideral. Tais sistemas são fundamentais para a determinação da hora local, para o apontamento de telescópios e para cálculos orbitais de precisão [2].

A Figura 2.1 apresenta uma representação esquemática da esfera celeste, ilustrando o horizonte do observador, os polos celestes, o equador celeste e o movimento aparente dos astros.

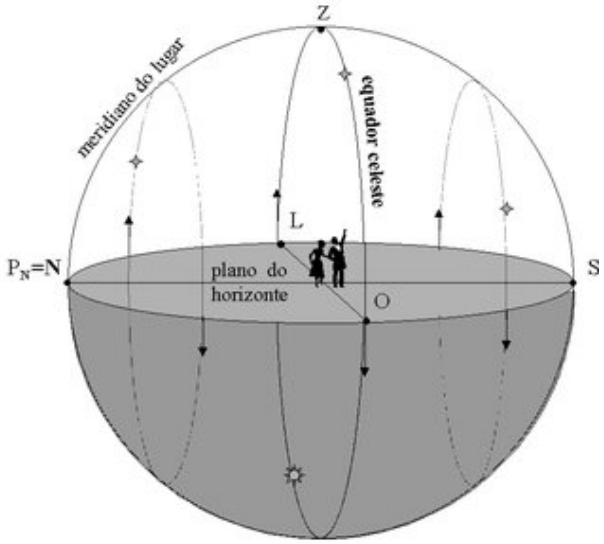

Figura 2.1: Esfera Celeste

2.1.1 Tempo Solar

O tempo solar é definido como a medida do intervalo de tempo baseado no movimento aparente do Sol na esfera celeste. Quando o Sol completa uma volta aparente na esfera celeste, retornando à mesma posição relativa ao observador, tem-se a definição de um dia solar. Em termos práticos, o dia solar corresponde ao tempo que o Sol leva para retornar ao zênite de um determinado ponto na superfície terrestre, perfazendo aproximadamente 24 horas. Assim, cada hora solar equivale a 15° de deslocamento aparente do Sol em relação ao zênite do observador [2].

O tempo solar aparente, calculado com base na posição real do Sol em um dado meridiano, considera a soma de 12h com o ângulo horário solar. No entanto, essa duração não é constante ao longo do ano, devido à excentricidade da órbita terrestre e à inclinação do eixo da Terra em relação ao plano da eclíptica, o que causa variações perceptíveis na velocidade aparente do Sol [2].

Em função dessa variabilidade, foi introduzido o conceito de *tempo solar médio*, que considera um Sol fictício movendo-se uniformemente ao longo do equador celeste. Esse tempo representa uma média do tempo solar aparente ao longo do ano, sendo mais apropriado para aplicações cotidianas e sistemas de medição regulares.

Como o tempo solar médio se refere ao meridiano local — o qual varia de acordo com a longitude geográfica —, sua utilização direta não é prática para sincronização de horários em grandes regiões. Por isso, adota-se o sistema de fusos horários, que divide a Terra em 24 setores de 15° de longitude cada, correspondendo a uma hora por fuso. O meridiano central de cada fuso serve como referência para padronizar o tempo médio solar naquela região.

Os fusos são numerados a partir do Meridiano de Greenwich, que define o tempo uni-

versal coordenado (UTC). Fusos localizados a leste de Greenwich apresentam adiantamento em relação ao UTC, enquanto os fusos a oeste apresentam atraso. A Figura 2.2 ilustra a divisão da Terra em fusos horários e sua relação com o meridiano de referência.

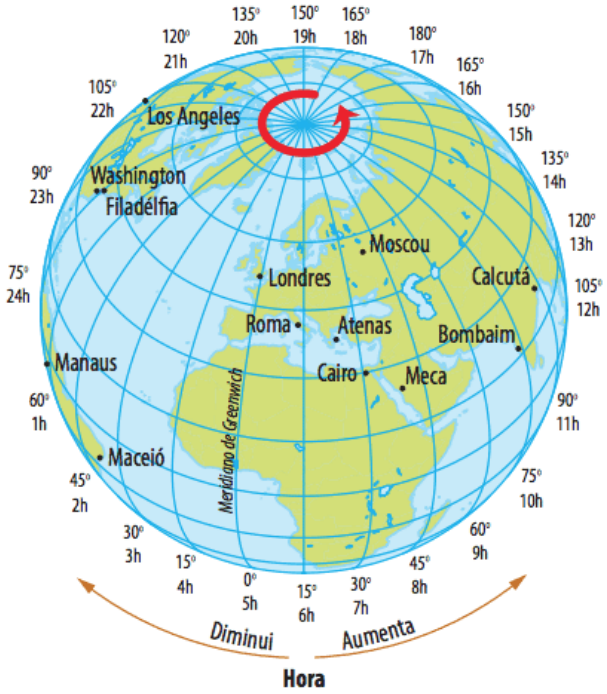

Figura 2.2: Fusos Horários

2.1.2 Tempo Universal

Conforme descrito por Curtis, o Tempo Universal (Universal Time — UT) é estabelecido com base na passagem do Sol pelo Meridiano de Greenwich, localizado na longitude 0°, sendo que esse evento ocorre às 12 horas UT. Esse sistema constitui uma referência fundamental para a medição do tempo em escala global.

A conversão entre a hora local e o Tempo Universal é realizada por meio do ajuste correspondente ao fuso horário da localidade em questão. Para isso, basta considerar que o UT é equivalente à hora local subtraída do valor do fuso horário. Convenciona-se que os fusos situados a leste de Greenwich possuem valor negativo em relação ao UT, enquanto os fusos a oeste apresentam valor positivo.

A relação entre a hora local e o Tempo Universal pode ser determinada a partir da diferença de fuso horário do local considerado. De modo geral, o Tempo Universal é obtido pela subtração do fuso horário da hora local. No sistema de fusos, convenciona-se que regiões localizadas a leste do Meridiano de Greenwich possuem fusos com sinal negativo em relação ao UT, enquanto regiões a oeste apresentam fusos com sinal positivo.

2.1.3 Tempo Sideral de Greenwich

Tempo Sideral de *Greenwich*

O Tempo Sideral de *Greenwich*, também chamado de Ângulo Hora de Greenwich (θ_g), corresponde ao ângulo entre o Meridiano de *Greenwich* e o Equinócio Vernal (γ). Essa medida representa a rotação da Terra em relação às estrelas fixas, e não em relação ao Sol, sendo essencial para aplicações astronômicas e orbitais [**vallado**].

A Terra leva aproximadamente 23h56min para completar uma rotação sideral, o que equivale a cerca de 4 minutos a menos que o dia solar médio. Essa diferença ocorre porque, além de girar em torno de seu próprio eixo, a Terra também se translada ao redor do Sol, exigindo um pequeno deslocamento adicional (1°) para que o Sol retorne à mesma posição no céu em dois dias solares consecutivos.

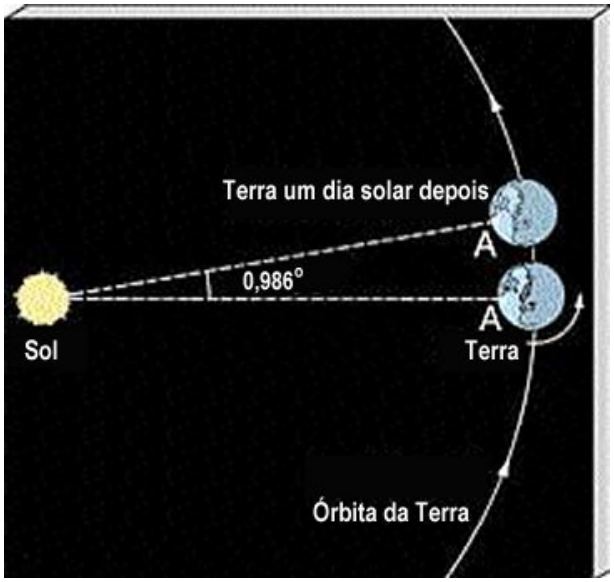

Figura 2.3: Dia Solar

Tempo Sideral Local O Tempo Sideral Local (LST — *Local Sidereal Time*) é obtido a partir do Tempo Sideral de Greenwich somando-se a ele a longitude geográfica λ do observador (positiva para leste, negativa para oeste):

$$\theta = \theta_g + \lambda \tag{2.1}$$

Cálculo do Tempo Sideral de Greenwich O valor do Tempo Sideral de Greenwich às 0h00min UT de uma data qualquer é dado por:

$$\theta_{g0} = 99,6909833 + 36000,7689 \cdot SJ + 0,00038708 \cdot SJ^2 \tag{2.2}$$

em que SJ representa o número de séculos julianos desde 0h UT de 1 de janeiro de 1900, calculado como:

$$SJ = \frac{DJ - 2415020}{36525} \tag{2.3}$$

Aqui, DJ é a Data Juliana correspondente, obtida a partir do calendário gregoriano com a fórmula:

$$DJ = 367y - \text{INT}\left(\frac{7}{4}\left(y + \text{INT}\left(\frac{m+9}{12}\right)\right)\right) + \text{INT}\left(\frac{275m}{9}\right) + d + 1721013.5 \tag{2.4}$$

em que d, m e y são o dia, mês e ano da data considerada.

Correção para horário desejado A partir de θ_{g0}, o valor do tempo sideral de Greenwich para qualquer horário UT é obtido por:

$$\theta_g = \theta_{g0} + (t - t_0) \cdot \dot{\theta} \tag{2.5}$$

ou, alternativamente, com base na hora universal em horas decimais:

$$\theta_g = \theta_{g0} + 360,98564724 \cdot \frac{UT}{24} \tag{2.6}$$

Observações sobre o Dia Juliano O Dia Juliano (DJ) possui diferentes variantes conforme o contexto astronômico adotado. Entre elas, destacam-se:

- Dia Juliano Astronômico Padrão, que inicia às 12h UT de 1 de janeiro de 4713 a.C.;
- Dia Juliano Modificado, cujo ponto inicial é 0h UT de 17 de novembro de 1858;
- Dia Juliano Reduzido e Dia Juliano Truncado, utilizados para simplificação de cálculos computacionais e dados em tempo real.

Cabe destacar que nem sempre está documentada qual versão do DJ é adotada em sistemas nacionais específicos, como no caso do Brasil.

2.2 Sistemas de Coordenadas

A descrição precisa do movimento de um veículo espacial requer a definição adequada do sistema de coordenadas. Diversos sistemas são utilizados, cada um com propósitos específicos relacionados à observação terrestre ou à dinâmica orbital. Nesta seção, são apresentados os sistemas Cartesiano Terrestre, Cartesiano Geográfico, Inercial (ECI), bem como as transformações entre eles.

2.2.1 Sistema Cartesiano Terrestre

Este sistema tem origem no centro de massa da Terra. O eixo O_z aponta para o polo norte geográfico, o eixo O_xt encontra-se na interseção entre o meridiano de Greenwich e o equador, enquanto o eixo O_yt completa o sistema destrógio. A Figura 2.4 apresenta a representação desse sistema.

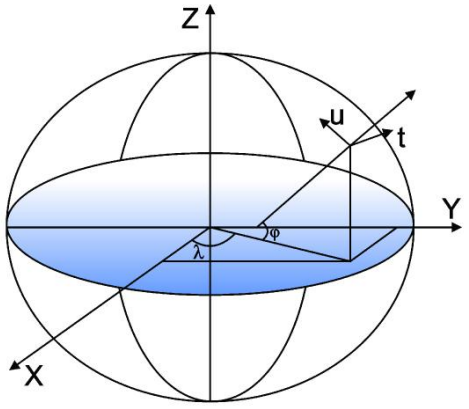

Figura 2.4: Figura de sistema cartesiano

As coordenadas cartesianas (x, y, z) de um ponto podem ser relacionadas aos parâmetros geográficos por:

$$\begin{cases} x = R\cos(\lambda)\cos(\phi') \\ y = R\sin(\lambda)\cos(\phi') \\ z = R\sin(\phi') \end{cases} \tag{2.7}$$

2.2.2 Sistema Cartesiano Geográfico

Utiliza os mesmos eixos do sistema terrestre, mas distingue entre latitude geodésica ϕ e latitude geocêntrica ϕ. A Figura 2.5 ilustra graficamente essa diferença.

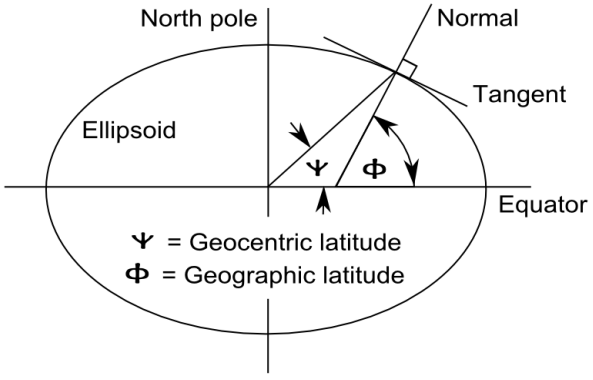

Figura 2.5: Latitude geodésica vs. geocêntrica.

2.2.3 Sistema de Coordenadas Inerciais (ECI)

O sistema ECI (Earth-Centered Inertial) tem origem no centro de massa da Terra. O eixo Ox aponta para o ponto vernal $gamma$, o eixo Oz é paralelo ao eixo de rotação da Terra, e Oy completa o sistema destrógio. Este sistema é fixo em relação às estrelas distantes (com exceção da precessão).

2.2.4 Transformação entre Sistemas de Coordenadas

A transformação entre sistemas de coordenadas é fundamental para aplicações em navegação, georreferenciamento e dinâmica orbital. Em particular, a conversão das coordenadas do Sistema Cartesiano Geográfico para o Sistema Cartesiano Terrestre (ECEF — Earth-Centered, Earth-Fixed) é realizada por meio do conjunto de equações a seguir, conforme proposto por Curtis [**curtis2013**]:

$$\begin{cases} x = \left(\dfrac{a}{\sqrt{1 - e^2 \sin^2(\phi)}} + H \right) \cos(\phi) \cos(\lambda) \\ y = \left(\dfrac{a}{\sqrt{1 - e^2 \sin^2(\phi)}} + H \right) \cos(\phi) \sin(\lambda) \\ z = \left(\dfrac{a(1 - e^2)}{\sqrt{1 - e^2 \sin^2(\phi)}} + H \right) \sin(\phi) \end{cases} \tag{2.8}$$

onde:

- a é o semi-eixo maior do elipsoide de referência (ex.: GRS80 ou WGS84);
- H é a altitude do ponto acima do nível do mar;
- ϕ é a latitude geodésica;
- λ é a longitude;
- e é a excentricidade do elipsoide, definida por:

$$e = \sqrt{\frac{a^2 - b^2}{a^2}} \tag{2.9}$$

com b representando o semi-eixo menor.

Para convenção de sinais:

- A latitude (ϕ) é positiva no hemisfério norte e negativa no hemisfério sul.
- A longitude (λ) é positiva para o leste do Meridiano de Greenwich e negativa para o oeste.

A seguir, a conversão das coordenadas do Sistema Cartesiano Terrestre para o Sistema Inercial (ECI — Earth-Centered Inertial) exige uma rotação em torno do eixo O_z do sistema terrestre, considerando o ângulo horário sideral de Greenwich, denotado por θ_g. Essa rotação alinha o meridiano de Greenwich ao equinócio vernal, baseando-se em um referencial fixo em relação às estrelas distantes.

A transformação é representada por:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} \cos(\theta_g) & \sin(\theta_g) & 0 \\ -\sin(\theta_g) & \cos(\theta_g) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} \tag{2.10}$$

onde o índice t refere-se ao Sistema Cartesiano Terrestre e o índice i ao Sistema de Coordenadas Inerciais.

Essa transformação é essencial para aplicações em rastreamento orbital, análise de órbita e determinação de atitude, pois permite converter medições obtidas em estações terrestres para um sistema fixo no espaço.

2.3 Ground Track

O *ground track* corresponde à projeção da trajetória orbital de um satélite sobre a superfície terrestre. Trata-se da representação bidimensional do caminho percorrido pelo veículo espacial ao redor do planeta, considerando a rotação da Terra. Essa projeção permite visualizar como a órbita se comporta em relação a um plano fixo na superfície, sendo uma ferramenta fundamental para o planejamento e análise de missões espaciais.

Um exemplo clássico de *ground track* é observado em órbitas do tipo Molniya, caracterizadas por alta excentricidade e inclinação elevada. A Figura 2.6 apresenta a projeção dessa órbita sobre o globo terrestre, onde é possível observar a trajetória ondulada típica de veículos em órbitas fortemente inclinadas.

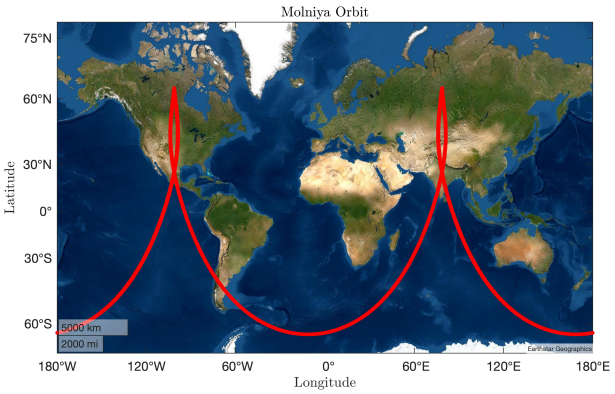

Figura 2.6: Órbita do tipo Molniya

Há uma variedade de formas de *ground track*, diretamente relacionadas ao tipo de órbita:

circulares, elípticas, com inclinações elevadas ou nulas, e com diferentes altitudes. Por exemplo, órbitas com inclinação zero apresentam *ground tracks* horizontais localizados ao longo do equador terrestre. Já órbitas inclinadas, como as polares, resultam em projeções que cobrem amplas faixas longitudinais da Terra, sendo muito utilizadas para sensoriamento remoto, pois permitem cobertura global com o passar do tempo.

Outro caso de interesse é o das órbitas geoestacionárias, que são circulares e possuem inclinação nula com período orbital de 24 horas. Nessas condições, o *ground track* do satélite se reduz a um ponto fixo sobre a superfície terrestre, visto que o veículo orbita a mesma taxa angular da Terra. Essa característica é ideal para satélites de telecomunicação e monitoramento meteorológico, pois permite a permanência contínua sobre uma mesma região do planeta [.

Os parâmetros orbitais influenciam fortemente a forma do *ground track*. O semi-eixo maior, por exemplo, determina o raio da órbita e, consequentemente, a periodicidade com que o satélite sobrevoa os mesmos meridianos. Um valor elevado do semi-eixo maior faz com que o satélite pareça executar um movimento angular maior do que $360°$, resultando em sobreposições no plano projetado.

A excentricidade orbital também impacta o *ground track*. Enquanto órbitas circulares geram projeções simétricas em torno do Equador, órbitas elípticas apresentam assimetrias devido às variações de velocidade nos pontos de apogeu e perigeu. No apogeu, a velocidade do satélite é menor, fazendo com que ele permaneça mais tempo em determinadas longitudes. Já no perigeu, o satélite se desloca rapidamente, gerando trechos mais comprimidos no *ground track*. Isso causa, por vezes, projeções cruzadas — quando o satélite aparenta se mover ora para leste, ora para oeste no plano bidimensional, mesmo mantendo uma única trajetória espacial.

A Figura 2.7 exibe um exemplo de *ground track* de uma órbita geoestacionária, enquanto a Figura 2.8 apresenta uma órbita polar.

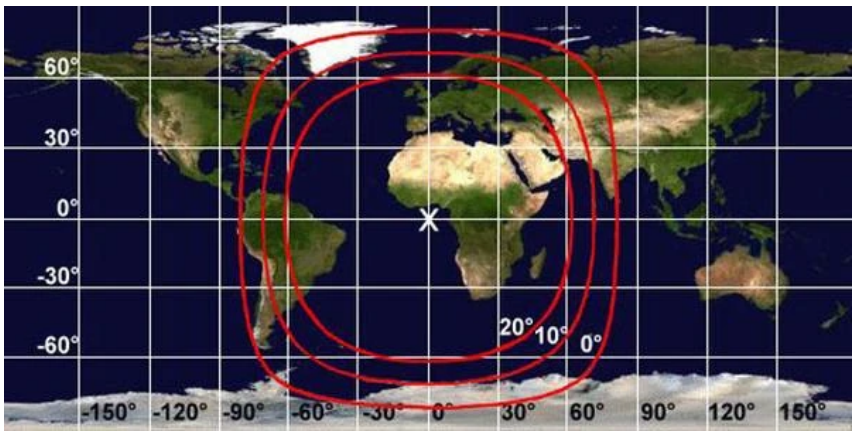

Figura 2.7: Órbita tipo Geoestacionária

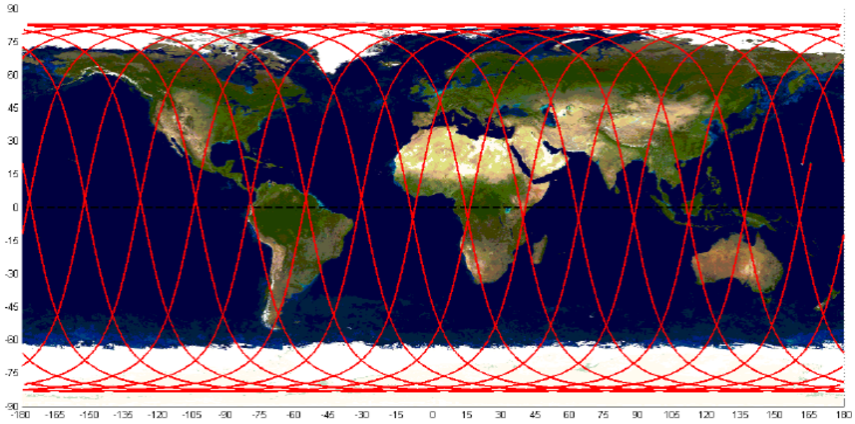

Figura 2.8: Órbita tipo Polar

Capítulo 3

Objetivo e Estrutura Geral

O programa principal foi desenvolvido para realizar conversões entre sistemas de coordenadas geodésicas, terrestres e inerciais. A estrutura central consiste na classe `LocalizaPonto`, que encapsula todas as funcionalidades necessárias para estas transformações.

3.1 Resultado

Este estudo apresenta os *ground tracks* de quatro satélites distintos: **CBERS-4A**, **ISS (Estação Espacial Internacional)**, **MOLNIYA 1-91** e **STARONE D2**, cada um com características orbitais específicas que influenciam sua trajetória aparente.

Os resultados ilustram como diferentes configurações orbitais – como órbitas baixas (LEO), órbitas médias (MEO) e órbitas altamente elípticas (HEO) – geram padrões únicos de cobertura terrestre. Por exemplo, a ISS, em sua órbita LEO, exibe um *ground track* denso e frequente, enquanto satélites como o MOLNIYA, em órbitas HEO, apresentam trajetórias assimétricas com maior permanência em altas latitudes. Já satélites geoestacionários, como o STARONE D2, mostram um *ground track* praticamente fixo sobre o equador.

As figuras a seguir destacam essas diferenças, fornecendo insights sobre o desenho orbital e a aplicação de cada satélite, essenciais para missões de observação, comunicação ou monitoramento global.

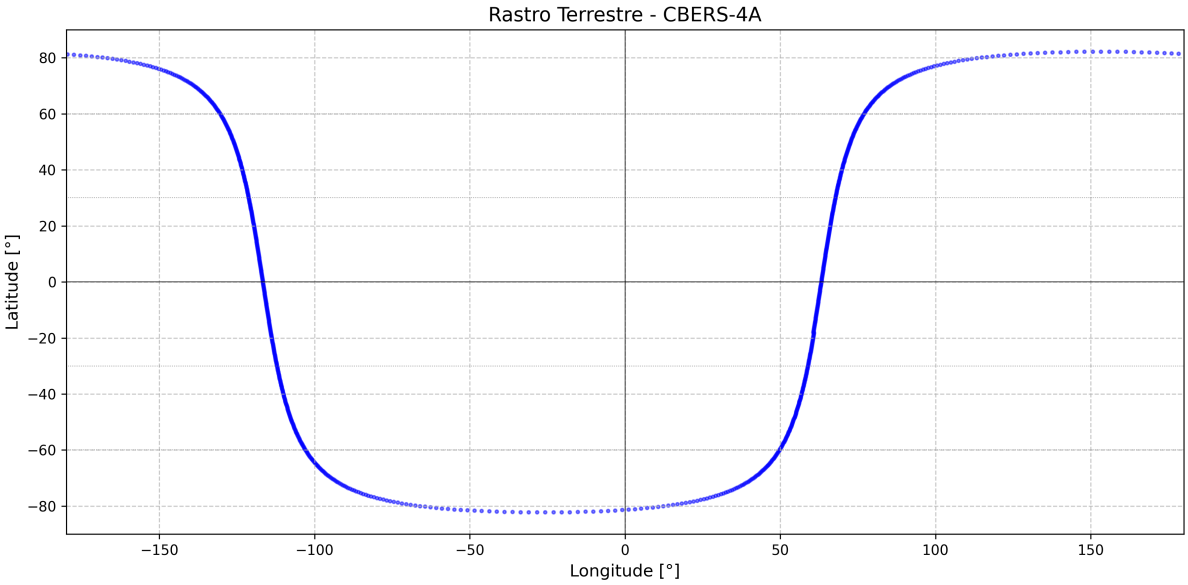

Figura 3.1: Ground Track CBERS

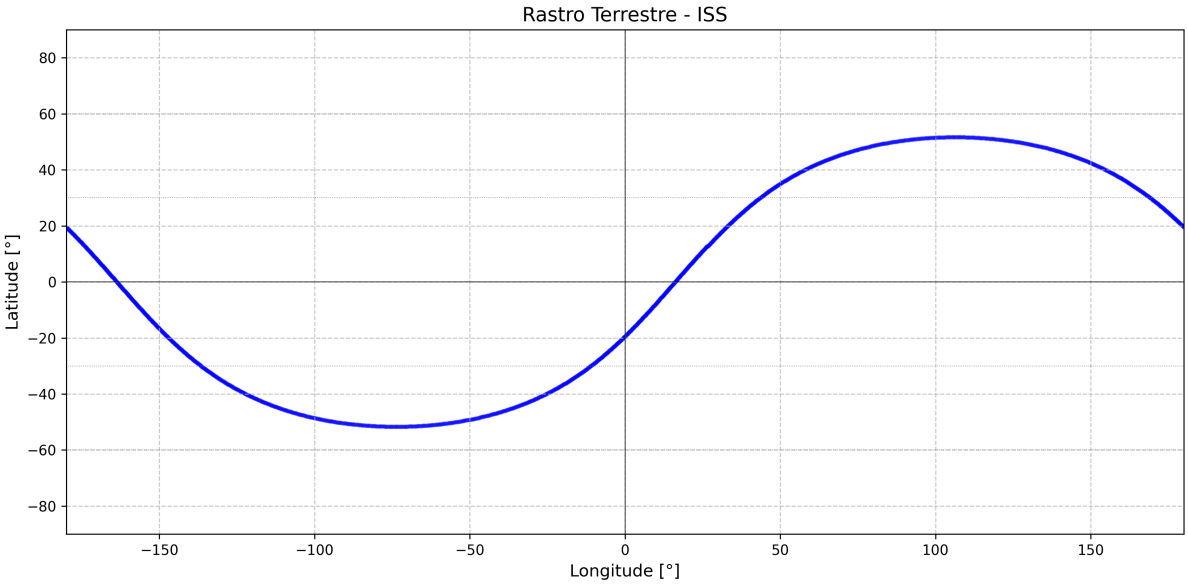

Figura 3.2: Ground Track ISS

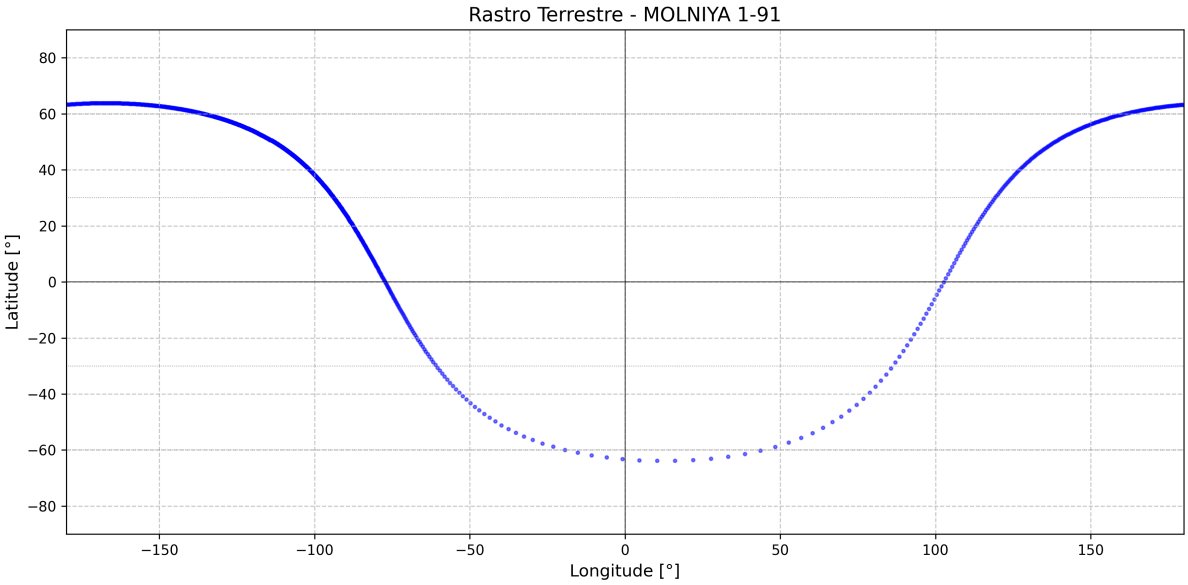

Figura 3.3: Ground Track MOLNIYA

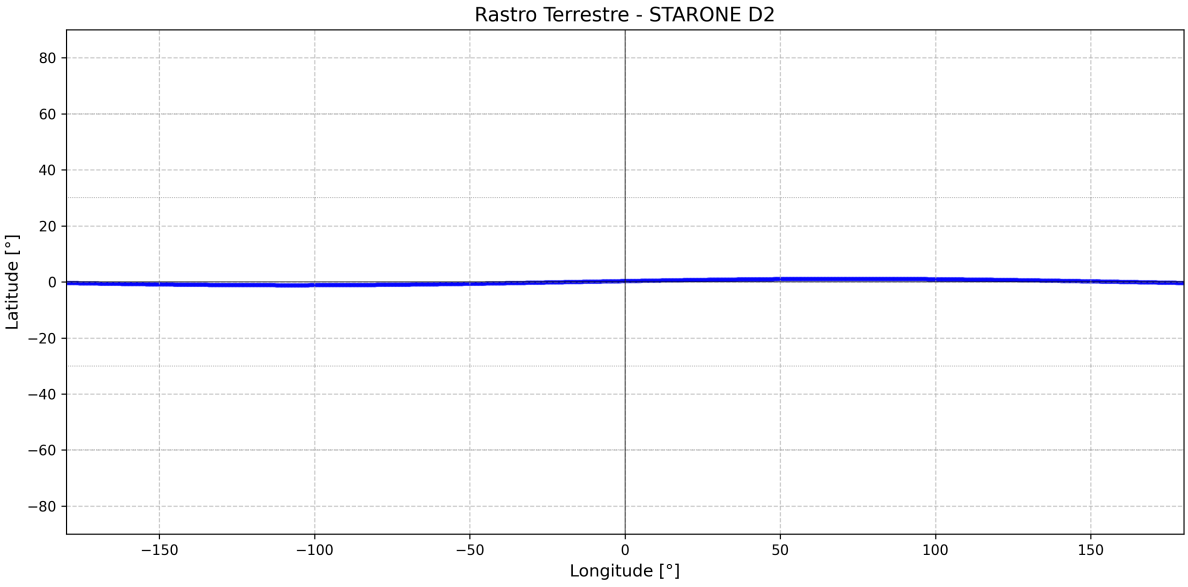

Figura 3.4: Ground Track STARONE

A Tabela 3.1 apresenta as coordenadas cartesianas dos pontos nos sistemas terrestre, inercial e recuperado, expressas em metros. Essas coordenadas representam a posição dos pontos em diferentes sistemas de referência usados na análise espacial.

Tabela 3.1: Coordenadas em sistemas terrestre, inercial e recuperado

Ponto	Sistema	Coordenadas X (m)	Coordenadas Y (m)	Coordenadas Z (m)
1	Terrestre	5186540.577130	-3653846.197315	-653798.938410
	Inercial	-633889.855447	6312604.758973	-653798.938410
	Recuperado	5186540.577130	-3653846.197315	-653798.938410
2	Terrestre	4552875.975972	-4459283.952792	-258552.335741
	Inercial	3631112.236534	5237262.352594	-258552.335741
	Recuperado	4552875.975972	-4459283.952792	-258552.335741
3	Terrestre	4018867.632662	-4244306.726064	-2545867.827774
	Inercial	-5693304.637041	-1323525.195687	-2545867.827774
	Recuperado	4018867.632662	-4244306.726064	-2545867.827774

A Tabela 3.2 mostra os versores associados à posição de cada ponto nos sistemas terrestre e inercial. Esses vetores unitários indicam a direção dos pontos em relação à origem dos respectivos sistemas de coordenadas.

Tabela 3.2: Versores nas bases terrestre e inercial

Ponto	Sistema	Versor
1	Terrestre	(0.813199, -0.572887, -0.102509)
1	Inercial	(-0.099388, 0.989754, -0.102509)
2	Terrestre	(0.713824, -0.699151, -0.040537)
2	Inercial	(0.569305, 0.821126, -0.040537)
3	Terrestre	(0.630362, -0.665722, -0.399321)
3	Inercial	(-0.892999, -0.207596, -0.399321)

3.2 Classes e Métodos Implementados

3.2.1 Classe LocalizaPonto

A classe principal contém os seguintes métodos essenciais:

- `__init__`: Inicializa os parâmetros do elipsoide GRS80
- `gms_para_decimal`: Conversão de graus, minutos, segundos para decimal
- `geografica_para_terrestre`: Transformação de coordenadas geográficas para terrestres

- `terrestre_para_inercial`: Conversão de coordenadas terrestres para inerciais
- `tempo_sideral_greenwich`: Cálculo do tempo sideral de Greenwich
- `processar_pontos`: Processamento em lote de múltiplos pontos

3.3 Conversão de Coordenadas

3.3.1 Geográficas para Terrestres

A transformação implementada segue a formulação:

$$\begin{aligned} N &= \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} \\ x &= (N + h) \cos \phi \cos \lambda \\ y &= (N + h) \cos \phi \sin \lambda \\ z &= \left[N(1 - e^2) + h \right] \sin \phi \end{aligned} \tag{3.1}$$

onde:

- a: semi-eixo maior do elipsoide
- e: excentricidade
- ϕ, λ, h: latitude, longitude e altitude

3.3.2 Terrestres para Inerciais

Utiliza uma matriz de rotação baseada no tempo sideral de Greenwich (θ_G):

$$\mathbf{R} = \begin{bmatrix} \cos \theta_G & \sin \theta_G & 0 \\ -\sin \theta_G & \cos \theta_G & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

3.4 Cálculo do Tempo Sideral

Implementado conforme a fórmula:

$$\theta_G = 280.46061837 + 360.98564736629 \cdot (JD - 2451545.0) + 0.000387933 \cdot T^2 - \frac{T^3}{38710000} \tag{3.3}$$

onde T é o número de séculos julianos desde J2000.0.

3.5 Testes e Validação

O programa inclui um conjunto de testes com três pontos geográficos distintos, verificando:

- Consistência nas transformações direta e inversa
- Normalização de vetores unitários
- Precisão numérica (6 casas decimais)

Capítulo 4

Desenvolvimento do Programa Secundário (ground_track.py)

4.1 Objetivo e Aplicação

Este módulo foi projetado para calcular e visualizar órbitas de satélites e seus respectivos rastros terrestres, utilizando dados de TLE (Two-Line Elements).

4.2 Classes e Métodos Principais

4.2.1 Classe CalculadorRastroTerrestre

Contém os principais métodos para:

- Leitura e processamento de TLEs
- Integração numérica das equações de movimento
- Cálculo do período orbital
- Transformação para coordenadas geográficas
- Visualização 2D e 3D dos resultados

4.3 Dinâmica Orbital

4.3.1 Equações de Movimento

As equações diferenciais implementadas são:

$$\begin{aligned} \frac{d\mathbf{r}}{dt} &= \mathbf{v} \\ \frac{d\mathbf{v}}{dt} &= -\frac{\mu}{r^3}\mathbf{r} \end{aligned} \tag{4.1}$$

integradas numericamente com o método RK45.

4.3.2 Cálculo do Período Orbital

Utiliza a relação energética:

$$T = 2\pi\sqrt{\frac{a^3}{\mu}} \tag{4.2}$$

onde a é o semi-eixo maior da órbita.

4.4 Transformação para Rastro Terrestre

4.4.1 Conversão para Lat/Lon

Implementa as transformações:

$$\begin{aligned} \phi &= \arcsin\left(\frac{z}{r}\right) \\ \lambda &= \arctan 2(y, x) - \theta_G \end{aligned} \tag{4.3}$$

com normalização para o intervalo $[-180°, 180°]$.

4.5 Visualização

4.5.1 Gráfico 2D do Rastro

Gera mapas com:

- Linhas de referência (Equador, Greenwich)
- Grade de coordenadas
- Destaque para latitudes notáveis

4.5.2 Visualização 3D da Órbita

Apresenta:

- Trajetória do satélite em 3D

- Elipsoide terrestre de referência
- Escala automática dos eixos

4.6 Exemplos Implementados

O programa processa quatro satélites com características orbitais distintas:

- ISS (Órbita baixa, inclinada)
- CBERS-4A (Órbita polar heliossíncrona)
- MOLNIYA (Órbita altamente elíptica)
- STARONE D2 (Geoestacionário)

Capítulo 5

Integração entre os Módulos

5.1 Consistência nos Modelos

Ambos os módulos utilizam:

- Mesmo elipsoide de referência (GRS80)
- Mesma implementação do tempo sideral
- Sistema de coordenadas compatível

5.2 Aplicações Conjuntas

Os sistemas podem ser integrados para:

- Determinar posições de satélites em relação a pontos na superfície
- Calcular janelas de visibilidade
- Realizar análises de cobertura terrestre

5.3 Possíveis Melhorias

- Implementação de perturbações orbitais
- Cálculo de estações do ano nos gráficos
- Interpolação para visualização suavizada

Capítulo 6

Conclusões

Primeiramente, foi observado que os valores encontrados para latitude e longitude devem ser tratados de forma que fiquem contidos nos intervalos correspondentes ao do planisfério em questão. Para este relatório, o intervalo considerado para a latitude foi de -90° a $+90^\circ$, enquanto que para a longitude foi de -180° a $+180^\circ$.

Além disso, foi possível verificar que a rotação da Terra gera "descontinuidades" no *ground track*, dado que a cada revolução do V/E, a Terra terá rotacionado em relação à revolução anterior, fazendo com que, após um período orbital, o veículo não se encontre na mesma posição que se encontrava em relação à Terra antes de realizar uma volta ao redor do planeta. Isso ocorre pois os satélites considerados na atividade possuem um período orbital diferente do período de rotação terrestre.

Contudo, é válido ressaltar que não há descontinuidade verdadeira na trajetória projetada do veículo e, mais do que isso, em órbitas retrógradas, o veículo poderá passar por um mesmo meridiano mais de uma vez ao longo de um período orbital, o que não ocorre no caso de órbitas progradas.

Por fim, foi visto que é possível determinar a inclinação da órbita analisada observando os valores máximo e mínimo da latitude. Para órbitas que possuem inclinação i entre 0° e 90°, o maior valor de latitude encontrada é igual ao ângulo de inclinação da órbita. Já para órbitas retrógradas, que possuem inclinação i entre 90° e 180°, é necessário subtrair a máxima latitude de 180° para obter o valor de i.

Bibliografia

[1] VALLADO, D. A. *Fundamentals of Astrodynamics and Applications*. 4. ed. El Segundo: Microcosm Press, 2013.

[2] CURTIS, H. D. *Orbital Mechanics for Engineering Students*. Butterworth-Heinemann, 2013.

[3] PROJ Contributors. *Geodetic Conversions*. Disponível em: `https://proj.org/operations/conversions/geoc.html`

[4] ADCS for Beginners. *Earth-Centered Inertial Frame*. Disponível em: `https://adcsforbeginners.wordpress.com/tag/earth-centred-inertial-frame/`

[5] Wikipedia. *Ground track*. Disponível em: `https://en.wikipedia.org/wiki/Ground_track`

[6] Wikipedia. *Geodetic Reference System 1980*. Disponível em: `https://en.wikipedia.org/wiki/Geodetic_Reference_System_1980`

Apêndice A

Programa principal main

Conteúdo adicional que não foi incluído no corpo principal do relatório.

```
from math import sqrt, radians, cos, sin, acos, asin
import datetime
import math
import unittest

class LocalizaPonto: #classe que cont m as fun  es para
   geolocalizar um ponto
    def __init__(self):
        # Par metros do elipsoide GSW84
        self.a = 6378137.0  # semi-eixo maior [m]
        self.b = 6356752.0  # semi-eixo menor [m]
        self.e = sqrt((self.a**2 - self.b**2) / self.a**2)  #
           excentricidade

    def gms_para_decimal(self, grau, minutos, segundos, direcao):
        """Converte coordenadas de grau, minutos, segundos para
           decimal"""
        decimal = grau + minutos/60 + segundos/3600
        if direcao in ['S', 'W']:
            decimal *= -1
        return decimal

    def geografica_para_terrestre(self, lat_grau, lat_min,
       lat_seg, lat_dir,
                                  lon_grau, lon_min, lon_seg,
                                     lon_dir, alt):
        """Converte coordenadas geogr ficas para terrestres"""
        # Converter DMS para decimal
```

```
        phi = self.gms_para_decimal(lat_grau, lat_min, lat_seg,
            lat_dir)
        lam = self.gms_para_decimal(lon_grau, lon_min, lon_seg,
            lon_dir)

        # Converter para radianos
        phi_rad = radians(phi)
        lam_rad = radians(lam)

        # Calcular raio de curvatura
        N = self.a / sqrt(1 - (self.e**2 * sin(phi_rad)**2))

        # Calcular coordenadas terrestres
        x = (N + alt) * cos(phi_rad) * cos(lam_rad)
        y = (N + alt) * cos(phi_rad) * sin(lam_rad)
        z = (N * (1 - self.e**2) + alt) * sin(phi_rad)

        return np.array([x, y, z])

    def terrestre_para_inercial(self, r_terrestre, data,
        longitude):
        """Converte coordenadas terrestres para inerciais"""
        # Calcular tempo sideral de Greenwich
        theta_g = self.tempo_sideral_greenwich(data)

        # Matriz de rota  o
        R = np.array([
            [cos(theta_g), sin(theta_g), 0],
            [-sin(theta_g), cos(theta_g), 0],
            [0, 0, 1]
        ])

        # Aplicar rota  o
        r_inercial = R.T @ r_terrestre

        return r_inercial

    import math

    def tempo_sideral_greenwich(self, data):
        """Calcula o tempo sideral de Greenwich em radianos para
```

```
    uma data/hora espec fica"""

# Extrair componentes da data/hora
ano = data.year
mes = data.month
dia = data.day
hora = data.hour
minuto = data.minute
segundo = data.second

# Ajustar m s e ano se necess rio
if mes <= 2:
    ano -= 1
    mes += 12

# Calcular parte inteira do dia juliano
parte_a = ano // 100
parte_b = 2 - parte_a + (parte_a // 4)

# Calcular dia juliano (formula completa)
termo1 = 365.25 * (ano + 4716)
termo2 = 30.6001 * (mes + 1)
parte_inteira = int(termo1) + int(termo2) + dia + parte_b
    - 1524

# Calcular fra  o do dia
fra  o_dia = (hora + minuto/60.0 + segundo/3600.0) /
   24.0

# Dia juliano completo
dia_juliano = parte_inteira + fra  o_dia - 0.5

# S culos julianos desde J2000.0
T = (dia_juliano - 2451545.0) / 36525.0

# Calcular tempo sideral de Greenwich em grau
termo_constante = 280.46061837
termo_linear = 360.98564736629 * (dia_juliano -
   2451545.0)
termo_quadratico = 0.000387933 * (T * T)
termo_cubico = (T * T * T) / 38710000.0
```

```
        theta_g = termo_constante + termo_linear +
            termo_quadratico - termo_cubico

        # Normalizar para o intervalo [0, 360) grau
        theta_g = theta_g - 360.0 * (theta_g // 360)
        if theta_g < 0:
            theta_g += 360.0

        # Converter para radianos
        return theta_g * (math.pi / 180.0)


    def processar_pontos(self, pontos):
        """Processa uma lista de pontos e retorna os resultados
            """
        resultados = []

        for ponto in pontos:
            # Extrair dados do ponto
            dados_lat = ponto['latitude']
            dados_lon = ponto['longitude']
            altura = ponto['altitude']
            data_hora = ponto['datetime']

            # Converter para coordenadas terrestres
            r_terrestre = self.geografica_para_terrestre(
                dados_lat['grau'], dados_lat['min'], dados_lat['
                    seg'], dados_lat['dir'],
                dados_lon['grau'], dados_lon['min'], dados_lon['
                    seg'], dados_lon['dir'],
                altura
            )

            # Converter longitude para decimal
            lon_decimal = self.gms_para_decimal(
                dados_lon['grau'], dados_lon['min'], dados_lon['
                    seg'], dados_lon['dir']
            )

            # Converter para coordenadas inerciais
```

```
            r_inercial = self.terrestre_para_inercial(r_terrestre
                , data_hora, lon_decimal)

            # Calcular norma do vetor terrestre
            norma_terrestre = math.sqrt(
                r_terrestre[0]**2 + r_terrestre[1]**2 +
                    r_terrestre[2]**2
            )
            r_terrestre_unitario = [
                r_terrestre[0] / norma_terrestre,
                r_terrestre[1] / norma_terrestre,
                r_terrestre[2] / norma_terrestre
            ]

            # Calcular norma do vetor inercial
            norma_inercial = math.sqrt(
                r_inercial[0]**2 + r_inercial[1]**2 + r_inercial
                    [2]**2
            )
            r_inercial_unitario = [
                r_inercial[0] / norma_inercial,
                r_inercial[1] / norma_inercial,
                r_inercial[2] / norma_inercial
            ]

            resultados.append({
                'terrestre': r_terrestre,
                'terrestre_unitario': r_terrestre_unitario,
                'inercial': r_inercial,
                'inercial_unitario': r_inercial_unitario
            })

        return resultados

    def inercial_para_terrestre(self, r_inercial, data, longitude
        ):
        """Converte coordenadas inerciais de volta para
            terrestres: teste"""
        theta_g = self.tempo_sideral_greenwich(data)
        R = np.array([
            [cos(theta_g), sin(theta_g), 0],
```

```
            [-sin(theta_g), cos(theta_g), 0],
            [0, 0, 1]
        ])
        return R @ r_inercial  # Multiplica o direta (sem
            transposta)



if __name__ == "__main__":
    # Definir pontos da atividade
    pontos = [
        {
            'latitude': {'grau': 5, 'min': 55, 'seg': 23, 'dir':
                'S'},
            'longitude': {'grau': 35, 'min': 9, 'seg': 51, 'dir':
                'W'},
            'altitude': 39,
            'datetime': datetime.datetime(2025, 6, 3, 15, 54, 10)
        },
        {
            'latitude': {'grau': 2, 'min': 20, 'seg': 20, 'dir':
                'S'},
            'longitude': {'grau': 44, 'min': 24, 'seg': 18, 'dir'
                : 'W'},
            'altitude': 44,
            'datetime': datetime.datetime(2024, 7, 10, 11, 23,
                10)
        },
        {
            'latitude': {'grau': 23, 'min': 40, 'seg': 37, 'dir':
                'S'},
            'longitude': {'grau': 46, 'min': 33, 'seg': 46, 'dir'
                : 'W'},
            'altitude': 778,
            'datetime': datetime.datetime(2025, 6, 24, 21, 45,
                25)
        }
    ]

    localizador = LocalizaPonto()
```

```
resultados = localizador.processar_pontos(pontos)

def format_array(arr, decimals=6):
    return [f"{float(x):.{decimals}f}" for x in arr]

# Exibir resultados
for i, resultado in enumerate(resultados, 1):
    print(f"\nPonto {i}:")
    print("Coordenadas terrestres [m]:", format_array(
        resultado['terrestre']))
    print("Versor terrestre:", format_array(resultado['
        terrestre_unitario']))
    print("Coordenadas inerciais [m]:", format_array(
        resultado['inercial']))
    print("Versor inercial:", format_array(resultado['
        inercial_unitario']))

    # Teste de convers o inversa para cada ponto
    r_terrestre_recuperado = localizador.
        inercial_para_terrestre(
         resultado['inercial'],
         pontos[i-1]['datetime'],
         localizador.gms_para_decimal(
             pontos[i-1]['longitude']['grau'],
             pontos[i-1]['longitude']['min'],
             pontos[i-1]['longitude']['seg'],
             pontos[i-1]['longitude']['dir']
         )
    )
    print(f"Ponto {i} - terrestre recuperado:", format_array(
        r_terrestre_recuperado))
    print(f"Diferen a terrestre original vs recuperado:",
          format_array(np.array(resultado['terrestre']) - np.
              array(r_terrestre_recuperado)))
```

Apêndice B

Programa secundario Ground Track

Conteúdo adicional que não foi incluído no corpo principal do relatório.

```
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
from math import sin, cos, sqrt, radians, degrees, atan2, asin,
   pi
import datetime
from sgp4.api import Satrec, jday

class CalculadorRastroTerrestre:
    def __init__(self):
        # Constantes f sicas
        self.G = 6.67430e-11  # constante gravitacional [m^3/kg/s
           ^2]
        self.M_terra = 5.972e24  # massa da Terra [kg]
        self.mu = self.G * self.M_terra
        self.omega_terra = 7.2921150e-5  # velocidade angular da
           Terra [rad/s]

        # Par metros do elipsoide terrestre (GRS80)
        self.a = 6378137.0  # semi-eixo maior [m]
        self.b = 6356752.0  # semi-eixo menor [m]

    def obter_estado_do_tle(self, linha1, linha2, ano, mes, dia,
       hora, minuto, segundo):
        """Obt m posi  o e velocidade a partir de TLE usando
           SGP4"""
        sat = Satrec.twoline2rv(linha1, linha2)
        jd, fr = jday(ano, mes, dia, hora, minuto, segundo)
        e, r, v = sat.sgp4(jd, fr)
```

```
        if e != 0:
            raise RuntimeError("Erro ao processar TLE")
        return np.array(r) * 1000, np.array(v) * 1000  # converte
            km para m

    def dinamica_orbital(self, t, estado):
        """Equa  es diferenciais para movimento orbital"""
        x, y, z, vx, vy, vz = estado
        r = sqrt(x**2 + y**2 + z**2)

        dxdt = vx
        dydt = vy
        dzdt = vz
        dvxdt = -self.mu * x / r**3
        dvydt = -self.mu * y / r**3
        dvzdt = -self.mu * z / r**3

        return [dxdt, dydt, dzdt, dvxdt, dvydt, dvzdt]

    def calcular_orbita(self, r0, v0, periodo, n_pontos=1000):
        """Calcula  rbita  por um per odo"""
        estado0 = [r0[0], r0[1], r0[2], v0[0], v0[1], v0[2]]
        t_eval = np.linspace(0, periodo, n_pontos)
        sol = solve_ivp(self.dinamica_orbital, [0, periodo],
           estado0,
                        t_eval=t_eval, method='RK45', rtol=1e-8,
                           atol=1e-10)
        return sol.y.T, t_eval

    def dia_juliano(self, dt):
        """Calcula Dia Juliano a partir de datetime"""
        a = (14 - dt.month) // 12
        y = dt.year + 4800 - a
        m = dt.month + 12*a - 3

        jd = dt.day + (153*m + 2)//5 + 365*y + y//4 - y//100 + y
           //400 - 32045
        jd += (dt.hour - 12)/24 + dt.minute/1440 + dt.second
           /86400

        return jd
```

```
    def tempo_sideral_greenwich(self, jd):
        """Calcula Tempo Sideral de Greenwich"""
        T = (jd - 2451545.0) / 36525
        theta_g = 280.46061837 + 360.98564736629*(jd - 2451545.0)
            + 0.000387933*T**2 - T**3/38710000
        return theta_g % 360  # graus

    def calcular_rastro_terrestre(self, orbita, tempo_inicio):
        """Calcula coordenadas do rastro terrestre"""
        latitudes = []
        longitudes = []

        jd_inicio = self.dia_juliano(tempo_inicio)

        for i in range(len(orbita)):
            x, y, z = orbita[i,0], orbita[i,1], orbita[i,2]
            r = sqrt(x**2 + y**2 + z**2)

            # Calcula Dia Juliano atual
            jd = jd_inicio + (i/len(orbita)) * (1/1440)  #
               assumindo 1 per odo = 1 dia

            # Calcula GST em graus
            theta_g = self.tempo_sideral_greenwich(jd)

            # Calcula latitude e longitude
            lat = degrees(asin(z / r))
            lon = degrees(atan2(y, x)) - theta_g

            # Normaliza longitude para [-180, 180]
            while lon < -180:
                lon += 360
            while lon > 180:
                lon -= 360

            latitudes.append(lat)
            longitudes.append(lon)

        return latitudes, longitudes
```

```
    def plotar_rastro_terrestre(self, latitudes, longitudes,
        titulo="Rastro Terrestre"):
        """Plota o rastro terrestre"""
        plt.figure(figsize=(12, 6))

        # Plota rastro terrestre
        plt.scatter(longitudes, latitudes, s=5, c='b', alpha=0.5)

        # Formata o
        plt.title(titulo, fontsize=14)
        plt.xlabel("Longitude [ ]", fontsize=12)
        plt.ylabel("Latitude [ ]", fontsize=12)
        plt.xlim(-180, 180)
        plt.ylim(-90, 90)

        # Adiciona grade e linhas de refer ncia
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.axhline(0, color='k', linestyle='-', linewidth=0.5)
            # Equador
        plt.axvline(0, color='k', linestyle='-', linewidth=0.5)
            # Meridiano de Greenwich

        # Adiciona linhas de latitude de refer ncia
        for lat in [-60, -30, 30, 60]:
            plt.axhline(lat, color='gray', linestyle=':',
                linewidth=0.5)

        plt.tight_layout()
        plt.show()

    def plotar_orbita_3d(self, orbita, titulo=" rbita 3D"):
        """Plota rbita 3D com a Terra"""
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111, projection='3d')

        # Plota rbita
        ax.plot(orbita[:,0]/1000, orbita[:,1]/1000, orbita
            [:,2]/1000, 'b', linewidth=1)

        # Plota Terra
        u = np.linspace(0, 2 * np.pi, 100)
```

```
            v = np.linspace(0, np.pi, 100)
            x = self.a * np.outer(np.cos(u), np.sin(v)) / 1000
            y = self.a * np.outer(np.sin(u), np.sin(v)) / 1000
            z = self.b * np.outer(np.ones(np.size(u)), np.cos(v)) /
                1000
            ax.plot_surface(x, y, z, color='lightblue', alpha=0.3)

            ax.set_xlabel('X [km]')
            ax.set_ylabel('Y [km]')
            ax.set_zlabel('Z [km]')
            ax.set_title(titulo)

            # Equaliza eixos
            max_range = np.array([orbita[:,0].max()-orbita[:,0].min()
                ,
                                  orbita[:,1].max()-orbita[:,1].min(),
                                  orbita[:,2].max()-orbita[:,2].min()
                                     ]).max() / 1000 / 2.0
            mid_x = (orbita[:,0].max()+orbita[:,0].min()) * 0.5 /
                1000
            mid_y = (orbita[:,1].max()+orbita[:,1].min()) * 0.5 /
                1000
            mid_z = (orbita[:,2].max()+orbita[:,2].min()) * 0.5 /
                1000
            ax.set_xlim(mid_x - max_range, mid_x + max_range)
            ax.set_ylim(mid_y - max_range, mid_y + max_range)
            ax.set_zlim(mid_z - max_range, mid_z + max_range)

            plt.tight_layout()
            plt.show()

        def calcular_periodo_orbital(self, r0, v0):
            """Calcula per odo orbital a partir de posi  o e
                velocidade"""
            epsilon = (np.linalg.norm(v0)**2)/2 - self.mu/np.linalg.
                norm(r0)
            a = -self.mu/(2*epsilon)
            return 2 * pi * sqrt(a**3 / self.mu)


    if __name__ == "__main__":
```

```
        # Inicializa calculador
        rt = CalculadorRastroTerrestre()

        # TLEs de sat lites
        satelites = {
            "ISS": (
                "1 25544U 98067A   25150.54603503  .00012878  00000-0
                     23439-3 0  9999",
                "2 25544  51.6399  34.4830 0002197 166.0379 262.3840
                   15.49859072512427"
            ),
            "CBERS-4A": (
                "1 44883U 19093E   25150.83982066  .00001032  00000-0
                     13795-3 0  9998",
                "2 44883  97.7932 224.3455 0001774   6.2371 353.8864
                   14.81582034294467"
            ),
            "MOLNIYA 1-91": (
                "1 25485U 98054A   25150.46666369 -.00000126  00000-0
                     00000-0 0  9999",
                "2 25485  64.4919 341.5937 6788400 287.6338  12.7988
                    2.36440192204520"
            ),
            "STARONE D2": (
                "1 49055U 21069A   25150.42901189 -.00000270  00000-0
                     00000+0 0  9997",
                "2 49055   0.0110 322.8887 0001735 134.5547 235.1911
                    1.00271589 14066"
            )
        }

        # Data de refer ncia
        data_ref = datetime.datetime(2006, 9, 11, 0, 0, 0)

        # Processa cada sat lite
        for nome, (linha1, linha2) in satelites.items():
            print(f"\nProcessando {nome}...")

            # Obt m estado inicial do TLE
            r0, v0 = rt.obter_estado_do_tle(linha1, linha2, 2023, 5,
               30, 0, 0, 0)
```

```
        # Calcula per odo orbital
        periodo = rt.calcular_periodo_orbital(r0, v0)
        print(f"Per odo orbital: {periodo/3600:.2f} horas")

        # Calcula  rbita
        orbita, t = rt.calcular_orbita(r0, v0, periodo)

        # Calcula rastro terrestre
        lats, lons = rt.calcular_rastro_terrestre(orbita,
            data_ref)

        # Plota resultados
        rt.plotar_orbita_3d(orbita, f" rbita  3D - {nome}")
        rt.plotar_rastro_terrestre(lats, lons, f"Rastro Terrestre
            - {nome}")
```