UNIVERSIDADE FEDERAL DO ABC

Luan Henrique Glasser

Desenvolvimento de um Simulador em C++ para a Propagação de Detritos Espaciais em Órbita Alta

Brasil

6 de setembro de 2018

Luan Henrique Glasser

Desenvolvimento de um Simulador em C++ para a Propagação de Detritos Espaciais em Órbita Alta

Relatório apresentado à disciplina "Trabalho de Graduação III" do curso de Engenharia Aeroespacial da Universidade Federal do ABC, como requisito para obtenção do grau de Engenheiro Aeroespacial.

Universidade Federal do ABC

Orientadora: Dra. Claudia Celeste Celestino de Paula Santos

Brasil

6 de setembro de 2018

Luan Henrique Glasser
Desenvolvimento de um Simulador em C++ para a Propagação de Detritos Espaciais em Órbita Alta/ Luan Henrique Glasser. – Brasil, 6 de setembro de 2018-
87 p. : il. (algumas color.) ; 30 cm.

Orientadora: Dra. Claudia Celeste Celestino de Paula Santos

Monografia – Universidade Federal do ABC , 6 de setembro de 2018.

1. Simulação computacional de órbitas. 2. Problema de quatro corpos. 3. Pressão de radiação solar. I. Claudia Celeste Celestino de Paula Santos. II. Universidade Federal do ABC. III. Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas. IV. Desenvolvimento de um Simulador em C++ para a Propagação de Detritos Espaciais em Órbita Alta.

Luan Henrique Glasser

Desenvolvimento de um Simulador em C++ para a Propagação de Detritos Espaciais em Órbita Alta

Relatório apresentado à disciplina "Trabalho de Graduação III" do curso de Engenharia Aeroespacial da Universidade Federal do ABC, como requisito para obtenção do grau de Engenheiro Aeroespacial.

Trabalho aprovado. Brasil, 6 de setembro de 2018.

Dra. Claudia Celeste Celestino de Paula Santos
Orientadora

Dr. Lendro Baroni
Professor Convidado

Dr. Rodolpho Vilhena de Moraes
Professor Convidado

Brasil
6 de setembro de 2018

Para minhas raízes.

Agradecimentos

Meus primeiros agradecimentos são destinados a quatro pessoas: meus pais, Maria C. ”Tininha” da Silva Glasser e Marcos Glasser, e meus Tios, Carmem Lúcia ”Melu” da Silva Sato e Jânio ”Mitio” Sato. Essas pessoas fomentaram essa jornada quase sabática que foi a graduação na UFABC e me deram todo apoio do mundo. Me considero alguém de sorte por sempre ter tido essas pessoas em minha retaguarda.

Agradeço a meu irmão e prima - irmã -, Murilo de Carvalho Glasser e Gabriela da Silva Sato, pela paciência, conversas e brincadeiras ao longo desses anos. Espero sempre - tentar - servir de exemplo (ou de mal exemplo, quando necessário) a vocês. Obrigado pela fraternidade que sempre me deram. Agradeço a meu tio André Luís da Silva, que também acompanhou de perto o caminho trilhado.

Um obrigado mais que especial a Nathalia Paula, minha namorada, que acompanhou mais perto que qualquer pessoa, sentiu na pele e compartilhou, comigo, cada passo, dificuldade, estresse, desafio, tristeza, sofrimento, felicidade, conquista e evolução pessoal provocados por essa jornada de seis anos e pouco. Muito do que sou hoje vem dela e seu companheirismo e amor me mantiveram - e mantém - nos trilhos sem sair de rota.

Obrigado a Christian Herrera, Elizabeth Santibañez, Brunno Mark e Stephani Christy, pais e irmãos de minha namorada. O apoio, conselhos e conversas com vocês foram importantes em muitos momentos.

Agradeço a meus avós Luiz Carlos da Silva e Maria Oneti Batista da Silva. A meus avós José de Carvalho Glasser e Maria Soares Glasser. Vocês me ensinaram a cuidar das pessoas e são um modelo de conduta para mim.

Vô Carlos, obrigado em especial pelo interesse que me ensinou a ter pela engenharia, em sua oficina, e pela estratégia, em centenas (milhares?) de jogos de damas.

Agradeço a professora Cláudia Celeste, por ter me aceitado como aluno orientado. Foi um prazer desenvolver esse trabalho contigo e espero ter mais chances nos tempos vindouros.

Obrigado aos professores que compuseram a banca avaliadora: os titulares Leandro Baroni e Rodolpho Vilhena; e os suplentes Maria Cecília Zanardi e Carlos Solorzano.

Agradeço a UFABC, que moldou a pessoa que sou hoje, me ensinou a pensar, me deu outras visões de mundo e me fez aprender, principalmente, que cair no caminho acontece, mas ficar caído não é uma opção.

"All you have to decide is what to do with the time is given to you."
(Gandalf, Terceira Era da Terra-Média)
"É tipo como lixo que se movimenta no espaço?"
(Flávio Siqueira, 2018)
"Absque sudore et labore nullum opus perfectum est."
(Schrevelius, 1176)

Resumo

Muitos detritos estão em órbita alta ao redor da Terra, representando perigos a veículos, satélites em operação, em especial de comunicação e climáticos, e outros ainda por lançar. É importante, portanto, estudar o comportamento desses detritos em órbita e dispor de tecnologias que permitam esses estudos. Neste trabalho foi construído um simulador de evolução orbital em C++, utilizando como base o simulador de Celestino (2005), programado em Fortran, para obter a evolução de detritos espaciais em órbita alta sujeitos a perturbações da pressão de radiação solar e gravitacionais da Lua e do Sol. Foram utilizadas equações de movimento modeladas para cada perturbação e o integrador numérico integrate_const da biblioteca Odeint, para C++, de forma a realizar a propagação orbital. O objetivo foi usar o propagador desenvolvido para descrever o comportamento do detrito, obtendo resultados numéricos coerentes.

Palavras-chave: Propagador orbital. Simulação. Mecânica celeste. C++. Integração numérica. Detritos. Detritos espaciais. Mecânica orbital. Problema de quatro corpos. P4C. Perturbações luni-solares. Perturbação de pressão de radiação solar.

Abstract

There are many debris in high Earth orbit, representing hazard to vehicles, in-operation satellites, specially climate and communication ones, and others still not launched. It is important, then, to study the space debris behavior. In this work, it was built an orbit evolution simulator in C++, using as reference the simulator of orbits developed by Celestino (2005) in Fortran, to obtain the space debris orbit evolution in high earth orbit subjected to solar radiation pressure and gravitational from moon and sun perturbations. It was utilized the movement equations modeled for each perturbation and the numeric integrator integrate_const from Boost libraries, to C++, in order to perform the orbit propagation. The objective was to use the developed orbital propagator to describe the debri behavior, obtaining coherent numeric results.

Keywords: Orbital propagator. Simulation. Celestial mechanics. C++. Numeric integration. Debris. Space debris. Orbital mechanics. Four body problem. Luni-solar perturbations. Solar radiation pressure perturbation.

Lista de abreviaturas e siglas

cte	Constante
EDO	Equação Diferencial Ordinária
ED	Equações Diferenciais
f90	Fortran 90
f95	Fortran 95
HEO	High Earth Orbit
LEO	Low Earth Orbit
MEO	Mean Earth Orbit
MC	Mecânica Celeste
MO	Mecânica Orbital
PRS	Pressão de Radiação Solar
P2C	Problema de 2 Corpos
P4C	Problema de 4 Corpos
POC	Propagador Orbital em C++
POF	Propagador Orbital em Fortran
SGDC	Satélite Geoestacionário de Defesa e Comunicações Estratégicas

Lista de símbolos

ϵ'	Ângulo da eclíptica com relação ao plano equatorial
A	Área
x	Componente x da posição orbital
v_x	Componente x da velocidade orbital
y	Componente y da posição orbital
v_y	Componente y da velocidade orbital
z	Componente z da posição orbital
v_z	Componente z da velocidade orbital
$x_{k,ji}$	Componente da posição do i-ésimo corpo relativa ao j-ésimo corpo na direção do eixo k (i = 1, 2, 3, 4, j = 1, 2, 3, 4, $i <> j$, k = X, Y, Z)
$v_{k,ji}$	Componente da velocidade do i-ésimo corpo relativa ao j-ésimo corpo na direção do eixo k (i = 1, 2, 3, 4, j = 1, 2, 3, 4, $i <> j$, k = X, Y, Z)
h_i	Componente do momento angular específico (i = X, Y, Z)
G	Constante Universal da Gravitação = 6,6742.10^{11} $m^3/(kg\ s^2)$
τ	Tempo de passagem pelo pericentro da órbita
C	Constante
X	Eixo X no referencial inercial
Y	Eixo Y no referencial inercial
Z	Eixo Z no referencial inercial
ϵ	Energia mecânica total específica
F_g	Força gravitacional
H	Altitude
$x_{i,j}$	i-ésima componente de posição do j-ésimo corpo no sistema de coordenadas inercial (i = 1, 2, 3, 4, j = X, Y, Z)

$v_{i,j}$	i-ésima componente de velocidade do j-ésimo corpo no sistema de coordenadas inercial (i = 1, 2, 3, 4, $j = X$, Y, Z)
v	Magnitude da velocidade orbital
v_r	Magnitude da velocidade radial em uma órbita
r	Magnitude do raio orbital
h	Magnitude do vetor momento angular específico
n	Movimento médio
μ	Parâmetro Gravitacional
T	Período orbital
r_a	Raio orbital do foco ao apogeu
r_p	Raio orbital do foco ao perigeu
$OXYZ$	Sistema de Coordenadas Inercial
$Oxyz$	Sistema de Coordenadas Orbital
t	Tempo
UA	Unidade Astronômica
$\hat{i}$	Versor na direção do eixo x
$\hat{i}$	Versor na direção do eixo x
$\hat{j}$	Versor na direção do eixo y
$\hat{j}$	Versor na direção do eixo y
$\hat{k}$	Versor na direção do eixo z
$\hat{k}$	Versor na direção do eixo z
$\mathbf{h}$	Vetor momento angular específico
$\mathbf{r_i}$	Vetor posição o corpo i
$\mathbf{v_i}$	Vetor velocidade do corpo i
E	Anomalia excêntrica
M	Anomalia média

f	Anomalia verdadeira
Ω	Argumento do nodo ascendente
ω	Argumento do perigeu
N_i	Componente i do vetor da linha dos nodos ($i = X, Y, Z$)
$\dot{\mathbf{r}}_\mathbf{i}$	Derivada primeira do vetor posição do corpo i
$\ddot{\mathbf{r}}_\mathbf{i}$	Derivada segunda do vetor posição do corpo i
e	Excentricidade
i	Inclinação da órbita
N	Magnitude do vetor da linha dos nodos
a	semi-eixo maior
$\mathbf{a}_\mathbf{i}$	Vetor aceleração do corpo i
$\mathbf{e}$	Vetor excentricidade
$\frac{d\mathbf{X}}{dt}$	Derivada do vetor de estado
UC	Unidade de Comprimento. 1 $UC = 384400\ km$
UM	Unidade de Massa. 1 $UM = 6{,}04768 \times 10^{24}\ kg$
UT	Unidade de Tempo. 1 $UT = 86400\ s$
$\mathbf{N}$	Vetor da linha dos nodos
$\mathbf{X}$	Vetor de Estado
$\mathbf{X_0}$	Vetor de Estado Inicial
$x0$	Vetor de estado inicial na sintaxe do C++
$\hat{a}_{k,4,PRS}$	Componente do versor de aceleração de perturbação de pressão de radiação solar na direção do eixo k ($k = X, Y, Z$)
A_s	Área do corpo sobre a qual os fótons da pressão de radiação solar incidem
w	Constante Solar = 1360 W/m^2
O	Termos de ordens superiores
$x(0)$	Valor da função x no tempo $t = 0$

$\hat{x}$	Valor médio de x
$\hat{y}$	Valor médio de y
c	Velocidade da Luz = 3E+8 m/s
$a_{\hat{PRS}}$	Versor da aceleração de perturbação de pressão de radiação solar
Θ	Ângulo entre o versor normal da superfície iluminada do detrito e a linha de incidência de luz solar
r'	Coeficiente de reflectividade
γ	Coeficiente de umbra e penumbra
$\rho_{x,y}$	Correlação de Pearson
$\dot{g}(x,t)$	O ponto sobre g representa a derivada de g no tempo
$g'(x,t)$	A linha em g representa a derivada de g com relação a x
ϕ	Razão Área-Massa
n_{passos}	Número total de passos de integração na sintaxe do C++
dt	Passo de integração
$t1$	Tempo final de integração na sintaxe do C++
$t0$	Tempo inicial de integração na sintaxe do C++

Sumário

1 Introdução Contextualizada

A Guerra Fria foi uma competição entre os Estados Unidos (EUA) e a antiga União Soviética (URSS) nas mais variadas áreas e escalas. A Corrida Espacial foi a parte dessa competição que focou no desenvolvimento científico-tecnológico do setor aeroespacial. A largada foi dada em 1957, quando a URSS pôs em órbita o satélite Sputnik I, que permaneceu em voo por 21 dias antes de se desintegrar na atmosfera da Terra. Desde então, aproximadamente 28000 objetos foram postos em órbita, dos quais cerca de 9000 ainda estão no espaço. Desses, 25% estão em operação, segundo Brito et al. (2013).

Corpos em órbita estão sujeitos às intempéries do ambiente espacial e podem permanecer em órbita por variados períodos, dependendo da altitude. De acordo com Brito et al. (2013), detritos com menos de 10 cm de diâmetro podem levar, para serem removidos: dias, para altitudes menores que 200 km; anos, para altitudes entre 200 km e 600 km; centenas de anos para altitudes entre 600 km e 800 km; e podem ser permanentes para altitudes maiores que 36000 km. Essas diferenças de tempo até remoção relacionam-se com o fato de que as perturbações que um detrito espacial sofre podem variar com a altitude em que esse se encontra, dentre outros fatores.

Velocidades orbitais são elevadas, da ordem de quilômetros por segundo, o que pode ser verificado em Curtis (2005). Corpos em velocidades tão altas possuem muita energia cinética e podem causar danos catastróficos a satélites, caso ocorram colisões, como os casos do Ariane I contra o Celise, em 1996, ou do Iridium 3 contra o Cosmos 2251, em 2009 (BRITO et al., 2013). Colisões como essas e outros tipos de fragmentação geram um fenômeno chamado de *nuvens de detritos*, causando a poluição da vizinhança onde haviam sido originalmente alocados e, atualmente, não há maneiras efetivas de lidar com esse tipo de poluição espacial (BRITO et al., 2013). Celestino (2005) estudou a evolução orbital desse tipo de nuvem, para detritos que orbitam a Terra, sujeitos a diversas perturbações, investigando o efeito global das perturbações sobre os detritos.

Devido à importância desse tema, diversos trabalhos podem ser encontrados na literatura. Colombo et al. (2011) estudaram a dinâmica orbital de espaçonaves com grandes razões área-massa sob o efeito de J2 e pressão de radiação solar. Foram considerados em seus estudos pequenos satélites montados em *chips* eletrônicos, ou *spacechips*, como sendo os corpos com grandes razões área-massa. Colombo e McInnes (2011) também investigaram como perturbações devidas à pressão de radiação solar assimétrica, na presença da sombra da Terra, e de arrasto atmosférico, podem ser balanceadas para obter órbitas de longa vida para satélites em micro escala, chamados de *smart dust* (poeira inteligente), sem uso de controle ativo.

Ainda no campo de perturbações devidas à pressão de radiação solar, Valk et al. (2007) estudaram de maneira analítica e semi analítica o movimento orbital de um detrito espacial sujeito a pressão de radiação solar direta e orbitando o anel geostacionário terrestre. Fieseler (1998) investigou maneiras de utilizar velas solares em órbitas baixas, propondo uma nova configuração de vela solar capaz de fazê-lo para órbitas de alta inclinação; para tanto, usou um tipo de arrasto modelado por pressão de radiação solar. Moraes (1981) investigou os efeitos de pressão de radiação solar direta e do arrasto atmosférico para satélites terrestres.

Pardal et al. (2011) analisaram a modelagem da órbita de um satélite artificial de GPS, com o intuito de verificar como diferentes modelagens podem afetar a precisão da determinação da órbita. Isso foi feito considerando os efeitos da perturbação geopotencial de alta ordem e e grau; a pressão de radiação solar direta; as perturbações gravitacionais da Lua e do Sol (luni-solares). Segundo os autores, de maneira geral, as perturbações luni-solares tendem a provocar efeitos no comportamento dos argumentos do nodo ascendente e do perigeu, gerando precessão; e a de pressão de radiação solar provoca perturbações periódicas e seculares nos argumentos do nodo ascendente, do perigeu e na anomalia média, além de perturbações periódicas no semieixo maior, na excentricidade e na inclinação.

Vista a quantidade de objetos em órbita da Terra e considerando que o espaço orbital, apesar de extenso, é finito, é importante compreender como as perturbações do ambiente espacial afetam o movimento orbital dos corpos orbitantes, que, neste trabalho, foram considerados detritos espaciais. As perturbações analisadas foram as provocadas pela radiação solar e pelas atrações gravitacionais luni-solares. Em posse deste conhecimento, formula-se que o problema tratado neste trabalho foi o de quatro corpos, tratamento esse realizado por meio de simulação computacional.

O objetivo deste trabalho foi programar um propagador orbital na linguagem de programação C++, utilizando bibliotecas auxiliares de álgebra linear para construir o vetor de estado inicial da integração e bibliotecas de integração numérica para propagar movimento orbital. O simulador construído teve outro como referência, o de Celestino (2005), escrito em Fortran, cujas entradas e resultados foram aproveitados para padronizar o vetor de estado inicial e validar os resultados da integração numérica no C++.

A etapa de desenvolvimento numérico, neste trabalho, resultou na construção do simulador em C++ e em sua validação. A primeira parte do trabalho foi compreender o programa em Fortran, para dar referências à programação no C++. Ao mesmo tempo os estudos de fundamentação teórica foram conduzidos, sendo escrita esta seção na monografia.

Após o processo de simulação em Fortran ter sido compreendido, deu-se início à transcrição das entradas do Fortran para o C++, com objetivo de padronizar o vetor de estado inicial. Neste momento foram procuradas as bibliotecas auxiliares para realizar a construção do vetor de estados e executar a integração numérica.

Ao ser conseguido um vetor de estado padronizado, foram implementadas as equações da dinâmica do problema de quatro corpos, portanto, considerando o movimento do detrito ao redor da Terra perturbado pela Lua e pelo Sol. Esse código foi validado com os resultados do propagador em Fortran, o que foi feito com um *script* programado em Python para correlacionar os resultados dos dois simuladores. Depois, foi implementada a pressão de radiação solar com base em Fieseler (1998). Esses resultados também foram validados. Foi construído, então, um *script* gerador de gráficos a partir dos resultados, ainda no Python, usando a biblioteca Matplotlib. Por fim, foram simuladas diferentes condições orbitais.

Esta monografia conta com seis capítulos. O primeiro capítulo, este, introduz o leitor aos temas correlatos e trabalhados. O Capítulo 2 apresenta de maneira detalhada os objetivos do projeto. O Capítulo 3 discute as bases teóricas usadas neste trabalho. O Capítulo 4 mostra passo a passo como foi construído e validado o simulador orbital em C++. O Capítulo 5 apresenta a validação do propagador orbital desenvolvido em C++ com relação aos resultados do propagador em Fortran, de Celestino (2005), além das condições iniciais de simulação e os resultados de cada uma delas de maneira gráfica. O Capítulo 6 dá as conclusões do trabalho e algumas direções para trabalhos futuros sobre a ferramenta desenvolvida. Há, além dos seis capítulos, um apêndice que apresenta o método de Curtis (2005) para trabalhar com elementos orbitais.

2 Objetivos

O objetivo geral deste trabalho foi desenvolver um simulador computacional em C++ para realizar a propagação do movimento orbital de detritos espaciais. Os detritos foram submetidos a dois tipos de perturbação: gravitacional, do problema de 4 corpos, e da pressão de radiação solar. O interesse principal foi estudar o comportamento dos detritos perturbados em órbita alta.

Para atingir o objetivo geral, este trabalho foi dividido em 10 objetivos específicos:

1. compreender o propagador orbital de Celestino (2005), construído em Fortran;
2. encontrar métodos para auxiliar a manipulação de vetores e matrizes no C++;
3. transcrever as entradas do simulador em C++ a partir das entradas do código em Fortran;
4. encontrar métodos para integrar as equações diferenciais em C++;
5. implementar em C++ as equações dinâmicas do problema de 4 corpos considerando apenas perturbações luni-solares sobre o detrito;
6. validar os resultados obtidos com o C++ para as perturbações luni-solares, com referência nos resultados do propagador em Fortran;
7. implementar a função de perturbação de radiação solar no propagador em C++;
8. validar os resultados obtidos com o C++ para a perturbação de pressão de radiação solar, com referência nos resultados do propagador em Fortran;
9. utilizar o propagador desenvolvido para simular órbitas teste em diversas condições e para um caso real, considerado o Satélite Geoestacionário de Defesa e Comunicações Estratégicas (SGDC);
10. documentar o projeto.

Este documento descreve as atividades realizadas para desenvolver um simulador de evolução orbital. Esta é uma aplicação numérica do problema de 4 corpos. Todo o trabalho realizado foi feito com ferramentas *open-source* ou em versões gratuitas. Espera-se que estudantes, professores, pesquisadores, engenheiros ou entusiastas, tenham acesso a essa ferramenta e sirvam-se bem de suas funcionalidades.

3 Fundamentos Teóricos

Eeste documento foi baseado nas teorias e aplicações da Mecânica Celeste (MC). Duas fontes principais serviram de fundamentos, em especial para este capítulo: *Orbital Mechanics for Engineering Students* (CURTIS, 2005), livro didático que instrui o leitor nos meandros da MC; e Estudo da Dinâmica de Pequenos Detritos Espaciais e Meteoroides de Celestino (2005), que é um estudo avançado sobre a evolução orbital de detritos em diversas condições, sujeitos a diversas perturbações.

3.1 Interpretação das Leis de Kepler para o Movimento Planetário

As Leis de Kepler, que regem o movimento planetário, são apresentadas a seguir.

- (Primeira Lei) Os planetas se movem em órbitas elípticas com o Sol em um dos focos.

  A Equação 3.1 descreve o movimento em uma trajetória elíptica de um corpo em órbita, conforme dito na Primeira Lei. Nesta equação, r é magnitude da posição orbital medida a partir do foco, e é a excentricidade e f é a anomalia verdadeira.

$$r = \frac{a(1-e^2)}{1+e\cos f} \tag{3.1}$$

- (Segunda Lei) Em suas órbitas ao redor do Sol, os planetas varrem áreas iguais em tempos iguais.

  A Equação 3.2 é uma interpretação da Segunda Lei. Esta equação descreve que a velocidade areolar (a velocidade de varredura de área), isto é, a derivada da área A, varrida pelo vetor posição em relação ao tempo t, é igual à metade da magnitude momento angular h, que é constante em relação ao tempo.

$$\frac{dA}{dt} = \frac{h}{2} = cte \tag{3.2}$$

- (Terceira Lei) O quadrado do período para completar uma órbita é proporcional ao cubo da distância média em relação ao Sol.

  A Terceira Lei de Kepler é descrita pela Equação 3.3, que diz que o quadrado do período orbital T é proporcional ao cubo do semieixo maior a, por uma constante

que é, por sua vez, proporcional a 2π e inversamente proporcional ao parâmetro gravitacional μ.

$$T^2 = \frac{2\pi}{\mu} a^3 \tag{3.3}$$

A Terceira Lei de Kepler merece ainda um comentário extra. Essa lei foi descoberta em 15 de maio de 1618, por Johanes Kepler, e foi publicada no ano de 1619, em seu trabalho *Harmonices Mundi*. Foi essa lei, não uma maçã, que levou Isaac Newton à sua Lei da Gravitação Universal, que descreve que a força de atração gravitacional F_g, entre duas massas m_1 e m_2, é proporcional à Constante Universal da Gravitação G ($G = 6{,}6742.10^{11}$ $m^3/(kg\, s^2)$) e inversamente proporcional ao quadrado da distância r entre as massas. A Equação 3.4 descreve F_g.

$$F_g = G\frac{m_1 m_2}{r^2} \tag{3.4}$$

3.2 Problema Reduzido de Dois Corpos

A Figura 1 ilustra um corpo 2 de massa m_2 orbitando um corpo 1 de massa m_1. Este é conhecido como o problema clássico de dois corpos (P2C) (CURTIS, 2005), que pode fundamentar a compreensão do problema de 4 corpos (P4C), foco deste trabalho. No P2C, por interação gravitacional, m_2 movimenta-se em torno de m_1 ($m_1 > m_2$), percorrendo a órbita mostrada com determinada velocidade, a depender do vetor posição $\mathbf{r}$. À medida que m_2 afasta-se de m_1, a velocidade orbital diminui, até atingir o ponto mais distante de m_1, chamado de apoapsis, onde a velocidade de m_2 é mínima. Então, m_2 volta a aproximar-se do corpo 1, ganhando velocidade. Quando m_2 chega à periapsis, ponto orbital mais próximo de m_1, sua velocidade é máxima. As grandezas perigeu r_p e apogeu r_a são casos particulares, respectivamente, da periapsis e da apoapsis, visto que enquanto o sufixo "apsis" serve para quaisquer corpos celestes, o sufixo "geu" refere-se a órbitas ao redor da Terra. Fosse considerada uma órbita em torno do Sol, os mesmos seriam chamados de periélio e afélio (CURTIS, 2005).

A órbita elíptica, apresentada na Figura 2, é uma representação do problema de dois corpos de um ponto de vista perpendicular à órbita, tal que esta reside no plano xy. Nesta figura, m_1 está localizado no foco da elipse. É onde também estão os eixos coordenados xy. A posição angular de m_2, em relação a m_1, é dada pela anomalia verdadeira f, definida como o ângulo entre o eixo x e o vetor da posição orbital. Podem ser vistos, dispostos no desenho, o semieixo maior a, o semieixo menor b, a distância até o foco é dada por c e o vetor posição orbital $\mathbf{r}$.

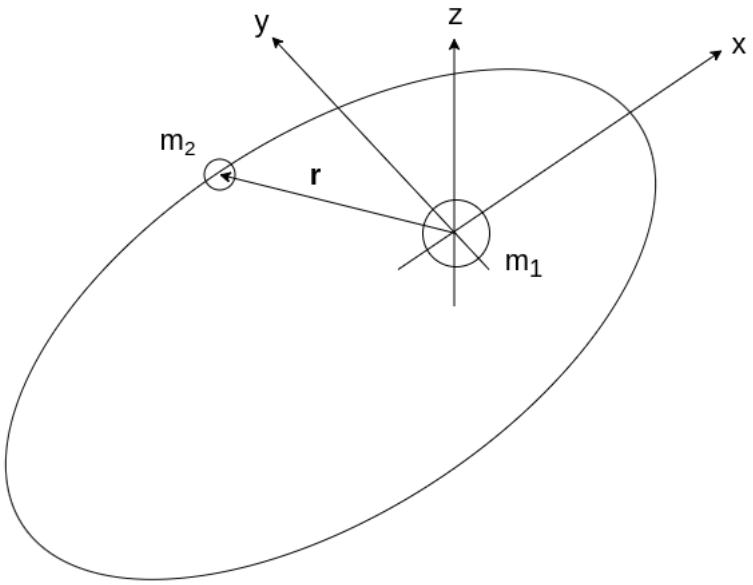

Figura 1 – Problema de dois corpos. O corpo de massa m_2 orbita o corpo de massa m_1.

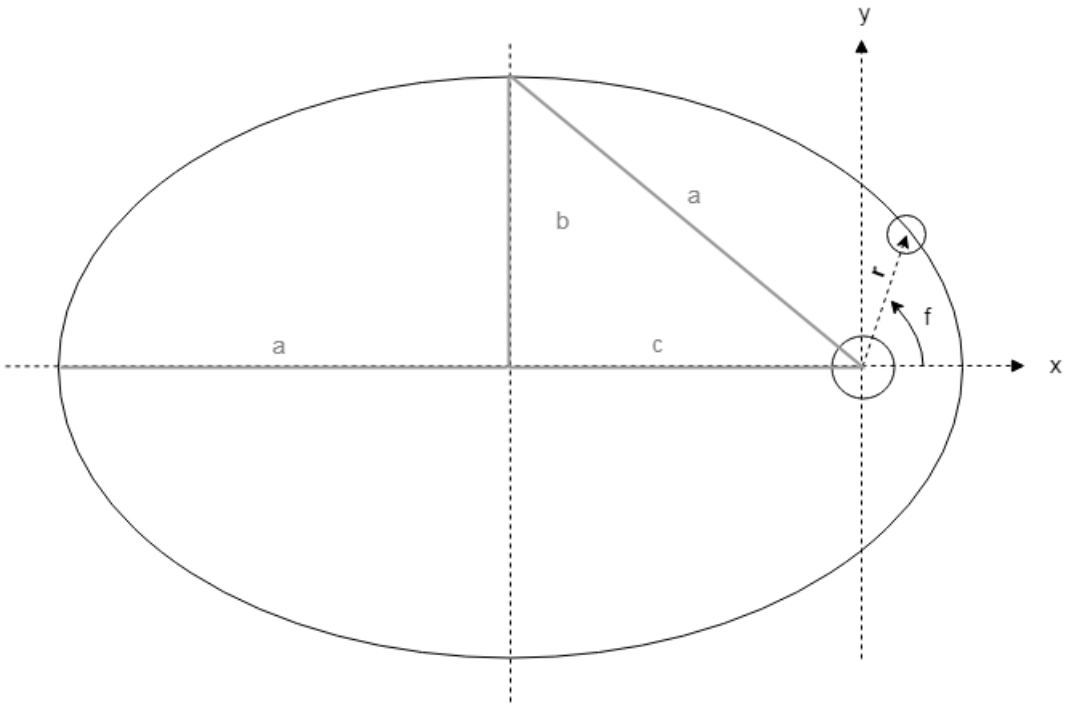

Figura 2 – Problema de dois corpos, vista superior. O corpo de massa m_2 orbita o corpo de massa m_1.

Tendo em vista a Figura 2, retorne à Equação 3.1 de uma órbita elíptica. Ela descreve, em coordenadas polares, a magnitude do vetor posição orbital r em função de três parâmetros orbitais, o semieixo maior, a excentricidade e a anomalia verdadeira. O

movimento representado por esta equação é plano e elíptico, de um corpo em relação a outro. Note que a é a medida do centro da elipse até seus pontos extremos mais afastados, de forma que de apogeu a perigeu a órbita mede $2a$. Com base nisso, uma maneira de determinar o semieixo maior a partir de r_p e r_a é dada pela Equação 3.5. O semieixo maior fornece a noção de tamanho à órbita. Quanto maior for o valor de a, maior será a órbita.

$$a = \frac{r_p + r_a}{2} \tag{3.5}$$

Outro parâmetro importante mostrado na Equação 3.1 é a excentricidade, definida por $e = c/a$, que, para órbitas elípticas, tem valores que seguem a condição $0 \leq e < 1$. Essa grandeza é o que define a forma da órbita, se a elipse é mais ou menos alongada. Dentro desse intervalo, quanto maior a excentricidade, mais alongada é a elipse que representa a órbita e, quanto menor for seu valor, menos alongada. Se a excentricidade for igualada a zero, a órbita será circular. A Equação 3.6 mostra como calcular a excentricidade em função de c e a e de r_a e r_p (CURTIS, 2005).

$$e = \frac{c}{a} = \frac{r_a - r_p}{r_a + r_p} \tag{3.6}$$

No caso do P2C da Figura 3, m_2 é o corpo orbitante e m_1 o orbitado, mas, diferente do que já foi apresentado, as posições orbitais dos dois corpos são tomadas com relação a um referencial inercial $OXYZ$, não em relação a um referencial orbital $Oxyz$. Em relação a $OXYZ$, a posição de m_1 é dada pelo vetor $\mathbf{r_1}$ e de m_2 pelo vetor $\mathbf{r_2}$.

A força de atração gravitacional $\mathbf{F_{21}}$ entre dois corpos (m_1 e m_2) é dada pela Equação 3.7. Nela, $\ddot{\mathbf{r}}_\mathbf{2}$ é a aceleração de m_2, medida em relação ao referencial inercial, e o versor da posição do corpo 1 em relação ao corpo 2 é $\hat{\mathbf{r}}_\mathbf{21}$. G é Constante Universal da Gravitação.

$$\mathbf{F_{21}} = m_2\ddot{\mathbf{r}}_\mathbf{2} = G\frac{m_1 m_2}{||\mathbf{r_{12}}||^2}\hat{\mathbf{r}}_\mathbf{21} \tag{3.7}$$

A Equação 3.8 foi obtida dividindo a Equação 3.7 pela massa m_2. μ_1 é o parâmetro gravitacional do corpo 1.

$$\ddot{\mathbf{r}}_\mathbf{2} = \frac{Gm_1}{||\mathbf{r_{21}}||^3}(\mathbf{r_1} - \mathbf{r_2}) = \frac{\mu_1}{||\mathbf{r_{21}}||^3}(\mathbf{r_1} - \mathbf{r_2}) \tag{3.8}$$

Nesta equação, a posição do corpo 1 em relação ao corpo 2, ou o negativo da posição do corpo 2 em relação o corpo 1, é dada por

$$\mathbf{r_{21}} = \mathbf{r_1} - \mathbf{r_2} = -(\mathbf{r_2} - \mathbf{r_1}) = -\mathbf{r_{12}}$$

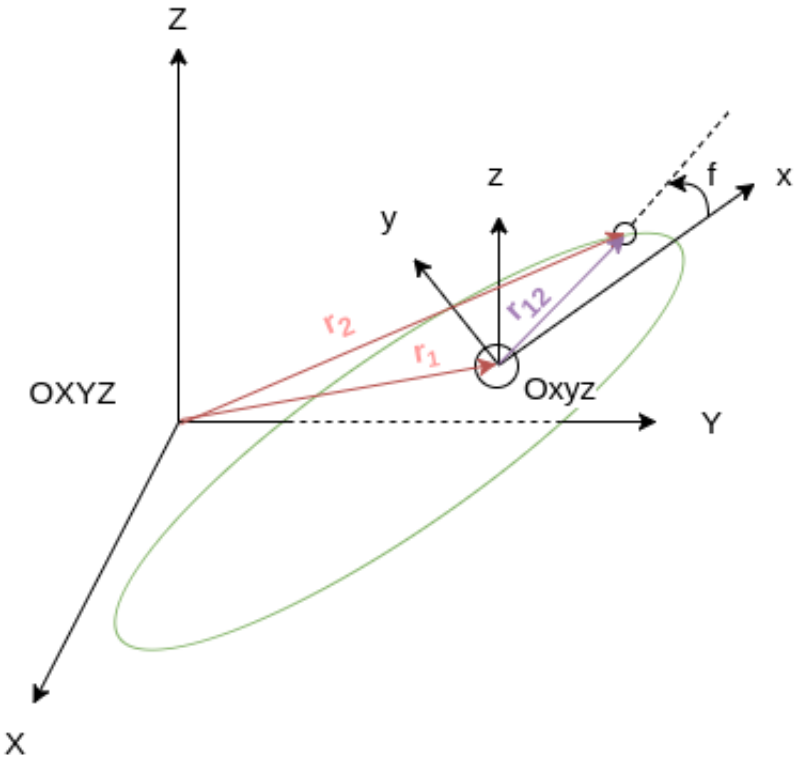

Figura 3 – Problema de dois corpos, as posições dos corpos são medidas com relação a um referencial inercial $OXYZ$. O corpo 2, de massa m_2, orbita o corpo 1, de massa m_1. O referencial orbital $Oxyz$ está fixado no corpo 1.

A Equação 3.9 pode ser obtida a partir da Terceira Lei de Kepler. Essa equação descreve período orbital T de uma órbita elíptica.

$$T = \frac{2\pi}{\sqrt{\mu}} a^{3/2} \tag{3.9}$$

Órbitas elípticas não possuem movimento uniforme, visto que o corpo que orbita acelera da apoapsis à periapsis e desacelera da periapsis à apoapsis (CURTIS, 2005). Entretanto, é possível calcular um movimento médio n, dado pela Equação 3.10.

$$n = \frac{2\pi}{T} \tag{3.10}$$

A ideia do movimento médio funciona como se o vetor posição orbital tivesse magnitude constante e fosse medido a partir do centro da órbita, não do foco da elipse, gerando uma órbita circular imaginária que circunscreve a órbita elíptica, conforme é mostrado na Figura 4. A variável f é a anomalia verdadeira. A variável E é a anomalia excêntrica, que é o ângulo do raio orbital, medido a partir do centro da órbita, mas que intersecta a órbita circular imaginária conforme mostrado. Há ainda uma terceira anomalia, chamada de anomalia média M, que é uma posição angular imaginária em relação ao centro, se o corpo orbitante percorresse o círculo com movimento uniforme.

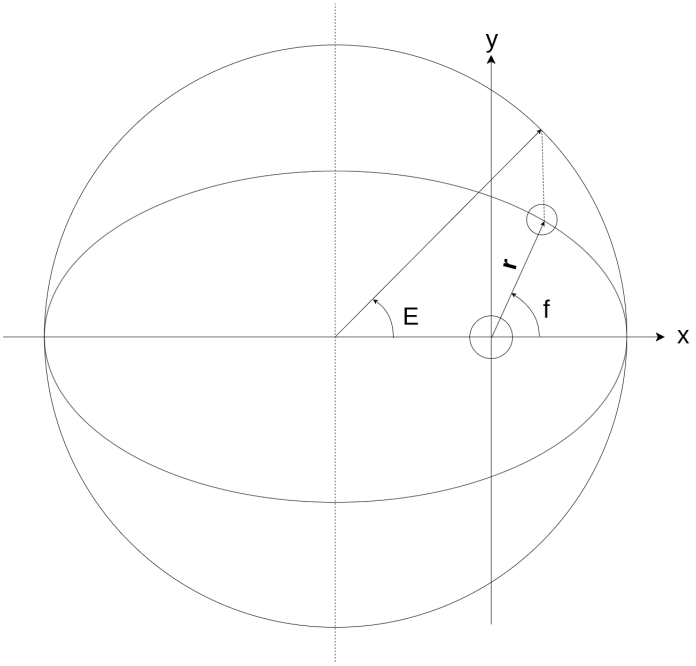

Figura 4 – Anomalias: verdadeira (f) e excêntrica (E).

De acordo com Curtis (2005), a relação entre E e f é dada pela Equação 3.11.

$$\mathrm{tg}^2\frac{f}{2} = \frac{1-e}{1+e}\,\mathrm{tg}^2\frac{E}{2} \tag{3.11}$$

A relação entre M e E é descrita pela Equação 3.12. Nesta equação, t é o tempo percorrido em órbita.

$$M = E - e\,\mathrm{sen}E = n(t-\tau) \tag{3.12}$$

sendo τ o tempo de passagem pela periapsis.

A Equação 3.12 é conhecida como Equação de Kepler e é transcendental, visto que não é possível isolar a variável E. Portanto, deve-se resolvê-la numericamente. O Método de Newton-Raphson, para estimar raízes de equações, aplicado à Equação 3.12, é mostrado na Equação 3.13 e serve para estimar E de maneira iterativa. As iterações são contabilizadas pelos índices (i) (iteração corrente) e $(i+1)$ (próxima iteração) (CURTIS, 2005).

$$E_{i+1} = E_i - \frac{E_i - e\ \mathrm{sen}E_i - M}{1 - e\ \cos E_i} \tag{3.13}$$

De acordo com Curtis (2005), duas grandezas físicas se conservam em uma órbita. São elas o momento angular específico $\mathbf{h}$ e a energia mecânica total específica ϵ ("específico" equivale a "por unidade de massa").

A grandeza **h** pode ser calculada pela Equação 3.14. Esta equação mostra as componentes x, y e z de **h**, nas direções, respectivamente, dos versores $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$ e $\hat{\mathbf{k}}$; o vetor posição orbital, representado por **r**; e o vetor velocidade orbital $\mathbf{v} = \dot{\mathbf{r}}$.

$$\mathbf{h} = [h_X\hat{\mathbf{i}} + h_Y\hat{\mathbf{j}} + h_Z\hat{\mathbf{k}}] = \mathbf{r} \times \mathbf{v} = \mathbf{r} \times \dot{\mathbf{r}} \tag{3.14}$$

A Equação 3.15 relaciona a energia mecânica total específica com a velocidade orbital v, o parâmetro gravitacional μ e a magnitude da posição orbital r. Ela recebe o nome de *vis-viva*, que significa "força viva".

$$\epsilon = \frac{v^2}{2} - \frac{\mu}{r} \tag{3.15}$$

A magnitude da velocidade orbital v pode ser obtida pela Equação 3.16.

$$v = \sqrt{\mu\left[\frac{2}{r} - \frac{1}{a}\right]} \tag{3.16}$$

De acordo com Curtis (2005), vale a relação da Equação 3.17 para órbitas elípticas e circulares.

$$\epsilon = -\mu/(2a) \tag{3.17}$$

Órbitas podem ser classificadas em termos de suas altitudes. Para órbitas terrestres, se a altitude for entre 180 km e 2000 km, esta será uma órbita baixa (*low Earth orbit*, LEO). Uma órbita de altitude variando entre 2000 km e cerca de 36000 km é considerada média (*mid Earth orbit*, MEO). Para altitudes maiores que 36000 km, a órbita é classificada como alta (*high Earth orbit*, HEO). Exemplos de satélites em diferentes órbitas são: a Estação Espacial Internacional (ISS), que ocupa uma órbita LEO; satélites de GPS, característicos de órbitas MEO; e satélites de comunicação, que normalmente ocupam órbitas HEO (RIEBEEK, 2009).

Há um tipo particular de órbita HEO que merece atenção. Esse é o caso da órbita geossíncrona, na qual um satélite tem período orbital de um dia. Um caso particular de órbita geossíncrona é a órbita geoestacionária (GEO), cuja trajetória se dá sobre o plano equatorial da Terra. Como a Terra dá, em torno de si mesma, uma volta por dia, um satélite em órbita GEO ficará, relativamente a um ponto da superfície terrestre, sempre numa mesma posição. Isto quer dizer que caso um satélite seja posto em órbita GEO sobre o Brasil, como por exemplo o SGDC, o mesmo orbitará em torno da Terra sempre em cima do país. Esse é um dos motivos pelos quais essas órbitas são visadas para satélites de comunicação. A altitude de uma órbita GEO é de cerca de 36000 km (CURTIS, 2005; RIEBEEK, 2009).

3.3 Elementos Orbitais

De acordo com Curtis (2005), são necessários 6 elementos para determinar uma órbita. Esses são chamados de Elementos Orbitais, um conjunto formado pelas variáveis: a, e, inclinação i, argumento do nodo ascendente Ω, argumento do perigeu ω e f. As grandezas i, Ω, ω e f são mostradas na Figura 5. Os elementos orbitais Ω, i e ω são Ângulos de Euler (CURTIS, 2005) e descrevem rotações que a órbita sofre no espaço com relação a um referencial inercial.

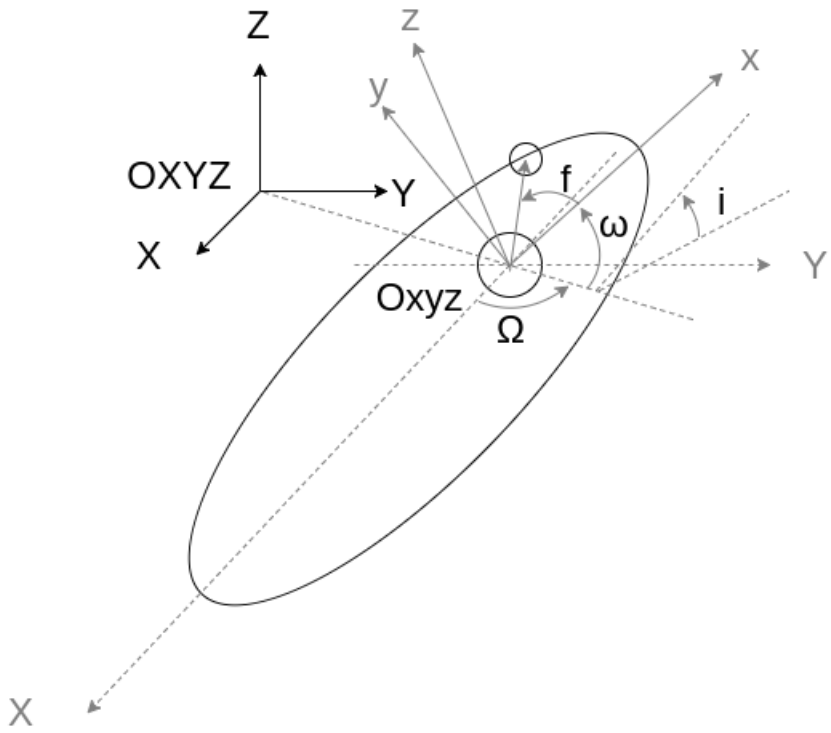

Figura 5 – Os elementos orbitais angulares: inclinação i, argumento do nodo ascendente Ω, argumento do perigeu ω e anomalia verdadeira f. Além desses, não mostrados na figura, há a e e.

O elemento orbital Ω é o ângulo entre a linha da intersecção do plano orbital com o plano XY, chamada de linha dos nodos, e o eixo X inercial; a rotação que gera Ω é em torno do eixo Z inercial. O elemento i inclina a órbita com relação ao plano XY, por uma rotação em torno do eixo y orbital. ω atua sobre a direção do perigeu, rotacionando a órbita em torno do eixo z orbital.

3.4 Sistemas de Coordenadas e Rotações

É possível definir os dois sistemas de coordenadas usados neste trabalho. O primeiro, sistema de coordenadas inercial, considera que X aponta na direção do equinócio vernal $\bar{\gamma}$. Essa direção é dada pela intersecção do plano eclíptico com o equatorial terrestre, que

diferem um do outro pelo ângulo $\epsilon' = 23{,}4^o$ (CURTIS, 2005). O eixo Z aponta para a direção do eixo de rotação da Terra ao redor do Sol, normal ao plano eclíptico. O eixo Y completa o sistema de coordenadas $OXYZ$.

O segundo sistema de coordenadas usado neste trabalho é o orbital, em que o eixo x aponta para a direção da periapsis, a partir do foco. O eixo z é normal ao plano da órbita, cujo sentido positivo aponta na direção do vetor momento angular. O eixo y completa o sistema.

Pode haver a necessidade representar vetores ora em $OXYZ$, ora em $Oxyz$. A rotação entre um e outro pode ser feita multiplicando vetores, representados em um referencial, por matrizes de rotação que levem ao outro referencial (CURTIS, 2005). As Equações 3.18, 3.19 e 3.20 representam as matrizes de rotação $\mathbf{R_i}(\theta)$, que efetuam a rotação em torno no eixo i por um ângulo θ.

$$\mathbf{R_x}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \text{sen}\theta \\ 0 & -\text{sen}\theta & \cos\theta \end{bmatrix} \tag{3.18}$$

$$\mathbf{R_y}(\theta) = \begin{bmatrix} \cos\theta & 0 & -\text{sen}\theta \\ 0 & 1 & 0 \\ \text{sen}\theta & 0 & \cos\theta \end{bmatrix} \tag{3.19}$$

$$\mathbf{R_z}(\theta) = \begin{bmatrix} \cos\theta & \text{sen}\theta & 0 \\ -\text{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

De acordo com Curtis (2005), matrizes de rotação são ortogonais, então $\mathbf{R}\,\mathbf{R}^T = \mathbf{I}$, $\mathbf{I}$ é a matriz identidade. Isso implica que a transposta da matriz $\mathbf{R}$ é igual à sua inversa, então $\mathbf{R}^T = \mathbf{R}^{-1}$. Se a matriz $\mathbf{R}$ leva do referencial orbital para o inercial, sua inversa faz operação contrária, levando do referencial inercial para o orbital. Multiplicar matrizes de rotação gera uma única matriz, que produz uma rotação combinada equivalente à rotação de cada matriz.

3.5 Problema de N Corpos

A teoria apresentada até aqui foi baseada P2C. A dinâmica desse sistema conta apenas com o movimento de duas massas e com o efeito gravitacional de uma massa em relação a outra, motivo pelo qual o P2C possui limitações; afinal, existem mais que dois corpos no espaço. Um sistema formado por Sol, Terra e Lua, por exemplo, é um sistema dinâmico formado por três corpos ($P3C$), caso em que a modelagem do P2C deixa de

funcionar. De fato, o $P3C$ não tem solução analítica (HEGGIE, 2005), como é feito no P2C, exceto no caso restrito do problema de três corpos, conforme foi apresentado em Curtis (2005).

Considere o caso em que há um número N de corpos em órbitas, interagindo, entre si, classicamente, com base na mecânica de Newton e em suas leis de gravitação. Este é o problema de N corpos (PNC). A modelagem da dinâmica desse sistema é dada pelas Equações 3.21. Nesta equação, $\ddot{\mathbf{r}}_\mathbf{i}$ é o vetor aceleração do i-ésimo corpo, μ_j é o parâmetro do j-ésimo corpo, $\mathbf{r_i}$ é o vetor posição do i-ésimo corpo e $\mathbf{r_j}$ é o vetor posição do j-ésimo corpo (CELESTINO, 2005).

$$\ddot{\mathbf{r}}_\mathbf{i} = -\sum_{j=1, j\neq i}^{N} \mu_j \frac{\mathbf{r_i} - \mathbf{r_j}}{||\mathbf{r_i} - \mathbf{r_j}||^3} \tag{3.21}$$

Essas equações fornecem um modelo matemático aproximado com inúmeras aplicações em astrofísica. Isso inclui a dinâmica dos planetas do Sistema Solar, de estrelas, *clusters* de estrelas, galáxias, até mesmo o comportamento de partículas de matéria escura. O problema tratado neste trabalho foi o de quatro corpos P4C, que é um caso particular do PNC, considerando Sol, Terra, Lua e detrito.

3.6 Perturbações Orbitais

Uma perturbação, de acordo com Shamsul et al. (2017), é algo que provoca um desvio num movimento teórico ou previsto. O espaço sideral não é um ambiente livre de perturbações, de forma que corpos em órbita sofrem seus efeitos (RIEBEEK, 2009; CELESTINO, 2005). Neste trabalho foram levadas em conta as perturbações luni-solares e de pressão de radiação solar (PRS).

Considere o sistema dinâmico formado por Sol (1), Terra (2), Lua (3) e detrito (4). Neste sistema, o detrito sofre influência de perturbações gravitacionais dos outros três corpos. Ele também influencia os três, apesar de a sua influência ser desprezível, se comparada com a que sofre, tendo em vista o fato de que sua massa é muito menor. Segundo Celestino (2005), a força $\mathbf{F_4}$ sofrida pelo detrito, provocada pelos outros corpos, pode ser representada pela Equação 3.22. O tipo de perturbação nesta equação de movimento é chamado de luni-solar. Segundo Pardal et al. (2011), essa perturbação atua principalmente sobre Ω e ω, o que provoca precessão da órbita e do plano orbital.

$$\mathbf{F_4} = \frac{\mu_1}{\mathbf{r_{41}^3}}(\mathbf{r_1} - \mathbf{r_4}) + \frac{\mu_2}{\mathbf{r_{42}^3}}(\mathbf{r_2} - \mathbf{r_4}) + \frac{\mu_3}{\mathbf{r_{43}^3}}(\mathbf{r_3} - \mathbf{r_4}) \tag{3.22}$$

A outra perturbação considerada neste trabalho foi a de PRS. Esta é provocada por fótons, emitidos pelo sol, interagindo com o detrito em órbita. O que acontece é que fótons,

mesmo sem massa, possuem momento. Quando estes colidem com o detrito, transferem o momento que carregam para o objeto. Se o objeto reflete completamente o fóton, o momento absorvido é máximo.

A força resultante aplicada no satélite é dada pela Equação 3.23. Nesta equação, F_{solar} é a força sofrida pelo objeto com a transferência do momento do fóton, r' é o coeficiente de reflectividade, $w = 1360\ W/m^2$ é a constante solar a 1 UA, A_s é a área sobre a qual os fótons incidem, c é a velocidade da luz e θ é o ângulo entre os fótons incidentes e o versor normal ao plano que recebe os feixes de fótons (FIESELER, 1998). Segundo Pardal et al. (2011), a PRS pode provocar perturbações seculares e periódicas nos elementos Ω, ω e M; e perturbações periódicas em a, e e i.

$$F_{solar} = \frac{(1 + r')wA_s \cos^2 \theta}{c} \tag{3.23}$$

3.7 Integração Numérica e Resolução de ODEs

Zipfel (2007) denomina como Engenharia Virtual a engenharia baseada em computadores para aumento da produtividade. Dentro do campo de engenharia virtual, a modelagem e a simulação ocupam posições importantes, permitindo que engenheiros definam o *design* e explorem o desempenho de determinado produto usando apenas computadores (ZIPFEL, 2007).

É comum se deparar, no estudo de ciências naturais ou de engenharia, com equações diferenciais (ED). Estas são equações que colocam em relação uma função e sua derivada. Formalmente, uma equação diferencial de uma variável real é uma equação em que a incógnita é uma função real. Por exemplo, o tipo mais simples de ED é

$$\dot{x}(t) = f(t)$$

isto é, o problema de encontrar a primitiva de uma função. Na notação de Leibniz, essa equação pode ser reescrita na Equação 3.24, em que $x(t)$ é a primitiva da função $f(t)$, t é a variável independente - o tempo - e C uma constante qualquer, associada à integração indefinida (ASANO; COLLI, 2009).

$$x(t) = \int f(t)dt + C \tag{3.24}$$

Definindo a integral da Equação 3.24 entre o intervalo $[t_0, t_1]$, é obtida a Equação 3.25.

$$x(t) = x(t_0) + \int_{t_0}^{t_1} f(t)dt = x_0 + \int_{t_0}^{t_1} f(t)dt \tag{3.25}$$

Este é conhecido como problema de valor inicial (PVI), ou problema de Cauchy. Esta equação diz que uma função x no tempo t é igual ao valor inicial x_0 somado à integral da função f no intervalo de tempo $[t_0, t_1]$. O problema tratado neste trabalho é semelhante ao que propõe esta equação.

O fundamental, tratando-se da solução numérica de EDs do tipo $\dot{x} = f(t)$, é a discretização da variável t. Supondo um intervalo de tempo $[a \leq t \leq b]$, a discretização deste intervalo consiste em dividi-lo em diversas partes, tal que t_k é o k-ésimo tempo no intervalo discretizado. A diferença entre t_k e t_{k+1} é chamada de passo de integração t_h. Os passos podem ser ou não constantes. Como é apresentado em Asano e Colli (2009), determinar numericamente uma função $x(t)$ significa encontrar, com precisão suficientemente boa, os valores $x_0 = x(t_0)$, $x_1 = x(t_1)$, ..., $x_N = x(t_N)$, para o intervalo de tempo discretizado em

$$a = t_0 < t_1 < ... < t_N = b.$$

A solução numérica de uma ED carregará consigo imprecisões em t, tendo em vista que a função $x(t)$ será calculada para t finito. De maneira geral, a redução do passo de interação t_h pode aumentar a precisão da solução.

A solução de equações diferenciais apresentada nesta Seção é feita por recorrência, tal que a condição inicial $x(t_0)$ é dada e, a partir dela, em sucessão, os próximos termos $x(t_k)$ são calculados. Isto pode feito por meio de uma expansão em Série de Taylor, como mostrado na Equação 3.26. O sobrescrito (m) representa a m-ésima derivada de x e a m-ésima potência de t_h. O termo $O(t_h^m)$ denota o restante da expansão e vai a zero quando t_h tende a zero. Além disso, os índices k são contabilizados de $k = 1$ a $N - 1$. Essa expressão mostra que quanto menor for t_h, melhor o polinômio (desconsiderando os termos $O(t_h^m)$) aproximará $x(t_{k+1})$. Em geral, mas não necessariamente, além de reduzir t_h, para aumentar a precisão do cálculo de $x(t_{k+1})$, pode-se aumentar m. Como ressalva, para essa aproximação funcionar, $x(t)$ tem que ser diferenciável até ordem m (ASANO; COLLI, 2009).

$$x(t_{k+1}) = x(t_k) + x^{(1)}(t_k)t_h + \frac{1}{2!}x^{(2)}(t_k)t_h^2 + ... + \frac{1}{m!}x^{(m)}(t)t_h^m + O(t_h^m) \tag{3.26}$$

Note que devem ser computadas as derivadas de x na Equação 3.26.

4 Desenvolvimento Numérico

4.1 Ponto de Partida

A intenção deste trabalho foi desenvolver uma solução computacional utilizando a linguagem de programação C++. A solução foi elaborada para ser uma ferramenta capaz de simular a evolução temporal de órbitas, tratando um problema de MC de forma numérica. A tecnologia que este trabalho desenvolveu foi chamada de Propagador Orbital em C++ (POC). De maneira geral, considerou-se que os detritos simulados tinham grandes razões área-massa ϕ e eram afetados pela atração gravitacional da Lua e do Sol, além sofrerem efeitos da PRS. Existem mais perturbações que poderiam ter sido consideradas, como, por exemplo, o arrasto atmosférico ou o achatamento terrestre, predominantes em órbitas LEO (RIEBEEK, 2009; CELESTINO, 2005). Entretanto, foram levadas em conta apenas perturbações luni-solares e de PRS, pois somente órbitas altas e de baixas excentricidades iniciais foram simuladas neste trabalho.

O trabalho desenvolvido foi baseado em Celestino (2005), em que foi produzido um Propagador Orbital em Fortran (POF). Este propagador foi usado como referência. POF tinha diversas diferenças entre o que foi construído. Algumas das diferenças entre os propagadores são o processo de armazenamento de resultados em arquivos, a linguagem de programação e o integrador numérico, entre outras minúcias.

Para construir o novo propagador orbital, o processo de desenvolvimento foi dividido em partes menores. A estrutura da decomposição do problema global em problemas locais ficou como a seguir:

1. compreensão do POF;
2. transcrição das entradas do POF para o POC;
3. implementação das equações dinâmicas e do integrador numérico do POC;
4. processamento de resultados pós-integração com o POC;
5. validação de resultados do POC.

O modelo trabalhado foi o P4C. Um esquema visual deste modelo é apresentado na Figura 6. À direita da figura é mostrado o modelo completo, com 4 corpos (1, 2, 3 e 4), suas órbitas e referenciais. À esquerda, para auxiliar a visualização são apresentados em destaque: (parte superior) a órbita do corpo 3 em torno do corpo 2, com seu referencial; e (parte inferior) a órbita do corpo 4 ao redor do corpo 2, com seu referencial.

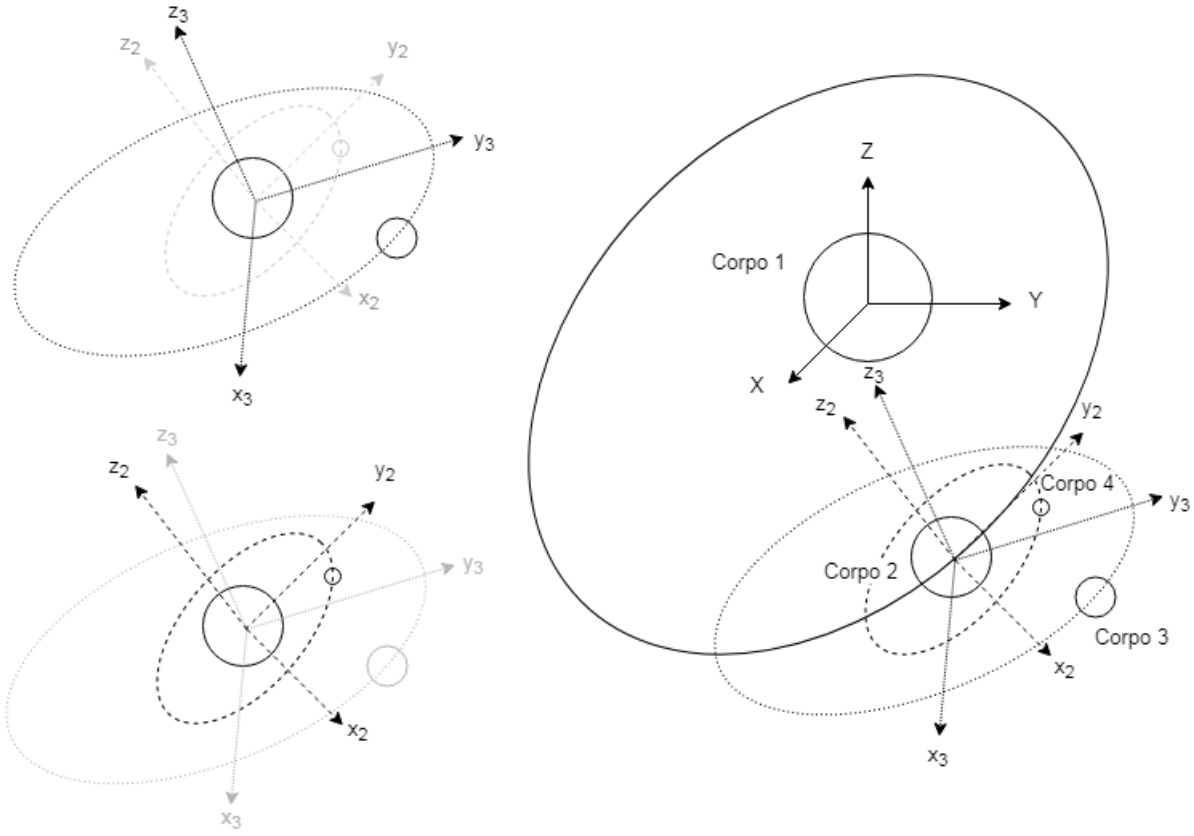

Figura 6 – Modelo esquemático do problema de 4 corpos trabalhado neste estudo.

A implementação do modelo do P4C foi feita segundo as considerações: Sol como corpo 1, Terra como corpo 2, Lua como corpo 3 e detrito como corpo 4. Foi estabelecido o referencial inercial $OXYZ$, centrado no Sol. Além disso, apesar de ter sido feito para o sistema Sol-Terra-Lua-Detrito, POC pode ser utilizado para outros problemas de quatro corpos. Para isso, deve, apenas, ser respeitado que o corpo 1 é o corpo central; o corpo 2 é orbitado pelos corpos 3 e 4; e que os corpos 2, 3 e 4 formam um sistema que orbita o corpo 1.

4.2 Compreender o Propagador Orbital Escrito em Fortran

Neste trabalho, propagador orbital é um *software* capaz de resolver as equações diferenciais de um problema orbital de N corpos ($N = 1, 2, ..., K$, sendo K um número qualquer), considerando as perturbações sofridas em órbita. Celestino (2005) usou o POF para estudar, considerando o P4C, o comportamento de um detrito sob diversas perturbações, como o arrasto de Poyinting-Robertson, perturbações luni-solares e de PRS.

POF, para realizar a propagação das órbitas, integra as equações de movimento com

um integrador numérico chamado de Gauss-Radau, de ordem 15 e que resolve equações diferenciais de segunda ordem, isto é, encontra a solução $x(t)$ para a ED $\ddot{x}(t) = f(t)$. No trabalho de Celestino (2005), esse integrador foi utilizado para realizar as simulações numéricas.

O estudo feito dos códigos do POF, etapa inicial deste projeto, permitiu a construção do diagrama da Figura 7, que apresenta, de maneira geral, o processo de propagação orbital desse simulador. Nesse processo, após a simulação ser iniciada, o programa abre os arquivos que receberão os resultados calculados. A seguir, as variáveis de entrada do programa são escritas com relação a $OXYZ$ e normalizadas, para que o programa trabalhe com números menores que números em escala orbital, consumindo menos memória. Depois da normalização, o vetor de estado é determinado calculando-se a posição e a velocidade de cada um dos corpos do P4C. O vetor de estado alimenta o integrador numérico Gauss-Radau, já configurado para simular a evolução das órbitas, com um tempo de final integração t_f definido.

Dentro do integrador são calculadas as perturbações orbitais e o movimento é integrado no tempo. A partir dos resultados normalizados, calculados com relação a $OXYZ$, é calculado o movimento do detrito com relação à Terra. Esses valores são desnormalizados e os elementos orbitais são calculados. Os resultados são, então, impressos no arquivo que havia sido aberto. Esse processo é repetido até que t_f seja atingido. Quando t_f é atingido, o processo de integração é finalizado. Neste momento, o arquivo de resultados, preenchido com as saídas da simulação, é fechado. A simulação do POF chega ao fim.

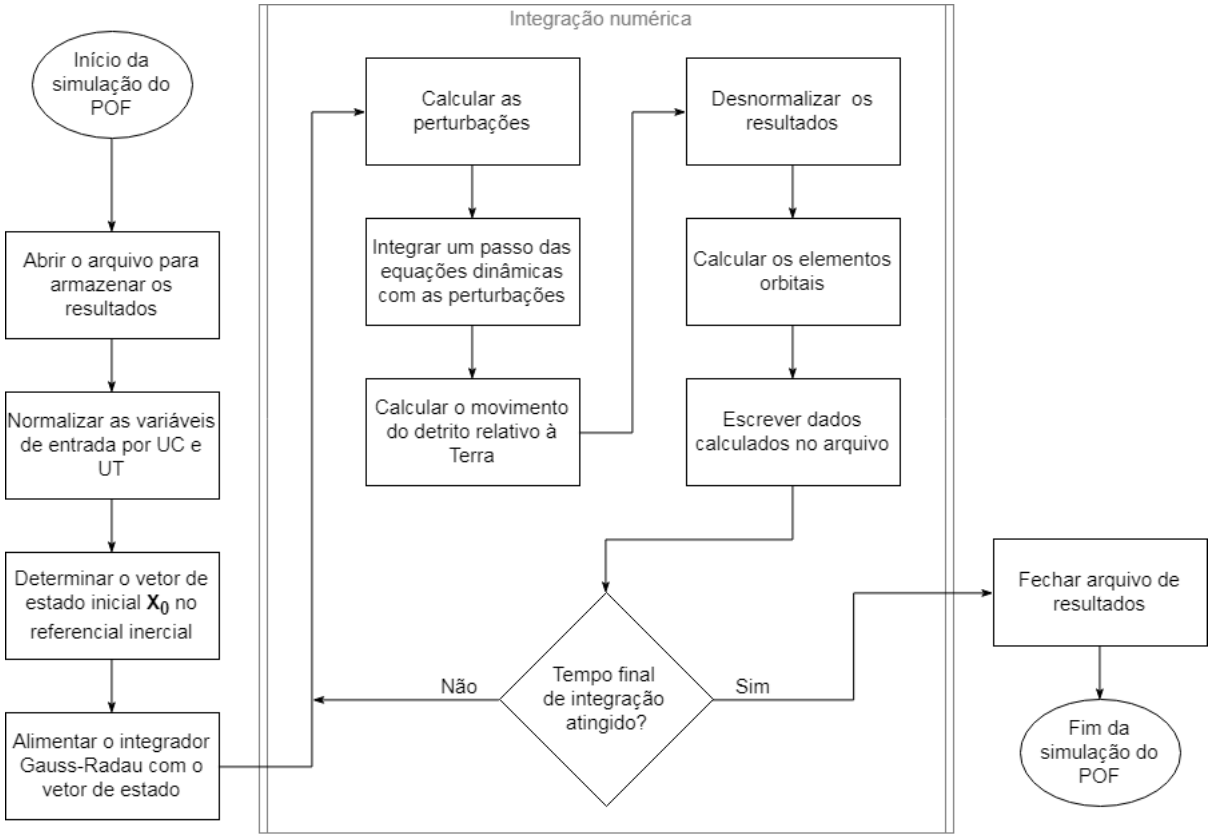

Figura 7 – Fluxograma simplificado do propagador orbital de Celestino (2005).

Finalizada a compreensão do processo de funcionamento do POF, deu-se início à transcrição de suas entradas para o POC.

4.3 Transcrição das Entradas do Propagador em Fortran para C++

As entradas do POF foram transcritas para C++ a partir do Fortran. Portanto, são quase idênticas, em termos de processo, apesar de em linguagens de programação diferentes, as maneiras como POF e POC recebem os dados de entrada e calculam o vetor de estado inicial. Essas diferenças são, de maneira geral:

- entre as linguagens de programação (o que mostrou-se um desafio por causa das diferenças de sintaxe);
- como os vetores de posição e velocidade são computados (no POC é utilizada uma biblioteca para cálculo matricial chamada Eigen, exclusiva do C++, como auxílio na determinação dos vetores de posição e velocidade do vetor de estado inicial);
- na ordem em que foram programadas as entradas da simulação;
- no tipo do vetor de estado.

Por outro lado, o que é igual de um código para o outro é:

- a implementação dos elementos orbitais de entrada;
- as equações de posição calculadas, a partir dos elementos orbitais, no referencial inercial;
- as equações de velocidade calculadas, a partir dos elementos orbitais, no referencial inercial;
- a construção do vetor de estado.

4.3.1 Fortran e C++

O POF foi escrito na versão 95 do Fortran (f95), uma pequena revisão da versão na qual foi escrito o integrador Gauss-Radau, a 90 (f90). Desde o surgimento da linguagem até as versões 90/95 passaram ao menos 30 anos. Hoje ainda existem aplicações e, passados mais de 20 anos do ano 1991, lançamento de f90, o Fortran já possui mais de 50 anos, um dos motivos de a linguagem ser tão eficiente (GAELZER, 2010).

Fortran significa *FORmula TRANslation* e foi desenvolvido por um time da IBM, liderado por John Backus, em meados dos anos 60. O objetivo era conseguir uma linguagem de alto nível, que tivesse uma sintaxe mais próxima da linguagem humana que da linguagem de máquina do computador, e que fosse quase tão eficiente quanto uma linguagem *Assembler* (linguagem *Assembler* era um tipo de programação com código-fonte numérico, traduzido para a CPU com programas chamados *Assemblers*, que eram, no mínimo, inconvenientes de se trabalhar) (GAELZER, 2010).

Com o desenvolvimento do Fortran foi possível afastar-se da programação de linguagem *Assembler*. O Fortran permitiu a escrita de equações quase de maneira simbólica, facilitando o trabalho de engenheiros e cientistas, deixando que mais atenção fosse dada à resolução do problema em si que à programação de um *Assembler*. A perda em eficiência foi pouca, com relação aos *Assemblers*, tendo em vista que muito cuidado foi dispendido na produção do compilador do Fortran (GAELZER, 2010).

Após 30 anos desde sua criação, haviam outras linguagens de programação em voga, como o C e o C++. Quando a versão Fortran 90 foi lançada, esta havia evoluído principalmente em termos de linguagem, para incluir recursos que o Fortran ainda não possuía mas de que outras linguagens já dispunham. Essas modificações foram tais como a capacidade de manipular matrizes com uma notação mais simples e poderosa, simplificando problemas matemáticos; possibilidade de usar ponteiros, de alocar memória dinamicamente, possibilidade de escrever programas recursivos, entre outras (GAELZER, 2010).

Ainda assim, apesar de todas as vantagens do Fortran 90/95, POC foi construído com a linguagem de programação C++. Os objetivos de utilizar o C++ foram diversificar a matriz de tecnologias disponíveis para estudar órbitas e preparar um *software*-base para o desenvolvimento de um ambiente de simulação orbital construído sob o paradigma de programação orientada a objetos (*POO*), que pode ser feito em um trabalho futuro. A diversificação da matriz de tecnologias permite a ampliação do número de ferramentas disponíveis para estudo, o que implica em mais possibilidades de abordar problemas. A ideia de construir um ambiente de simulação com *POO* aprimora a modelagem de objetos reais no computador, o que permitiria maior robustez em termos estruturais à simulação, além da redução da quantidade de código-fonte necessário para simular determinada situação. Entretanto, um ambiente de simulação desses no Fortran 90/95 não é factível, tendo em vista sua precariedade para programar com orientação a objetos (MÖSLI, 1995; GAELZER, 2010); e isso é algo que o C++ foi feito para fazer (ALBATROSS, 2016; ALBATROSS, 2017).

Além de ser possível programar com orientação a objetos, o C++, que é um incremento do C - seu nome significa exatamente isso: C-mais-mais (ALBATROSS, 2016) -, possui outras vantagens a serem levadas em consideração. Segundo Albatross (2017), o C++ é uma linguagem compilada, sendo uma das das mais rápidas do mundo, uma vez que o código-fonte for escrito de maneira otimizada. Além disso, é uma linguagem *strongly-typed* e *unsafe*, o que quer dizer que o C++ dá muita liberdade ao programador em termos de controle de memória e processamento; ao mesmo tempo, espera que ele ou ela saiba o que está fazendo. Dentre outras vantagens, o C++ oferece uma comunidade considerável de programadores em nosso planeta e grande variedade de bibliotecas; numa busca no SourceForge, há mais de 3000 bibliotecas disponíveis.

Foi utilizado na construção do POC um *Integrated Development Environment* (IDE) ou Ambiente de Desenvolvimento Integrado. Uma IDE é um tipo de programa de computador dentro do qual é possível desenvolver *softwares*. O POC foi construído dentro de uma IDE chamada CodeBlocks, que é *open-source*

4.3.2 O Novo Propagador Orbital

Estabelecido que o novo propagador orbital seria construído em C++, foi necessário repensar o processo de simulação. Essencialmente, porque a relação que o POF tem com como é calculado e atualizado o vetor de estado e as perturbações, na integração, é diferente de como isso deveria ser feito no POC, especialmente por causa da linguagem de programação e do integrador numérico diferentes. O diagrama apresentado na Figura 8 descreve, de maneira global, o processo de funcionamento do POC.

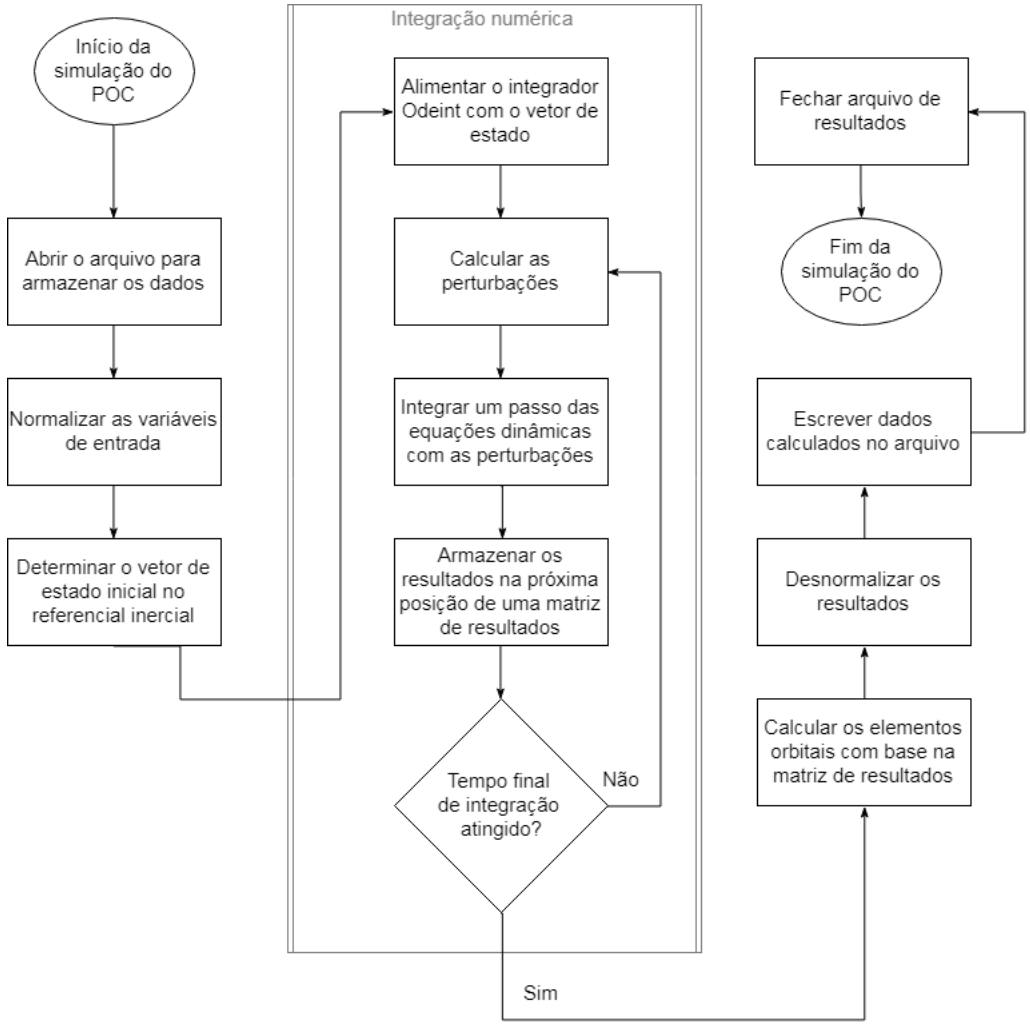

Figura 8 – Fluxograma simplificado do propagador orbital desenvolvido neste trabalho.

Note que os processos apresentados nas Figuras 7 e 8 são diferentes a partir do momento em que o integrador numérico é alimentado com o vetor de estado. Enquanto POF desnormaliza os resultados, calcula o movimento relativo do detrito para a Terra, calcula os elementos orbitais e armazena os resultados no arquivo de resultados durante a integração, POC trabalha de outra maneira.

O propagador orbital construído em C++, após alimentar o integrador com o vetor de estado, calcula as perturbações e integra as equações dinâmicas perturbadas; entretanto, esses resultados são salvos numa matriz de resultados declarada dentro do C++. É uma via de duas mãos. Inserir no processo uma matriz de resultados ocupa memória do computador e torna a simulação mais lenta. Entretanto, isso permite maior controle dos resultados, pois estes ainda não foram salvos no arquivo de resultados e podem ser manipulados com maior facilidade, o que foi preferido para cálculos pós-integração. A partir da matriz de resultados, são calculados os elementos orbitais, o movimento relativo do detrito para a Terra e a desnormalização. Apenas então os resultados são armazenados.

4.3.3 Biblioteca de Álgebra Linear Eigen

O C++ não possui nativamente métodos para cálculo matricial. Isso é uma desvantagem, pois pode tornar dispendiosos diversos cálculos e estender o trabalho do programador. Entretanto, existem bibliotecas desenvolvidas para vencer este problema e, neste trabalho, foi adotada a biblioteca Eigen (JACOB; GUENNEBAUD, 2018).

Segundo a citação no topo da página principal da wiki (JACOB; GUENNEBAUD, 2018), Eigen é uma biblioteca *template* do C++ para álgebra linear: matrizes, vetores, solucionadores numéricos e algoritmos relacionados. Ela é versátil, suporta vetores e matrizes de tamanhos fixos ou variáveis, uni, bi ou multidimensionais. Além disso, são aceitos todos os tipos numéricos padrão (inteiros, reais, imaginários etc.) e alguns customizáveis. O Eigen também suporta várias operações matriciais e vetoriais. É fácil de usar e elegante. Sobretudo, é rápido e confiável, além de detalhadamente documentado.

Esta é uma biblioteca que usa *templates*, dependendo apenas dos *header files*, que são os cabeçalhos do C++. Isto quer dizer que não há necessidade de instalar o Eigen. Basta fazer o *download*, da biblioteca, adicionar o caminho da pasta do Eigen nos diretórios de busca do CodeBlocks e adicionar os cabeçalhos do Eigen no código-fonte, como mostrado abaixo.

```
1 #include <Eigen/Dense>
2 using namespace Eigen;
```

O Eigen torna mais simples, dentre outras coisas, declarar e manipular vetores e matrizes. Para declarar, por exemplo, um vetor **v** de 3 posições e atribuir a ele valores 1, 2 e 3, basta escrever:

```
1 Vector3d v;
2 v << 1, 2, 3;
```

Outra maneira de criar um vetor com dimensão [3] e atribuir valores a ele pode ser:

```
1 VectorXd v(3);
2 v << 1, 2, 3;
```

De maneira análoga, uma matriz **M** com dimensões [2 x 2] pode ser criada com:

```
1 MatrixXd M(2, 2);
2 M << 11, 12, 21, 22;
```

Para fins de comparação, observe abaixo a multiplicação de um vetor **v** por uma matriz **M** construída sem o Eigen. A parte programada do código que de fato é pertinente à multiplicação pode ser contada da linha 5 para a linha 15, portanto, 11 linhas de código foram consumidas.

```
1  #include <iostream>
2  #include <stdio.h>
3  int main()
4  {
5      double v[2], M[2][2];
6      v[0] = 1;
7      v[1] = 2;
8      M[0][0] = 1;
9      M[0][1] = 1;
10     M[1][0] = 1;
11     M[1][1] = 1;
12     for(int i = 0;  i < 2; i++)
13     {
14         std::cout << M[i][0]*v[0] + M[i][1]*v[1] << std::endl;
15     }
16     return 0;
17 }
```

Por outro lado, observe abaixo o uso do Eigen para realizar a mesma multiplicação. A parte de programação que executa de fato a multiplicação vai da linha 6 à linha 10. Portanto, 5 linhas. Isso dá um ganho de 6 linhas para o Eigen neste caso.

```
1  #include <iostream>
2  #include <Eigen/Dense>
3  using namespace Eigen;
4  int main()
5  {
6    VectorXd v(2);
7    MatrixXd M(2, 2);
8    v << 1, 2;
9    M << 1, 1, 1, 1;
10   std::cout << M*v << std::endl;
11   return 0;
12 }
```

O uso principal do Eigen foi na construção do vetor de estado, para agilizar o tempo gasto na escrita do código dos vetores de posição e velocidade iniciais calculados a partir dos elementos orbitais. O Eigen permitiu que menos tempo fosse gasto produzindo a sintaxe e mais tempo fosse dedicado à elaboração das equações que produziram o estado inicial, que foi usado para alimentar o integrador.

4.3.4 Construção do Vetor Inicial de Estado

A construção do vetor inicial de estado $\mathbf{X}$ foi tema central dessa parte do trabalho. O vetor de estado descreve a situação do sistema em um determinado instante de tempo. Para o P4C, são quatro corpos considerados, cada um com três componentes de posição ($x_{i,j}$, $i = 1, 2, 3, 4$, $j = X, Y, Z$) e três de velocidade ($v_{i,j}$, $i = 1, 2, 3, 4$, $j = X, Y, Z$), calculadas com relação a $OXYZ$, totalizando 24 componentes. A estrutura de $\mathbf{X}$ segue o esquema da Equação 4.1.

$$\begin{aligned}\mathbf{X} &= [\mathbf{x}\ \mathbf{v}]^T \\ &= [x_{1,X}\ x_{1,Y}\ x_{1,Z}\ x_{2,X}\ x_{2,Y}\ x_{2,Z}\ x_{3,X}\ x_{3,Y}\ x_{3,Z}\ x_{4,X}\ x_{4,Y}\ x_{4,Z} \\ &\quad v_{1,X}\ v_{1,Y}\ v_{1,Z}\ v_{2,X}\ v_{2,Y}\ v_{2,Z}\ v_{3,X}\ v_{3,Y}\ v_{3,Z}\ v_{4,X}\ v_{4,Y}\ v_{4,Z}]^T\end{aligned} \tag{4.1}$$

O método de construção do vetor de estado inicial mostrado na Equação 4.1 funciona de acordo com a seguinte ideia:

1. definir a posição e a velocidade do corpo 1 com relação a $OXYZ$;
2. calcular a posição do corpo 2 relativa ao corpo 1;
3. calcular a posição do corpo 2 no referencial $OXYZ$;
4. calcular a posição do corpo 3 com relação ao corpo 2;
5. calcular a posição corpo 3 no referencial $OXYZ$;
6. calcular a posição do corpo 4 com relação ao corpo 2;
7. calcular a posição corpo 4 no referencial $OXYZ$;
8. repetir o procedimento para a velocidade.

Este processo de determinação de posição e velocidade pode ser melhor compreendido se for analisada a Figura 9. A figura mostra a posição de um corpo i com relação ao corpo 2 e ao corpo 1, além da decomposição trigonométrica da posição do corpo i com relação ao corpo 2 em termos dos elementos orbitais do corpo i.

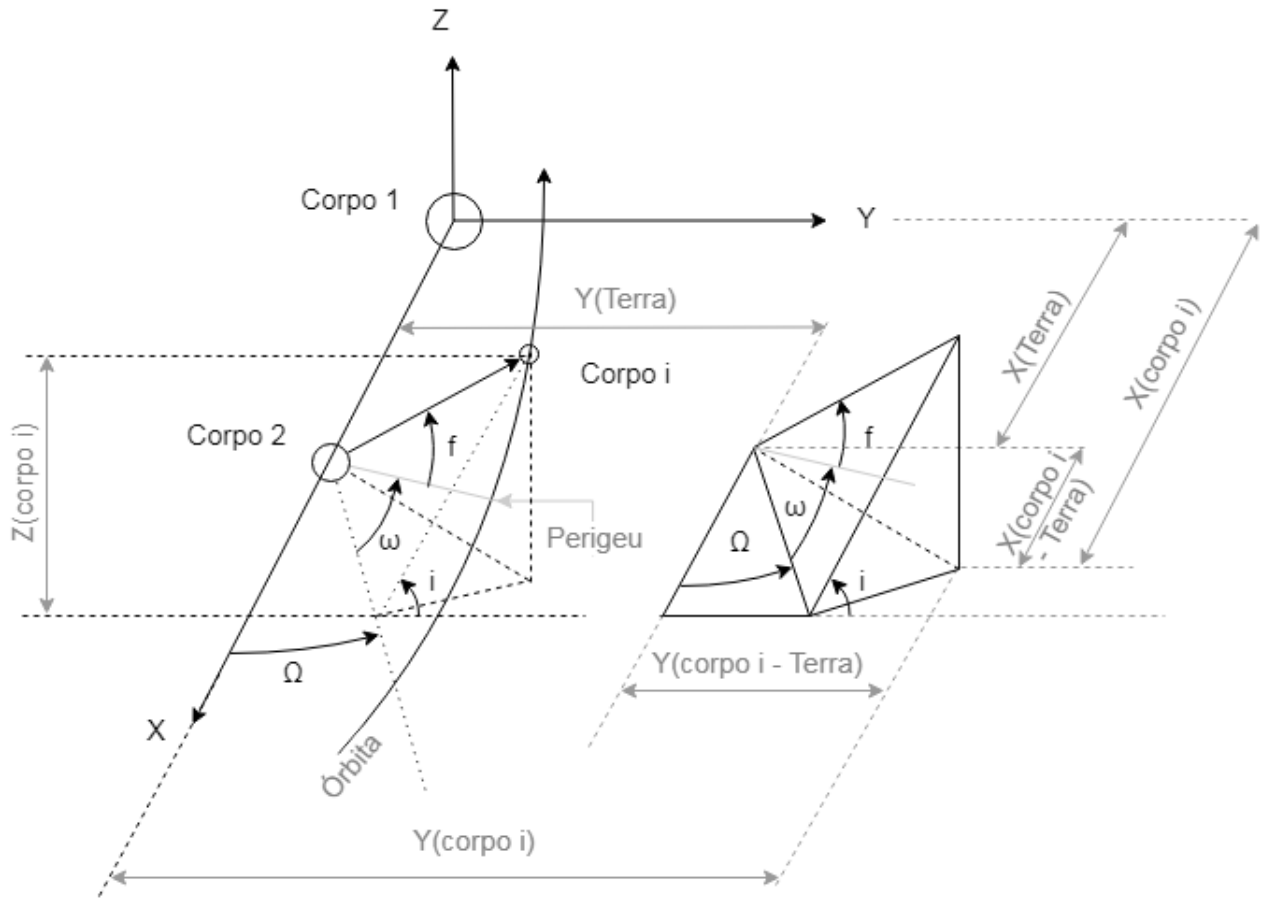

Figura 9 – Decomposição trigonométrica da posição de um corpo i, orbitando o corpo 2, formando um sistema que orbita o corpo 1.

Para entender como pode ser calculada a posição e a velocidade de um corpo i, que orbita um corpo 2, e que juntos orbitam um corpo 1, admita o corpo 1 como sendo o Sol e o corpo 2 a Terra. O corpo i pode ser a Lua ou um detrito ou um satélite qualquer. Segundo a Figura 6, o interesse, para descrever um corpo i com relação a $OXYZ$, é calcular as variáveis $[X(corpo\,i)\ Y(corpo\,i)\ Z(corpo\,i)]$, que são, respectivamente, as posições X, Y e Z do corpo i com relação ao referencial inercial. As derivadas dessas posições retornam, respectivamente, as velocidades $[V_X(corpo\,i)\ V_Y(corpo\,i)\ V_Z(corpo\,i)]$. (Observação 1: esta notação de posição e velocidade serve apenas para ilustrar o que é mostrado na Figura 9 com uma linguagem mais próxima da natural. Observação 2: os índices ji são usados para descrever o movimento do corpo i relativo ao j.)

Note que, de acordo com a Figura 9,

$$X(corpo\,i) = X(Terra) + X(corpo\,i - Terra),$$

o que é equivalente a

$$X(corpo\,i\,em\,OXYZ) = X(Terra\,em\,OXYZ) + X(corpo\,i\,relativo\,à\,Terra).$$

O mesmo pode ser dito das posições $Y(corpo\,i)$ e $Z(corpo\,i)$; e das velocidades $V_X(corpo\,i)$, $V_Y(corpo\,i)$ e $V_Z(corpo\,i)$; assim como de suas posições e velocidades relativas, denotadas pelos índices ji.

Reduzindo a notação para:

- $r_{X,i} = X(corpo\,i)$;
- $r_{Y,i} = Y(corpo\,i)$;
- $r_{Z,i} = Z(corpo\,i)$;
- $r_{X,ji} = X(corpo\,i - corpo\,j)$;
- $r_{Y,ji} = Y(corpo\,i - corpo\,j)$;
- $r_{Z,ji} = Z(corpo\,i - corpo\,j)$;
- $v_{X,i} = V_X(corpo\,i)$;
- $v_{Y,i} = V_Y(corpo\,i)$;
- $v_{Z,i} = V_Z(corpo\,i)$;
- $v_{X,ji} = V_X(corpo\,i - corpo\,j)$;
- $v_{Y,ji} = V_Y(corpo\,i - corpo\,j)$;
- $v_{Z,ji} = V_Z(corpo\,i - corpo\,j)$;

tem-se que as Equações 4.2 modelam as posições de um corpo i com relação a um corpo j. Essas equações consideram os elementos orbitais com relação a $OXYZ$ e a magnitude do raio orbital r_{ji} entre o corpo j e o corpo i, tendo sido calculadas pela decomposição trigonométrica da posição do corpo i em relação ao corpo 2, mostrada na Figura 9.

$$\begin{aligned} r_{X,ji} &= r_{ji}[\cos\Omega_i \cos(\omega_i + f_i) - \operatorname{sen}\Omega_i \operatorname{sen}(\omega_i + f_i) \operatorname{sen} i_i] \\ r_{Y,ji} &= r_{ji}[\operatorname{sen}\Omega_i \cos(\omega_i + f_i) + \cos\Omega_i \operatorname{sen}(\omega_i + f_i) \operatorname{sen} i_i] \\ r_{Z,ji} &= r_{ji}[\operatorname{sen}(\omega_i + f_i) \operatorname{sen} i_i] \end{aligned} \tag{4.2}$$

As Equações 4.3 foram calculadas a partir das derivadas das Equações 4.2 por Celestino (2005). Nelas, são considerados os elementos orbitais do corpo i e parâmetro gravitacional μ_j do corpo j.

$$
\begin{aligned}
v_{X,ji} &= -\sqrt{\frac{\mu_j}{a_i}}\left\{\left[\frac{1}{\sqrt{1-e_i^2}}\right]\left[(\cos(i_i)(e_i\cos(\omega_i)+\cos(\omega_i+f_i))\,\mathrm{sen}(\Omega_i))+\right.\right.\\
&\quad \left.\left.(\cos(\Omega_i)(e_i\,\mathrm{sen}(\omega_i)+\,\mathrm{sen}(f_i+\omega_i)))\right]\right\}\\
v_{Y,ji} &= \sqrt{\frac{\mu_j}{a_i}}\left\{\left[\frac{1}{\sqrt{1-e_i^2}}\right]\left[(\cos(i_i)(e_i\cos(\omega_i)+\cos(f_i+\omega_i))\cos(\Omega_i)-\right.\right.\\
&\quad \left.\left.\mathrm{sen}(\Omega_i)(e_i\,\mathrm{sen}(\omega_i)+\,\mathrm{sen}(f_i+\omega_i)))\right]\right\}\\
v_{Z,ji} &= \sqrt{\frac{\mu_j}{a_i}}\left\{\left[\frac{1}{\sqrt{1-e_i^2}}\right]\left[e_i\cos(\omega_i)+\cos(f_i+\omega_i)\right]\mathrm{sen}(i_i))\right\}
\end{aligned}
\tag{4.3}
$$

A posição e velocidade para o corpo 1 tem apenas componentes de valor zero e, uma vez que essas equações foram calculadas para os corpos 2, 3 e 4, podem ser obtidos os movimentos relativos entre esses corpos para o P4C. É possível, então, montar o vetor de estado de acordo com o procedimento já apresentado, detalhado a seguir.

As Equações 4.4 e 4.5 definem os vetores posição posição e velocidade inicial do corpo 1 com relação ao referencial inercial. Neste trabalho, como $OXYZ$ estava centrado no corpo 1, esses valores ficaram iguais a zero.

$$\mathbf{r_1} = [r_{X,1}\; r_{Y,1}\; r_{Z,1}]^T \tag{4.4}$$

$$\mathbf{v_1} = [v_{X,1}\; v_{Y,1}\; v_{Z,1}]^T \tag{4.5}$$

Em seguida, foram calculados os vetores posição e velocidade relativos do corpo 2 com relação ao 1, utilizando as Equações 4.6 e 4.7. O movimento relativo foi calculado com as já apresentadas Equações 4.2 e 4.3.

$$\mathbf{r_{12}} = [r_{X,12}\; r_{Y,12}\; r_{Z,12}]^T \tag{4.6}$$

$$\mathbf{v_{12}} = [v_{X,12}\; v_{Y,12}\; v_{Z,12}]^T \tag{4.7}$$

As Equações 4.8 e 4.9 foram então aplicadas para determinar os vetores posição e velocidade do corpo 2 no referencial $OXYZ$.

$$\mathbf{r_2} = \mathbf{r_1} + \mathbf{r_{12}} \tag{4.8}$$

$$\mathbf{v_2} = \mathbf{v_1} + \mathbf{v_{12}} \tag{4.9}$$

O próximo passo foi calcular os vetores posição e velocidade do corpo 3 com relação ao corpo 2 por meio das Equações 4.10 e 4.11.

$$\mathbf{r_{23}} = [r_{X,23}\ r_{Y,23}\ r_{Z,23}]^T \tag{4.10}$$

$$\mathbf{v_{23}} = [v_{X,23}\ v_{Y,23}\ v_{Z,23}]^T \tag{4.11}$$

Os vetores de posição e velocidade do corpo 3, com relação a $OXYZ$, foram obtidos com as Equações 4.12 e 4.13.

$$\mathbf{r_3} = \mathbf{r_2} + \mathbf{r_{23}} \tag{4.12}$$

$$\mathbf{v_3} = \mathbf{v_2} + \mathbf{v_{23}} \tag{4.13}$$

O que foi feito ao corpo 3 foi repetido para calcular os vetores de posição e velocidade do corpo 4 em relação ao corpo 2, utilizando as Equações 4.14 e 4.15.

$$\mathbf{r_{24}} = [r_{X,24}\ r_{Y,24}\ r_{Z,24}]^T \tag{4.14}$$

$$\mathbf{v_{24}} = [v_{X,24}\ v_{Y,24}\ v_{Z,24}]^T \tag{4.15}$$

Por fim, os vetores posição e velocidade do corpo 4 com relação a $OXYZ$ são dados pelas Equações 4.16 e 4.17.

$$\mathbf{r_4} = \mathbf{r_2} + \mathbf{r_{24}} \tag{4.16}$$

$$\mathbf{v_4} = \mathbf{v_2} + \mathbf{v_{24}} \tag{4.17}$$

Cada uma das equações apresentadas acima é vetorial. Elas têm, portanto, três componentes cada. Se as componentes de posição forem dispostas sequencialmente num vetor para os corpos 1, 2, 3 e 4 e, depois, isso for feito para as componentes da velocidade, será obtido o vetor de estado inicial $\mathbf{X}$, apresentado na Equação 4.1.

4.3.5 Condições Iniciais: Sol, Terra, Lua e de Teste para o Detrito

Segundo Curtis (2005), os dados da Tabela 1 valem para o Sol, Terra e Lua. Esta tabela apresenta as massas desses corpos, seus elementos orbitais com relação à eclíptica e seus períodos orbitais. Esses dados são entrada para o POC, de forma que os vetores de posição e velocidade são calculados a partir deles.

Tabela 1 – Informações sobre o Sol, Terra e Lua.

Corpo	**Nome**	$m\ (kg)$	$a\ (km)$	e	$i\ (^o)$	$\Omega\ (^o)$	$\omega\ (^o)$	$T\ (dia^{-1})$
1	Sol	1,98911E+30	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
2	Terra	5,97420E+24	149597870,6910	0,0000	0,0000	0,0000	0,0000	365,2500
3	Lua	7,34830E+22	384400,0000	0,0549	5,1454	0,0000	0,0000	27,3217

A configuração inicial de elementos orbitais para o detrito é apresentada na Tabela 2 e foi usada como padrão na etapa de desenvolvimento numérico, do início à validação. Nesta tabela também é apresentada a razão área-massa ϕ considerada padrão para o detrito do POC durante o desenvolvimento numérico.

Tabela 2 – Elementos orbitais iniciais de teste para o detrito.

Corpo	$\phi\ (m^2/kg)$	$a\ (km)$	e	$i\ (^o)$	$\Omega\ (^o)$	$\omega\ (^o)$	$f\ (^o)$
Detrito	50	8500,7	0,0104	47	328	162	0

Cabe a ressalva de que apesar de a simulação ser executada para uma órbita LEO e os resultados dinâmicos serem coerentes, deve-se levar em conta que em órbita baixa há outras perturbações mais relevantes que as luni-solares e a PRS afetando o movimento do detrito. Essas perturbações mais relevantes são, por exemplo, o arrasto atmosférico e o achatamento terrestre (CELESTINO, 2005; RIEBEEK, 2009).

Isso quer dizer que, apesar de matematicamente os resultados estarem corretos, fisicamente não seriam observados por causa das perturbações mais relevantes que não foram consideradas na simulação. Entretanto, os resultados numéricos de POF para as condições da Tabela 4 podem ser usados para validar o POC.

Após calcular os vetores de posição e velocidade iniciais com relação a $OXYZ$, esses resultados foram organizados na Tabela 3. Os dados de teste do detrito também foram tabulados, pois serão utilizados adiante na validação dos resultados entre POF e POC.

Tabela 3 – Dados de entrada usados na validação entre os propagadores em Fortran e C++. Esses dados formam o vetor de estado inicial.

Estado	**Sol**	**Terra**	**Lua**	**Detrito**
$r_{X,i}$ (km)	0	149597870,6	149961167	149592025,4
$r_{Y,i}$ (km)	0	0	0	5743,3204
$r_{Z,i}$ (km)	0	0	0	1901,2424
$v_{X,i}$ (km/s)	0	0	0	-4,191695139
$v_{Y,i}$ (km/s)	0	29,78613716	30,85767886	27,11361396
$v_{Z,i}$ (km/s)	0	0	0,096487069	-4,812941597

Os dados de posição e velocidade iniciais do detrito também foram calculados relativos à Terra e são apresentados na Tabela 4.

Tabela 4 – Estado inicial de teste do detrito.

Estado do Detrito com Relação à Terra	**Detrito**
$r_{X,4}$ (km)	-5845,4
$r_{Y,4}$ (km)	5743,1
$r_{Z,4}$ (km)	1901,2
$v_{X,4}$ (km/s)	-4,1917
$v_{Y,4}$ (km/s)	-2,6731
$v_{Z,4}$ (km/s)	-4,8129

4.3.6 Normalização dos Valores de Entrada e o Vetor de Estado Inicial

Os dados gravados no vetor de estado inicial não entram com suas magnitudes reais no integrador. Antes disso eles são normalizados. Isso foi feito para que o computador pudesse trabalhar com números menores, de forma que números grandes (da ordem de centenas de milhares e milhões) afetassem menos os resultados e o desempenho do propagador (ASAITHAMBI, 2017).

Para efetuar as normalizações, foram definidos os seguintes padrões: UC para unidade de comprimento, UT para unidade de tempo e UM para unidade de massa. Os

valores de UC, UT e UM são mostrados na Tabela 5 com seus respectivos valores.

Tabela 5 – Valores dos fatores de normalização.

Fator de normalização	**Variável**	**Valor normalizado**	**Valor desnormalizado**	**Descrição**
Unidade de comprimento	UC	1	384400 km	Distância da Terra à Lua
Unidade de tempo	UT	1	86400 s	Segundos em um dia
Unidade de massa	UM	1	$6{,}04768\times10^{24}$ kg	Soma das massas da Terra e da Lua

Assim, a Constante e o Parâmetro Gravitacional devem ser dados pelas unidades mostradas nas Equações 4.18 e 4.19.

$$[G] = \frac{UC^3}{kg\ dia^2} \tag{4.18}$$

$$[\mu] = \frac{UC^3}{dia^2} \tag{4.19}$$

A Equação 4.20 mostra a unidade de medida da parte do vetor de estado inicial que representa os valores de posição dos 4 corpos.

$$[x_i] = UC \tag{4.20}$$

A Equação 4.21 mostra a unidade de medida da parte do vetor de estado inicial que representa os valores de velocidade dos 4 corpos.

$$[x_i] = \frac{UC}{dia} \tag{4.21}$$

Com base nos fatores de normalização apresentados, os valores do vetor inicial de estado, normalizados, que alimentaram o integrador numérico, são apresentados na Tabela 6. Esses dados foram calculados com relação ao referencial $OXYZ$. Além disso, as componentes de posição e velocidade inicial do detrito são referentes às condições de teste já apresentada.

Tabela 6 – Dados de entrada normalizados usados na validação entre os propagadores em Fortran e C++, com relação a $OXYZ$.

Estado (normalizado)	**Sol**	**Terra**	**Lua**	**Detrito**
$r_{X,i}$ (UC)	0	389,1724	390,1175	389,157194
$r_{Y,i}$ (UC)	0	0	0	0,014941
$r_{Z,i}$ (UC)	0	0	0	0,004946
$v_{X,i}$ (UC/dia)	0	0	0	-0,94215
$v_{Y,i}$ (UC/dia)	0	6,694907	6,935753	6,094215
$v_{Z,i}$ (UC/dia)	0	0	0,021687	-1,081785

Para analisar o efeito da normalização sobre o desempenho da simulação, o código foi executado 5 vezes para os dados normalizados e 5 vezes para os dados não normalizados. Os intervalos de integração foram de 25, 50, 75, 100 e 125 dias. O tempo de simulação, em segundos, foi medido para cada um dos intervalos de integração. Os resultados desses testes foram dispostos na Tabela 7. O ganho de tempo da simulação normalizada para a não normalizada foi de (0,137 ± 0,011) s/dia. Isto equivale a (13,7 ± 1,1) s a menos de processamento para cada 100 dias de simulação, demonstrando que a normalização melhora o desempenho do programa.

Tabela 7 – Comparação das simulações com dados de entrada normalizados e não normalizados.

Dias	**Normalizado (s)**	**Não Normalizado (s)**	**Ganho (s)**	**Ganho (s/dia)**
25	19,639	23,221	3,582	0,143
50	38,400	46,247	7,847	0,157
75	60,949	70,886	9,937	0,132
100	80,477	91,825	11,348	0,113
125	99,638	116,904	17,266	0,138

4.3.7 Bibliotecas Boost e Odeint

Com a mudança de linguagem de programação de Fortran para C++, não foi possível utilizar o mesmo código-fonte de integrador numérico usado por Celestino (2005). Esse problema foi resolvido usando um integrador numérico da biblioteca Odeint, que é parte das bibliotecas Boost do C++. Isso levou à necessidade de alterar o processo de construção da função de equações dinâmicas e das perturbações, tendo em vista que os integradores numéricos do Odeint integram equações diferenciais de primeira ordem

(AHNERT; MULANSKY, 2018c) e o Gauss-Radau, como foi usado no POF, integra equações diferenciais de segunda ordem.

De acordo com Dawes e Abrahams (2018c), o Boost é um conjunto de bibliotecas revisadas por pares, cuja intenção é ser amplamente utilizado em diversas aplicações. Segundo Dawes e Abrahams (2018a), essas bibliotecas deveriam ser utilizadas para aumentar a produtividade, tendo em vista a qualidade das mesmas, sendo encorajado por seus criadores usos comerciais ou não.

O Boost possui várias bibliotecas, abrangendo diversos tipos de aplicações. A maioria delas pode ser usada em um código apenas escrevendo o cabeçalho referente à biblioteca, isto é, chamando o *header file* desejado no início do programa (DAWES; ABRAHAMS, 2018b).

A biblioteca Odeint foi construída de forma a fornecer integradores numéricos para resolver problemas de valor inicial de equações diferenciais ordinárias (EDOs). Matematicamente, esses problemas são formulados por: $\dot{x}(t) = f(t) : x(0) = x_0$, sendo x uma função qualquer do tempo $f(t)$, $\dot{x}(t)$ a derivada de x no tempo e x_0 o valor de x no tempo zero (AHNERT; MULANSKY, 2018c). Note que essa é uma derivada de primeira ordem e vale ser ressaltada, pois essa é uma das grandes diferenças entre os integradores numéricos do POC e do POF; em contraposição aos integradores do Odeint, o Gauss-Radau, do POF, resolve derivadas de segunda ordem.

Os integradores numéricos do Odeint são funções que realizam a integração de EDOs com base nos argumentos que as alimentam. De maneira geral, essas funções dividem alguns argumentos em comum e, dependendo do fim para o qual elas foram desenvolvidas, podem ter também seus argumentos próprios. De acordo com Ahnert e Mulansky (2018b), os argumentos dos integradores numéricos da biblioteca Odeint são:

- *stepper*: é um tipo de função que executa um passo da solução das EDOs a partir de um passo de integração dt. Existem quatro *steppers* disponíveis: *Basic Stepper*, *Error Stepper*, *Controlled Stepper* e *Dense Output Stepper*. Cada um deles funciona para determinado fim, como apresentado a seguir:
  - *Basic Stepper*: conceito básico de *stepper* que calcula um passo da solução $x(t)$ da EDO para obter $x(t + dt)$, com base em dt dado. Esse *stepper* pode ser do tipo Runge-Kutta, simplético ou implícito (AHNERT; MULANSKY, 2018e);
  - *Error Stepper*: similar ao *basic stepper*, mas estima o erro do resultado;
  - *Controlled Stepper*: este *stepper* executa um passo da solução $x(t)$ da EDO para calcular $x(t + dt)$, dado dt e, a depender da estimativa do erro do resultado, o passo de integração pode ser rejeitado e um passo menor pode ser sugerido (AHNERT; MULANSKY, 2018a);

  - *Dense Output Stepper*: este *stepper* executa um passo da solução $x(t)$ das EDOs para obter $x(t + dt)$. Neste caso, dt pode ser ajustado devido ao controle de erro nos resultados. Este *stepper* também pode interpolar os resultados entre t e $t + dt$.

- *system*: este conceito modela a implementação algorítmica do lado direito das equações que serão integradas, isto é, do integrando. *System* deve ser modelado, portanto, seguindo o procedimento de cálculo que deve ser executado pelas equações a serem integradas;
- $x0$: este é o vetor de estado inicial que alimenta o integrador;
- $t0$: este é o tempo inicial da integração numérica, executada em um intervalo $[t0\ t1]$;
- $t1$: este é o tempo final da integração numérica, executada em um intervalo $[t0\ t1]$;
- dt: este é o passo de integração;
- *observer*: função de observação, pode ser chamada para analisar o vetor de estado corrente da integração durante a evolução temporal das EDOs;
- *max_step_checker*: é uma função que gera um erro se um número excessivo de passos (500, por padrão) acontecer entre duas chamadas da função de observação;
- n_{passos}: este é o número total de passos de integração;
- *times_start*: é a primeira componente de um vetor de tempos de observação;
- *times_end*: é a última componente de um vetor de tempos de observação.

O Odeint disponibiliza cinco integradores numéricos, cada um possuindo uma estratégia diferente para chamar a função de observação durante a integração. Os integradores são mostrados na lista abaixo. Os argumentos mostrados acima alimentam os integradores (AHNERT; MULANSKY, 2018b).

- integrate_const(*stepper*, *system*, $x0$, $t0$, $t1$, dt, *observer*, *max_step_checker*)
- integrate_n_steps(*stepper*, *system*, $x0$, $t0$, dt, n_{passos}, *observer*, *observer*, *max_step_checker*)
- integrate_adaptive(*stepper*, *system*, $x0$, $t0$, $t1$, dt, *observer*)
- integrate_times(*stepper*, *system*, $x0$, *times_start*, *times_end*, dt, *observer*, *max_step_checker*)
- integrate(*system*, $x0$, $t0$, $t1$, dt, *observer*)

As funcionalidades desses integradores numéricos são as seguintes (note que cada um tem uma estratégia diferente para chamar a função de observação):

- integrate_const: chama a função de observação com intervalos equidistantes a cada dt;
- integrate_n_steps: similar à integrate_const, mas, ao invés do tempo final de integração, é passado ao integrador o número de passos total de integração;
- integrate_adaptive: chama a função de observação a cada passo e, dependendo do *stepper* escolhido, os dt serão ou não equidistantes;
- integrate_times: é passado um vetor de tempos de observação ou os tempos inicial e final desse vetor, de forma que a função de observação é chamada a cada tempo dado;
- integrate: é um integrador numérico simplificado que se comporta de maneira igual ao integrate_adaptative, com exceção de que, para o integrate, o *stepper* é padronizado como *dense output stepper* baseado no método runge_kutta_dopri5 (AHNERT; MULANSKY, 2018b).

Neste trabalho foi optado pelo uso do integrador integrate_const. Essa escolha foi feita para que fosse obtido maior controle sobre a evolução temporal da integração e, por meio da função de observação, obter as medidas do vetor de estado de maneira temporalmente equidistante. Isso foi desejável para que os resultados do POC pudessem ser validados com os resultados do POF, uma comparação que é mais simples para resultados medidos de forma temporalmente equidistante.

A Figura 10 mostra o esquema básico de funcionamento do integrador integrate_const. Primeiro, dá-se o início da integração. O vetor de estado inicial $\mathbf{X}$ alimenta a função *system*, que contém as equações dinâmicas. Então as posições relativas entre todos os corpos são calculadas. Depois, com base nisso, são calculadas as derivadas $d\mathbf{X}/dt$ de cada componente do estado considerando as perturbações e as equações dinâmicas. Esse conjunto é integrado pelo *stepper* previamente escolhido, que realiza a integração até que o tempo final seja atingido. Por fim, o integrate_const retorna quantos passos de integração foram efetuados.

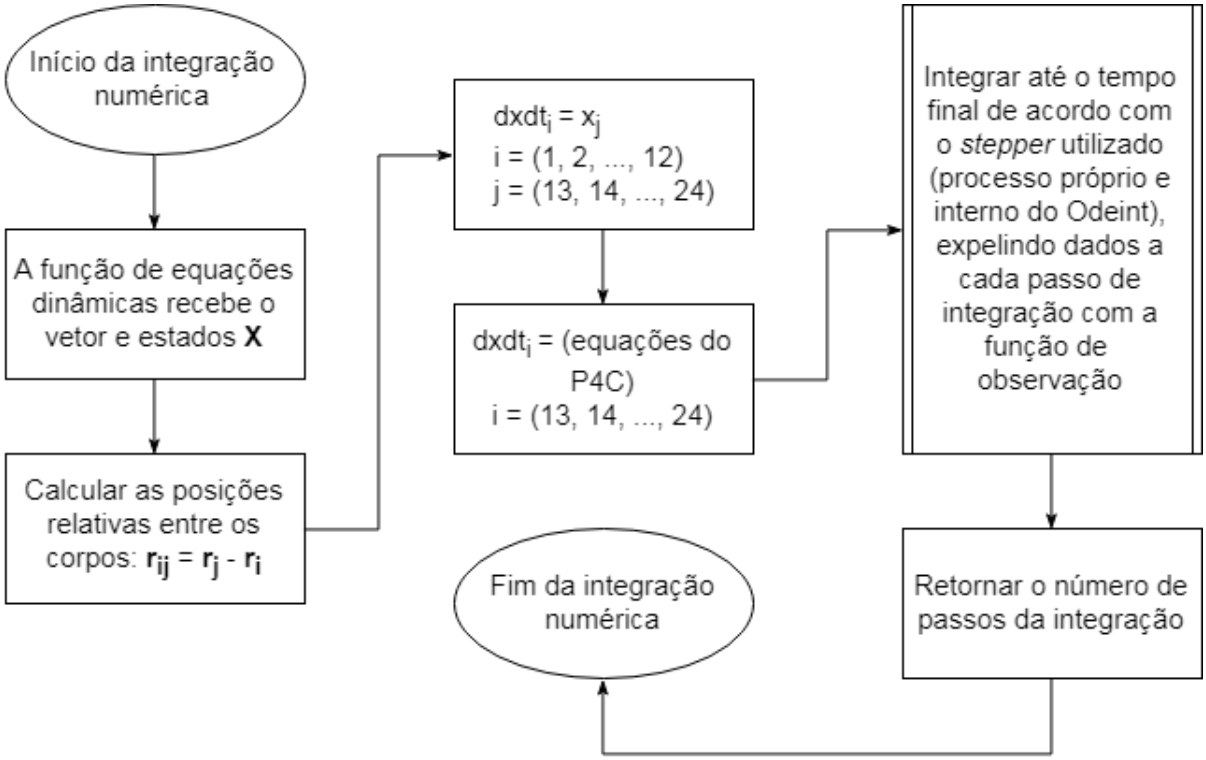

Figura 10 – Funcionamento básico e conceitual do integrador numérico integrate_const ba biblioteca Odeint.

O *stepper* escolhido para o POC foi do tipo *Error Stepper*. O método segundo o qual esse funciona é o Fehlberg 78 (AHNERT; MULANSKY, 2018c), ou método de Runge-Kutta-Fehlberg. Esse é um *stepper* de ordem 8, com estimativa de erro nos resultados. A versão Gauss-Radau usado por Celestino (2005) possui ordem 15.

A função a ser integrada, *system*, também tem um formato especial para ser chamada no código. Esse formato deve ser levado em consideração na construção do programa e tem uma sintaxe própria, que deve ser obedecida para o funcionamento correto (AHNERT; MULANSKY, 2018g; AHNERT; MULANSKY, 2018f). Ao declarar uma função dessas, o formato deve ser:

$$\text{sys}(x,\ dxdt,\ t)$$

em que *sys* é o nome da função *system*, x o vetor de estado inicial, $dxdt$ é o vetor de derivadas das componentes do estado inicial e t o tempo de integração (essas variáveis estão de acordo com a sintaxe do C++). De maneira geral, o que acontece é que *sys* recebe o vetor de estado, com o qual calcula o vetor $dxdt$. $dxdt$ é integrado pelo *stepper*, gerando um novo estado que sobrescreve o estado anterior. Nesse ínterim, após a integração, a função de observação coleta o estado calculado. Este processo se mantém até o fim da integração.

No Odeint, os *steppers* são implementados de maneira totalmente independente das operações matemáticas. Isso é feito dando ao usuário controle total sobre o tipo da variável de estado e das operações matemáticas para esse estado. O tipo a que a variável de estado é atribuída, quando declarada, é chamado de state_type. No POC, no cabeçalho do código é escrita a seguinte sentença para definir esse tipo:

```
1 typedef std::vector<double> state_type;
```

Isso define um tipo de vetor *double* que será usado para criar a variável de estado **X**.

A discussão feita até então aborda o que é necessário para utilizar os integradores numéricos da biblioteca Odeint. Sintetizando o que foi apresentado, a aplicação do integrate_const neste trabalho ficou como é mostrado a seguir.

```
1
2  // ... header files
3
4  typedef std::vector<double> state_type;
5
6  // ... entradas e metodos auxiliares
7
8  struct push_back_state_and_time {...}
9
10 void eq_din(const state_type &x, state_type &dxdt, const double
   times) {...}
11
12 // ... manipulacao de dados  de entrada
13
14 vector<state_type> y;
15 vector<double> t;
16 state_type x(24);
17
18 // ... calculo do vetor de estado inicial x,
19 // da matriz y de resultados e do vetor de tempos t
20
21 double t_f = 366; // neste caso, 366 dias de integracao como exemplo
22
23 runge_kutta_fehlberg78<state_type> stepper;
24
25 size_t steps = integrate_const(stepper, eq_din, x, 0.0, t_f, \
26 0.0001, push_back_state_and_time(y, t));
27
```

```
28 // ... pos-processamento de resultados
29
30 return 42;
```

No código apresentado acima, o primeiro comentário refere-se à chamada dos cabeçalhos necessários para programar o simulador. Depois, é criado o tipo state_type. O próximo comentário simboliza as entradas de dados e métodos auxiliares à simulação. Depois é declarada uma *struct* chamada push_back_state_and_time, que é a função de observação que o próprio Odeint usa em Ahnert e Mulansky (2018d). Depois eq_din (EQuações DINâmicas) é criada; essa é a função *system*. O próximo comentário simboliza as manipulações que os dados de entrada sofrem até o cálculo do vetor de estado inicial. Então são declaradas a matriz de resultados y, o vetor de tempos t e o vetor de estado x, que serão variáveis usadas no integrador. É feito, então, o cálculo do vetor de estado. Por fim, define-se o tempo final t_f, o *stepper* runge_kutta_fehlberg78 e o integrador numérico integrate_const é chamado. Os números "0.0" e "0.0001" referem-se a, respectivamente, o tempo inicial de integração e ao passo de integração.

4.4 Implementação das Equações Dinâmicas e do Integrador Numérico

4.4.1 Implementação das Equações Dinâmicas dos 4 Corpos

Esta seção trata da implementação do algoritmo que calcula o P4C no C++ e monta a função *system*, que alimenta o integrate_const. A base dessa implementação computacional são as Equações 4.22, de Celestino (2005). Essas equações estão descritas em formato reduzido pelo somatório e são vetoriais. Sua expansão leva a 12 equações diferenciais de aceleração, 3 para cada um dos corpos. Cada equação para um corpo conta com um termo que é dependente de cada um dos outros três corpos.

$$\ddot{\mathbf{r}}_\mathbf{i} = -\sum_{j=1,j\neq i}^{4} \mu_j \frac{\mathbf{r_i} - \mathbf{r_j}}{||\mathbf{r_i} - \mathbf{r_j}||^3} = \sum_{j=1,j\neq i}^{4} \mu_j \frac{\mathbf{r_j} - \mathbf{r_i}}{||\mathbf{r_i} - \mathbf{r_j}||^3} \tag{4.22}$$

Considere que o processo de integração para obter a posição a partir da segunda derivada da posição se dá pelas Equações 4.23, em que $\mathbf{a_i}$ é o vetor aceleração de um corpo i e $\mathbf{v_i}$ o vetor velocidade de um corpo i.

$$\ddot{\mathbf{r}}_\mathbf{i} = \mathbf{a_i} \Rightarrow \dot{\mathbf{r}}_\mathbf{i} = \mathbf{v_i} \Rightarrow \mathbf{r_i} \tag{4.23}$$

A ordem de um sistema de EDOs pode ser reduzida introduzindo novas variáveis de ordens menores (AHNERT; MULANSKY, 2018d). Levando em conta que a derivada da aceleração

é a velocidade e a velocidade é a derivada da posição, é possível reescrever as Equações 4.23 como as Equações 4.24. A função integrate_const integra, portanto, primeiro a derivada da velocidade, obtendo a velocidade; então, iguala-se a derivada da posição à velocidade e esta é integrada, obtendo-se a posição. Isso permite que um integrador de derivadas de primeira ordem resolva uma equação derivativa de segunda ordem. Esse método foi a base da construção computacional da função de equações dinâmicas deste trabalho.

$$\begin{aligned} \dot{\mathbf{v}}_\mathbf{i} &= \ddot{\mathbf{r}}_\mathbf{i} = \mathbf{a}_\mathbf{i} \\ \dot{\mathbf{r}}_\mathbf{i} &= \mathbf{v}_\mathbf{i} \end{aligned} \tag{4.24}$$

A construção das equações dinâmicas foi divida em duas partes. Primeiro foi elaborado o algoritmo para calcular apenas a dinâmica do P4C, usando as Equações 4.22. Somente depois disso as perturbações de PRS foram consideradas, formando a segunda parte da implementação. Dessa forma, as Equações 4.22 estão livres das perturbações da PRS, mas, do ponto de vista do detrito, sua órbita é perturbada pelas atrações gravitacionais da Lua e do Sol.

Implementar primeiro a dinâmica apenas com perturbações luni-solares sobre o detrito permitiu a construção da estrutura do código do simulador considerando o P4C. Essa estrutura de simulação foi necessária para validar os resultados do POC. Posteriormente, quando foram implementadas as perturbações de PRS, foram calculadas as componentes de aceleração de perturbação provocadas pela PRS e essas foram somadas às equações de movimento do detrito.

A implementação das equações dinâmicas com apenas perturbações luni-solares sobre o detrito é iniciada passando o vetor de estado $\mathbf{X}$ para a função *system*. O vetor de estado é construído conforme a Equação 4.25. O objetivo da função *system* é calcular as derivadas de cada componente deste vetor e alimentar o *stepper*, que realiza um passo da integração.

$$\mathbf{X} = [\mathbf{x}\ \mathbf{v}]^T \tag{4.25}$$

As Equações 4.26 associam a cada componente do vetor de estado uma variável de posição

e velocidade P4C.

$$\begin{aligned}
\mathbf{x_1} &= \mathbf{r_1} \\
\mathbf{x_2} &= \mathbf{r_2} \\
\mathbf{x_3} &= \mathbf{r_3} \\
\mathbf{x_4} &= \mathbf{r_4} \\
\mathbf{v_1} &= \dot{\mathbf{r}}_\mathbf{1} \\
\mathbf{v_2} &= \dot{\mathbf{r}}_\mathbf{2} \\
\mathbf{v_3} &= \dot{\mathbf{r}}_\mathbf{3} \\
\mathbf{v_4} &= \dot{\mathbf{r}}_\mathbf{4}
\end{aligned} \tag{4.26}$$

Note que as componentes do vetor de estado também são vetores, tal que $\mathbf{X}$ é um vetor de vetores. A notação vetorial foi adotada para simplificar a escrita das equações. As posições e velocidades do P4C usadas para construir o vetor de estado são mostradas nas Equações 4.27.

$$\begin{aligned}
\mathbf{r_1} &= [r_{X,1}\ r_{Y,1}\ r_{Z,1}]^T \\
\mathbf{r_2} &= [r_{X,2}\ r_{Y,2}\ r_{Z,2}]^T \\
\mathbf{r_3} &= [r_{X,3}\ r_{Y,3}\ r_{Z,3}]^T \\
\mathbf{r_4} &= [r_{X,4}\ r_{Y,4}\ r_{Z,4}]^T \\
\dot{\mathbf{r}}_\mathbf{1} &= [\dot{r}_{X,1}\ \dot{r}_{Y,1}\ \dot{r}_{Z,1}]^T \\
\dot{\mathbf{r}}_\mathbf{2} &= [\dot{r}_{X,2}\ \dot{r}_{Y,2}\ \dot{r}_{Z,2}]^T \\
\dot{\mathbf{r}}_\mathbf{3} &= [\dot{r}_{X,3}\ \dot{r}_{Y,3}\ \dot{r}_{Z,3}]^T \\
\dot{\mathbf{r}}_\mathbf{4} &= [\dot{r}_{X,4}\ \dot{r}_{Y,4}\ \dot{r}_{Z,4}]^T
\end{aligned} \tag{4.27}$$

As Equações 4.28 calculam as derivadas de $\mathbf{X}$. É possível notar, nestas equações, que o método de redução de ordem das EDOs foi adotado, tal que primeiro são calculadas as derivadas da posição com base na velocidade e, depois, são calculadas as derivadas da

velocidade com base nas equações do P4C.

$$
\begin{aligned}
\frac{d\mathbf{x_1}}{dt} &= \mathbf{v_1} \\
\frac{d\mathbf{x_2}}{dt} &= \mathbf{v_2} \\
\frac{d\mathbf{x_3}}{dt} &= \mathbf{v_3} \\
\frac{d\mathbf{x_4}}{dt} &= \mathbf{v_4} \\
\frac{d\mathbf{v_1}}{dt} &= \mu_2 \frac{\mathbf{r_2} - \mathbf{r_1}}{||\mathbf{r_1} - \mathbf{r_2}||^3} + \mu_3 \frac{\mathbf{r_3} - \mathbf{r_1}}{||\mathbf{r_1} - \mathbf{r_3}||^3} + \mu_4 \frac{\mathbf{r_4} - \mathbf{r_1}}{||\mathbf{r_1} - \mathbf{r_4}||^3} \\
\frac{d\mathbf{v_2}}{dt} &= \mu_1 \frac{\mathbf{r_1} - \mathbf{r_2}}{||\mathbf{r_2} - \mathbf{r_1}||^3} + \mu_3 \frac{\mathbf{r_3} - \mathbf{r_2}}{||\mathbf{r_2} - \mathbf{r_3}||^3} + \mu_4 \frac{\mathbf{r_4} - \mathbf{r_2}}{||\mathbf{r_2} - \mathbf{r_4}||^3} \\
\frac{d\mathbf{v_3}}{dt} &= \mu_1 \frac{\mathbf{r_1} - \mathbf{r_3}}{||\mathbf{r_3} - \mathbf{r_1}||^3} + \mu_2 \frac{\mathbf{r_2} - \mathbf{r_3}}{||\mathbf{r_3} - \mathbf{r_2}||^3} + \mu_4 \frac{\mathbf{r_4} - \mathbf{r_3}}{||\mathbf{r_3} - \mathbf{r_4}||^3} \\
\frac{d\mathbf{v_4}}{dt} &= \mu_1 \frac{\mathbf{r_1} - \mathbf{r_4}}{||\mathbf{r_4} - \mathbf{r_1}||^3} + \mu_2 \frac{\mathbf{r_2} - \mathbf{r_4}}{||\mathbf{r_4} - \mathbf{r_2}||^3} + \mu_3 \frac{\mathbf{r_3} - \mathbf{r_4}}{||\mathbf{r_4} - \mathbf{r_3}||^3}
\end{aligned}
\tag{4.28}
$$

Todas essas derivadas são agregadas num vetor de estado derivado, como mostrado na Equação 4.29.

$$
\frac{d\mathbf{X}}{dt} = \left[\frac{d\mathbf{x}}{dt} \; \frac{d\mathbf{v}}{dt} \right]^T \tag{4.29}
$$

A construção dos vetores de estado e de estado derivado, como apresentada até então, descreve o método usado para criar funções do tipo *system*, que também podem ser chamadas de integrandos. O *stepper* utilizado recebe o integrando e realiza um passo t_h da integração, calculando os resultados de $d\mathbf{X}/dt$, para depois sobrescrever, com eles, o vetor $\mathbf{X}$, permitindo que o processo possa ser reiniciado. Isso é feito até que seja atingido o tempo final da integração. Para sintetizar esse método, a Figura 11 foi elaborada, representando o processo de determinar o vetor de estado inicial; calcular o vetor de estado derivado; medir o estado com a função de obervação; e recalcular o estado derivado a partir dos resultados da integração.

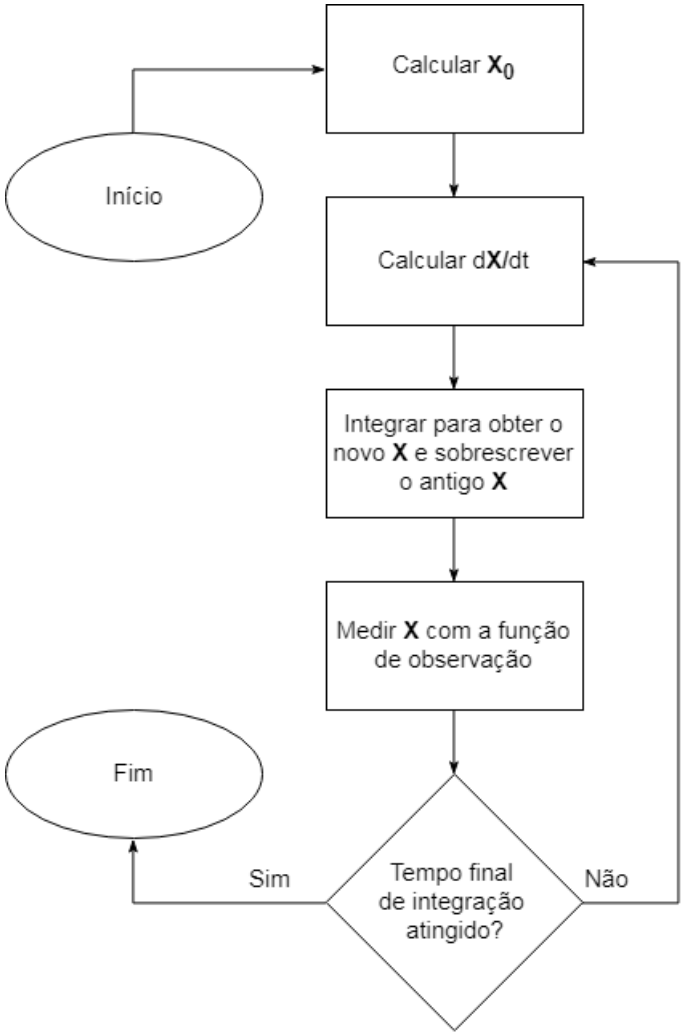

Figura 11 – Processo de integração.

Os resultados do integrador saem referenciados em $OXYZ$. Para calcular as posições e velocidades do detrito com relação à Terra, a partir da matriz de resultados da integração, basta subtrair as k-ésimas posições e velocidades do detrito das k-ésimas posições e velocidades da Terra. Isto é: $\mathbf{r_{24}}_k = \mathbf{r_4}_k - \mathbf{r_2}_k$.

Os maiores desafios encontrados nessa etapa do trabalho estavam relacionados com fazer o sistema de alimentação do integrador e integração funcionarem, retornando resultados coerentes. Para a obtenção de um funcionamento satisfatório, foram definidos o Runge-Kutta-Fehlberg 78 como *stepper* e passo de integração $dt = 0{,}0001\ dia$.

4.4.2 Implementando a Perturbação de Radiação Solar

Depois de a integração numérica com o POC ter sido validada com os resultados de POF considerando apenas perturbações luni-solares, foi implementada a perturbação de radiação solar. A PRS foi considerada nas equações dinâmicas calculando-se as componentes

de aceleração que essa perturbação provoca num detrito e somando essas componentes nas equações dinâmicas do detrito.

De acordo com Fieseler (1998), a força de pressão de radiação solar $\mathbf{F_{4,PRS}}$ pode ser descrita pela Equação 4.30. Esta função de PRS não é a mesma utilizada em POF. Nesta equação, r' é o coeficiente de reflectividade e depende do detrito, variando de 0 a 1; $w = 1360\ W/m^2$ a uma unidade astronômica ($1\ UA$) e é chamado de Constante Solar; c é a velocidade da luz; A é a área superficial do detrito; e Θ é o ângulo entre os fótons incidentes e a normal da superfície iluminada de área A. O coeficiente γ foi introduzido nesta equação para considerar as regiões de penumbra ($\gamma = 0{,}5$) e umbra ($\gamma = 0$). Entretanto, apesar de constar na equação, o artifício de γ não foi usado no simulador POC. O versor $\hat{\mathbf{a}}_{\mathbf{PRS}}$ indica a direção segundo a qual os fótons incidem no detrito.

$$\mathbf{F_{4,PRS}} = \gamma \frac{(1 + r')wA\cos^2\Theta}{c} \hat{\mathbf{a}_{\mathbf{PRS}}} \tag{4.30}$$

Dividindo a Equação 4.30 pela massa do detrito é obtida a Equação 4.31. Desta maneira, essa equação está num melhor formato, visto que lida com a aceleração $\mathbf{a_{4,PRS}}$ provocada pela PRS ao invés da força, bastando, assim, somar as componentes de aceleração calculadas nas equações de movimento do detrito. As componentes de $\mathbf{a_{4,PRS}}$ são mostradas na Equação 4.32.

$$\mathbf{a_{4,PRS}} = \gamma \frac{(1 + r')w\phi}{c} \hat{\mathbf{a}_{\mathbf{PRS}}} \tag{4.31}$$

$$\hat{\mathbf{a}_{\mathbf{4,PRS}}} = [\hat{a}_{X,4,PRS}\ \ \hat{a}_{Y,4,PRS}\ \ \hat{a}_{Z,4,PRS}]^T \tag{4.32}$$

Observe que ao dividir a Equação 4.30 pela massa m_4 do detrito, uma nova variável surge na Equação 4.31: ϕ, que é a razão área-massa do detrito, tal que $\phi = A/m_4$. Para validação do POC com relação ao POF, o valor padrão adotado neste trabalho para ϕ foi 50 m^2/kg. O detrito foi considerado tão pequeno quanto uma partícula, de forma que o ângulo $\Theta = 0^o$, fazendo com que $\cos^2\Theta = 1$. Além disso, foi considerado que o detrito reflete toda a luz do Sol incidente sobre ele, tal que $r' = 1$, resultando no caso de maior troca de momento.

Levando em conta as considerações feitas, as variáveis $\gamma(1 + r')w\phi/c$ são uma constante que representa a magnitude da aceleração de PRS. O versor $\hat{\mathbf{a}}_{\mathbf{PRS}}$ distribui essa magnitude entre X, Y e Z. Segundo Celestino (2005), as componentes desse versor podem ser calculadas com as Equações 4.33, em que λ_1 é a longitude do Sol, n_1 o movimento

médio do Sol com relação à Terra, t o tempo de integração e ϵ' o ângulo eclíptico.

$$\begin{aligned} \hat{a}_{X,4,PRS} &= -\cos(\lambda_1 + n_1 t) \\ \hat{a}_{Y,4,PRS} &= -\operatorname{sen}(\lambda_1 + n_1 t)\cos(\epsilon') \\ \hat{a}_{Z,4,PRS} &= -\operatorname{sen}(\lambda_1 + n_1 t)\operatorname{sen}(\epsilon') \end{aligned} \tag{4.33}$$

4.4.3 Gerando os Arquivos de Resultados

Os dados da matriz de resultados foram escritos em arquivos de resultados. Assim, foi possível tratar esses dados, criar gráficos e correlações numa etapa de processamento pós-integração, por meio de programas alternativos como o Matplotlib e a IDE Spyder (Python, Anaconda).

Para escrever arquivos de dados em C++, basta seguir conforme apresentado abaixo. Primeiro, na linha 1, é necessário criar um arquivo de resultados chamado "arquivo_de_resultados.csv"; o "w+" indica que esse arquivo receberá dados e a extensão é o *comma separated values* ".csv". O comentário da linha 3 simboliza o programa que gera os dados de tempo t, vetor de estado $\mathbf{x}$, a, e, i, Ω, ω e f. A função fprintf() recebe como argumento o arquivo aberto, o formato segundo o qual o arquivo será escrito
("%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f\n")
e os dados a serem escritos, separados por vírgula. Sobre o formato, cada %f simboliza uma entrada de dado de tipo ponto flutuante e \n é o comando para pular uma linha. No fim, é necessário fechar o arquivo com fclose().

```
1 FILE *arq = fopen("arquivo_de_resultados.csv", "w+");
2
3 // ... codigo gerando dados: t, vetor x, a, e, i,
4                                 // Omega, omega, f
5
6 fprintf(arq, "%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f\n", t, \
7 x[0], x[1], x[2], x[3], x[4], x[5], a, e, i, Omega, omega, f);
8
9 fclose(arq);
```

Quanto ao código apresentado acima para gerar arquivos em C++, vale ser feita uma observação. O método apresentado conta com apenas uma inserção de dados no arquivo de resultados. É necessário, portanto, implementá-lo em conjunto com os laços de repetição disponíveis na linguagem ou dentro do processo de integração, que repetir-se-á até que t_f seja atingido.

4.4.4 Gráficos com a Biblioteca Matplotlib do Python

Após a integração, foi necessário gerar os gráficos dos resultados da simulação. Isso foi feito com a biblioteca Matplotlib do Python. Para tanto, foi implementado um *script* que acessa os arquivos de resultados e plota os gráficos dos elementos orbitais, conforme a Figura 12.

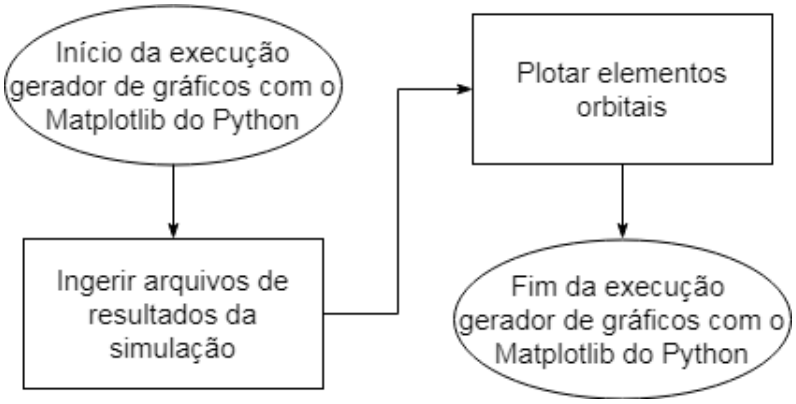

Figura 12 – Processo de geração de gráficos com o Matplotlib.

4.5 Validação de Resultados e Desempenho do Simulador

A estratégia adotada para validar o propagador orbital construído foi efetuar uma comparação entre os resultados do POC e do POF. Para tanto, foi necessário configurar os dois propagadores com as mesmas condições inciais e simular para um mesmo tempo de integração. A validade dos resultados do POC foi atestada com referência nos resultados do POF.

Na etapa de validação, as condições iniciais para o detrito, Terra, Lua e Sol foram as mesmas em todas as simulações. Essas condições, não normalizadas, foram apresentadas na Tabela 3 e, juntas, formam o vetor de estado inicial. As mesmas condições foram apresentadas, normalizadas, na Tabela 6. Além disso, para a validação, o detrito foi assumido com $\phi = 50\ m^2/kg$ e considerado como partícula.

Para calcular a correlação entre os dados de POC e POF, foi elaborado um *script* em Python, usando a IDE Spyder, da plataforma Anaconda. O programa, simples, funciona segundo o esquema da Figura 13. Ao ser executado o programa, os resultados das simulações do POC e do POF são acessados e carregados como *dafatrames*, um tipo de tabela para análise de dados, pela biblioteca Pandas, de forma que cada coluna represente todos os dados de uma componente de um vetor posição ou velocidade de um corpo. A comparação é feita entre as colunas que representam as mesmas componentes nos dois *dataframes*.

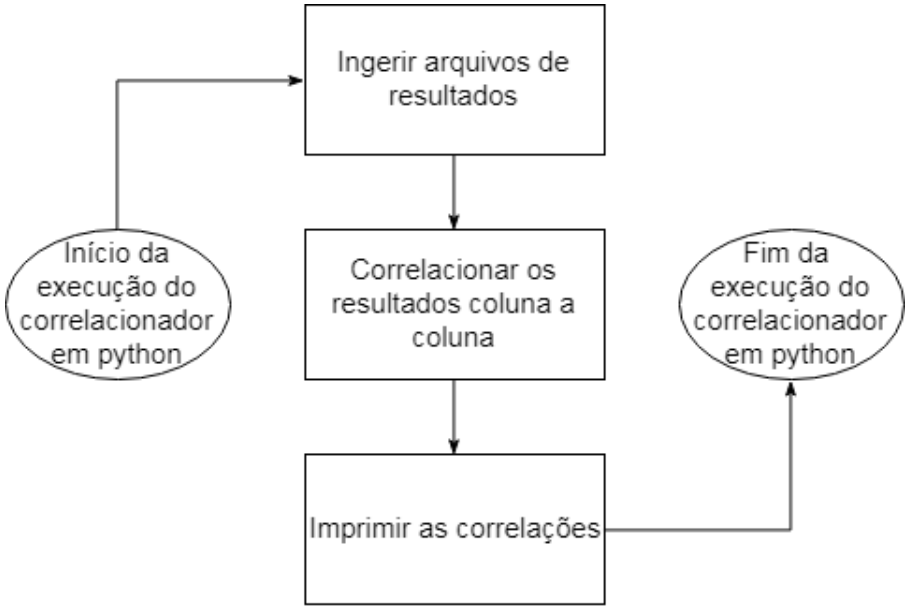

Figura 13 – Esquema de funcionamento do algoritmo em Python para calcular a correlação entre os resultados do propagador em C++ e Fortran.

Para comparar os resultados, é calculada a correlação entre as colunas dos dois *dataframes* com a Equação 4.34, que usa o modelo de correlação de Pearson (DALININA, 2017; BRYANT, 2018). Nesta equação, $\rho_{x,y}$ é o valor da correlação entre dois volumes de dados x e y, N é o número de linhas de resultados, $\hat{x}$ e $\hat{y}$ são suas médias. Bons resultados para POC, medindo a correlação com POF, devem ser tão próximos de 1 quanto possível.

$$\rho_{x,y} = \frac{\sum_{i=1}^{N}(x_i - \hat{x})(y_i - \hat{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \hat{x})^2(y_i - \hat{y})^2}} \tag{4.34}$$

Os resultados da correlação entre os dados são plotados no console do Spyder. De lá, foram tabulados e mostrados nas seções subsequentes. O Python, com esta IDE, mostrou funcionalidade satisfatória, processando grandes volumes de dados (os resultados das duas simulações tinham 25 colunas por cerca de 3600000 linhas cada, para 366 dias de integração) em menos de dois minutos.

5 Resultados do Simulador em C++

5.1 Validação das Perturbações Luni-Solares

A primeira validação realizada foi a do código que considerava apenas perturbações luni-solares atuando sobre o detrito. Das validações do simulador, essa foi a mais importante, tendo em vista que o POC nessa configuração serve de estrutura para a implementação de outras perturbações. O intervalo de integração utilizado foi de 50 dias, as condições iniciais do vetor de estado foram as estabelecidas na Tabela 6, sendo as simulações executadas, portanto, com dados normalizados. O detrito foi simulado com $\phi = 50\ m^2/kg$.

Os primeiros testes de correlação consistiram em executar a simulação uma vez para cada tipo de *stepper*: *basic stepper, error stepper, controlled stepper* e *dense output stepper*. Foi medida, para cada caso, a correlação apenas entre as posições orbitais, uma vez que, se estas estivessem coerentes, como foram calculadas a partir da integração da velocidade, as velocidades também deveriam estar.

Os resultados de correlação entre os propagadores POC e POF para cada *stepper* são apresentados na Tabela 8. O valor "nan" significa *not a number*. Ele aparece em algumas células nesta e nas próximas seções porque todos os valores das componentes às quais foi atribuído o "nan" são zero, gerando uma inconsistência matemática na Equação 4.34. Os dados dessas componentes foram verificados e são compatíveis.

Com relação aos *steppers*, excluindo as as células "nan", nota-se que as correlações tem valor 1 ou próximo de 1, o que era desejado. Isso significa que os resultados entre os dois propagadores são quase idênticos, com exceção de algumas componentes. Dentre essas, $r_{Z,4}$, a componente Z da posição do detrito, mostrou maior sensibilidade à variação dos *steppers*.

Tabela 8 – Correlação entre três diferentes *steppers*: *stepper*, *error stepper* e *controlled stepper*.

Posição	***Basic Stepper*: Runge-Kutta 4**	***Error Stepper*: Fehlberg 78**	***Controlled Stepper*: Bulirsch-Stoer**	***Dense Output Stepper*: Bulirsch-Stoer Dense Output**
$r_{X,1}$	1,000000000000000	1,000000000000000	1,000000000000000	nan
$r_{Y,1}$	1,000000000000000	1,000000000000000	1,000000000000000	nan
$r_{Z,1}$	nan	nan	nan	nan
$r_{X,2}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Y,2}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Z,2}$	0,999999999995000	0,999999999995000	0,999999999995000	nan
$r_{X,3}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Y,3}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Z,3}$	0,999999999999000	0,999999999999000	0,999999999999000	1,000000000000000
$r_{X,4}$	1,000000000000000	1,000000000000000	1,000000000000000	0,999999999805000
$r_{Y,4}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Z,4}$	0,999999997944000	0,999999997451000	0,999999997619000	0,999999999716000

A Tabela 9 mostra tempos (Tempo 1, Tempo 2 e Tempo 3), em segundos, de três testes de simulação feitos com cada *stepper*. A média e o desvio padrão de cada *stepper* foram computados.

Tabela 9 – Desempenho de vários *steppers* comparados com a simulação de Celestino (2005).

Processo de Integração	**Tempo 1 (s)**	**Tempo 2 (s)**	**Tempo 3 (s)**	**Tempo médio (s)**	**Desvio Padrão (s)**
Stepper Runge-Kutta 4	38,631	38,546	41,043	39,407	1,091
Error Stepper Fehlberg 78	20,531	20,122	20,190	20,281	0,167
Controlled Stepper Bulirsch-Stoer	31,532	31,522	30,540	31,198	0,439
Dense Output Stepper: Bulirsch-Stoer Dense Output	5,697	5,691	5,657	5,682	0,016
Referência do Fortran	16,190	15,378	15,626	15,731	0,306

Comparando os resultados de correlação para cada *stepper* na Tabela 8, nota-se que o Fehlberg 78 teve a menor correlação para a componente $r_{Z,4}$. Apesar disso, a magnitude da diferença entre o Fehlberg 78 e os *steppers*:

- Runge-Kutta 4 foi de: 4,93E-10; e
- Bulirsch-Stoer foi de: 1,68E-10.

Esses são valores irrisórios se comparados com os valores de correlação, de forma que isso não teve influência na escolha dos *stepper*. Entretanto, observando os tempos médios de integração para cada *stepper* na Tabela 9, nota-se que, com exceção do Bulirsch-Stoer *Dense Output*, a simulação mais veloz era feita pelo Fehlberg 78, motivando sua escolha para o POC. Note que todos os tempos, com exceção do tempo do Bulirsch-Stoer *Dense Output* foram inferiores ao propagador POF.

Tendo em vista os tempos de integração, por que não, então, selecionar o Bulirsch-Stoer *Dense Output*, que foi cerca de três vezes mais rápido que Gauss-Radau de POF e

quatro vezes mair rápido que Fehlberg 78? O motivo foi que o Bulirsch-Stoer *Dense Output* toma passos maiores de integração e interpola os resultados (AHNERT; MULANSKY, 2018f), o que não era desejado neste trabalho, tendo em vista a perda de controle do usuário sobre o passo de integração gerada por esse método.

A Tabela 10 mostra o impacto do tamanho do passo de integração sobre a correlação dos resultados. Foram simulados quatro passos diferentes: $dt = [0,1\ 0,01\ 0,001\ 0,0001]$ *dia*, fixando o *stepper* Fehlberg 78.

Tabela 10 – Correlação entre os resultados dos propagadores orbitais em C++ e Fortran.

Posição	dt = **0,0001** (*dia*)	dt = **0,001** (*dia*)	dt = **0,01** (*dia*)	dt = **0,1** (*dia*)
$r_{X,1}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Y,1}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Z,1}$	nan	nan	nan	nan
$r_{X,2}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Y,2}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Z,2}$	0,999999999995000	0,999999999987000	0,999999999986000	0,999999999929000
$r_{X,3}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Y,3}$	1,000000000000000	1,000000000000000	1,000000000000000	1,000000000000000
$r_{Z,3}$	0,999999999999000	0,999999999998000	0,999999999999000	1,000000000000000
$r_{X,4}$	1,000000000000000	1,000000000000000	0,999999893158000	0,999796020010000
$r_{Y,4}$	1,000000000000000	1,000000000000000	0,999999982836000	0,999900116223000
$r_{Z,4}$	0,999999997451000	0,999999995235000	0,233313419764000	-0,308185829989000

Novamente, a maior sensibilidade da correlação foi relacionada ao movimento do detrito, tal que quanto maior o passo de integração, menor a correlação. Para $dt = 0,1$ *dia*, a correlação foi negativa, indicando distanciamento entre os resultados de POC e POF. A componente de maior sensibilidade, como no teste anterior, foi a posição no eixo Z do detrito.

O passo de integração estabelecido, com base nesses resultados, foi $dt = 0,0001$ *dia*. Passos menores resultavam em tempos indesejados de simulação e, dependendo do *stepper*, em *overflow*. Segundo as correlações obtidas, também poderia ter sido utilizado $dt = 0,001$ *dia* para obter bons resultados.

5.2 Validação das Perturbações Luni-Solares e de PRS

A última validação de dados foi feita considerando o modelo completo implementado: PRS e perturbações luni-solares afetando o detrito, com intervalo de integração e razão área-massa iguais aos da seção anterior. Os resultados para este caso são mostrados na Tabela 11.

Tabela 11 – Correlação entre os resultados obtidos com o C++ e os de Celestino (2005), considerando pressão de radiação solar.

Posição	**Correlação**
$r_{X,1}$	1,000000000000000
$r_{Y,1}$	1,000000000000000
$r_{Z,1}$	nan
$r_{X,2}$	1,000000000000000
$r_{Y,2}$	1,000000000000000
$r_{Z,2}$	0,999999999995000
$r_{X,3}$	1,000000000000000
$r_{Y,3}$	1,000000000000000
$r_{Z,3}$	0,999999999999000
$r_{X,4}$	0,999999997281000
$r_{Y,4}$	0,999999999620000
$r_{Z,4}$	0,962335460982000

Observe que, novamente, a maior sensibilidade ocorreu na componente $r_{Z,4}$, com correlação cerca de 0,14 menor que no caso da simulação apenas com perturbações luni-solares. As demais correlações foram todas com valores acima de 0,999999999, indicando coerência entre os resultados de POC e POF. É interessante notar que as correlações foram satisfatórias mesmo com as funções de PRS entre os propagadores sendo diferentes, o que indica que o modelo da Equação 4.31 e que as hipóteses assumidas para o detrito fazem sentido, de acordo com os resultados do POF.

5.3 Simulação Apenas com Perturbações Luni-Solares

Esta seção apresenta simulações do POC considerando apenas perturbações luni-solares agindo sobre os detritos, sem que fossem computados os efeitos da PRS. O intervalo de integração foi de dois anos. A Tabela 12 mostra os valores iniciais de simulação utilizados.

Tabela 12 – Valores iniciais de simulação considerando apenas perturbações luni-solares

Grandeza	**Valor**
$a\ (km)$	42378
e	0,01
$i\ (graus)$	45
$\Omega\ (graus)$	45
$\omega\ (graus)$	45
$f\ (graus)$	0
$\phi\ (m^2/kg)$	20

A Figura 14 mostra o comportamento de a para perturbações luni-solares. Pode ser observado comportamento periódico do semieixo maior.

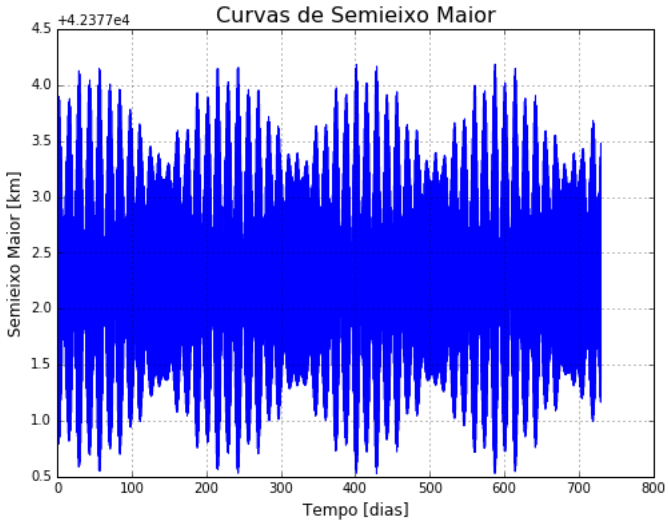

Figura 14 – Simulação do comportamento de a.

A Figura 15 mostra o comportamento de e. Os valores de e apresentam comportamento oscilatório, mas de pequenas amplitudes, além de uma pequena tendência de crescimento de e.

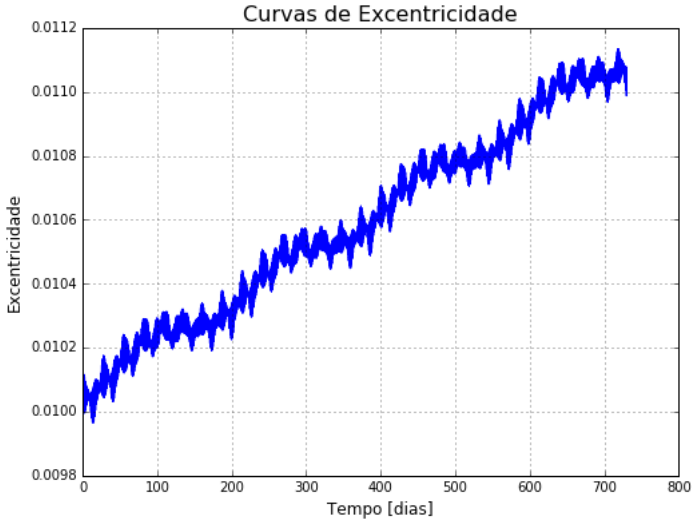

Figura 15 – Simulação do comportamento de e.

A Figura 16 mostra o comportamento de i. Nota-se um comportamento oscilatório e crescente na curva de i, mas de pequenas magnitudes.

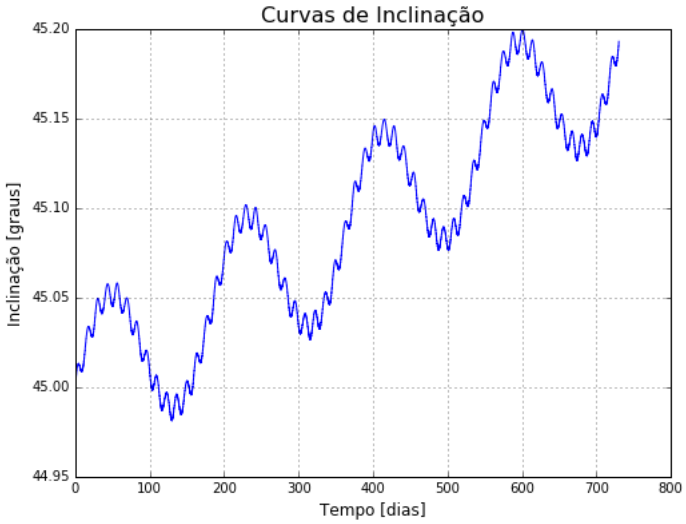

Figura 16 – Simulação do comportamento de i.

A Figura 17 mostra o comportamento de Ω. O argumento do nodo ascendente decresce com pequenas oscilações ao longo da integração.

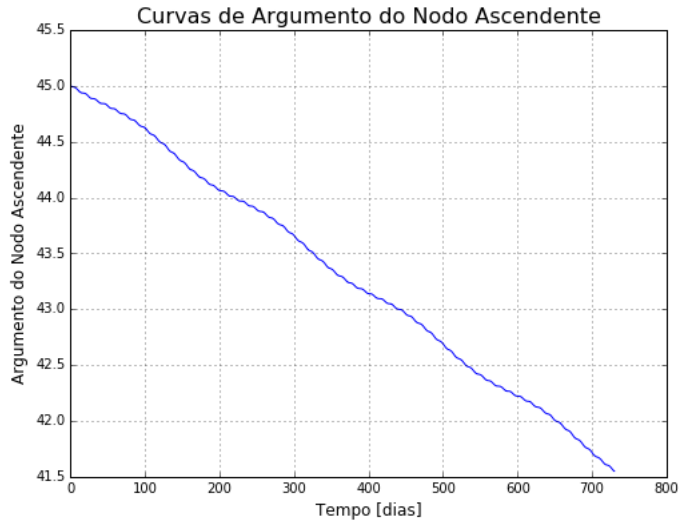

Figura 17 – Simulação do comportamento de Ω.

A Figura 18 mostra o comportamento de ω. Notam-se oscilações e comportamento crescente de pequenas magnitudes na curva de ω.

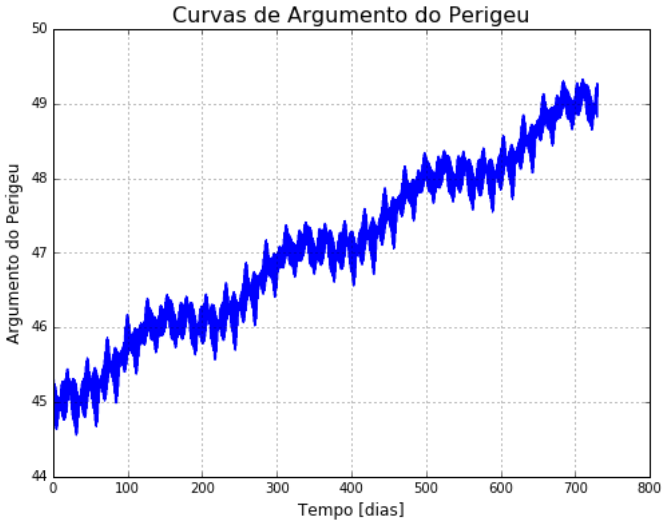

Figura 18 – Simulação do comportamento de ω.

5.4 Simulação com Diferentes Razões Área-Massa Iniciais

Esta seção apresenta resultados para simulações utilizando diferentes razões área-massa iniciais e um período de integração de 2 anos. A Tabela 13 apresenta os valores dos elementos fixados na simulação. A Tabela 14 mostra os valores iniciais dos ϕ usados.

Tabela 13 – Condições iniciais fixas para simulação.

Grandezas com valores fixos	
$a\ (km)$	42378
e	0,01
$i\ (graus)$	45
$\Omega\ (graus)$	45
$\omega\ (graus)$	45
$f\ (graus)$	0

Tabela 14 – Diferentes valores de ϕ simulados.

Valores de razão área-massa	
$\phi\ (m^2/kg)$	0,0001
$\phi\ (m^2/kg)$	10
$\phi\ (m^2/kg)$	20

A Figura 19 mostra o comportamento de a para valores diferentes de ϕ iniciais. A linha vermelha corresponde à simulação para um baixo valor de ϕ, adotado como 0,0001 m^2/kg. Aumentar os valores de ϕ aumenta os efeitos das perturbações sobre o movimento do detrito.

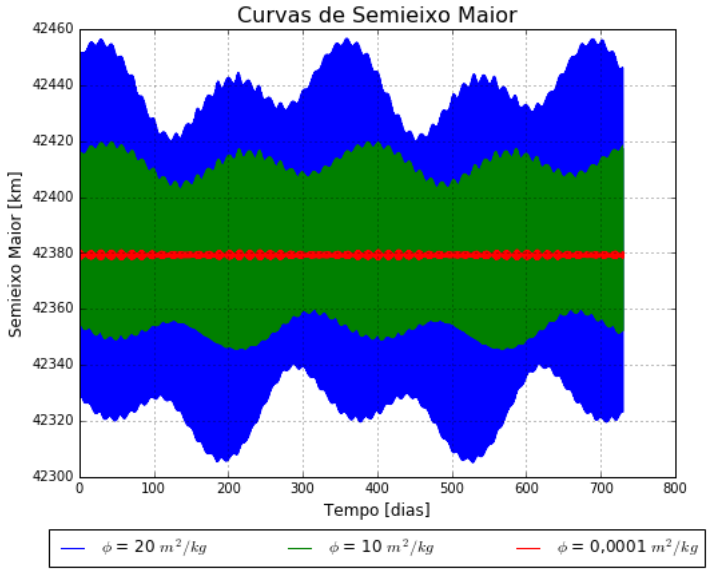

Figura 19 – Simulação do comportamento de a para diferentes ϕ iniciais.

A Figura 20 mostra o comportamento de e para valores iniciais diferentes de ϕ. Observa-se que para maiores valores de ϕ o comportamento orbital do detrito é mais afetado. No caso de $\phi = 0{,}0001\ m^2/kg$, na escala do gráfico, não podem ser identificadas variações significativas.

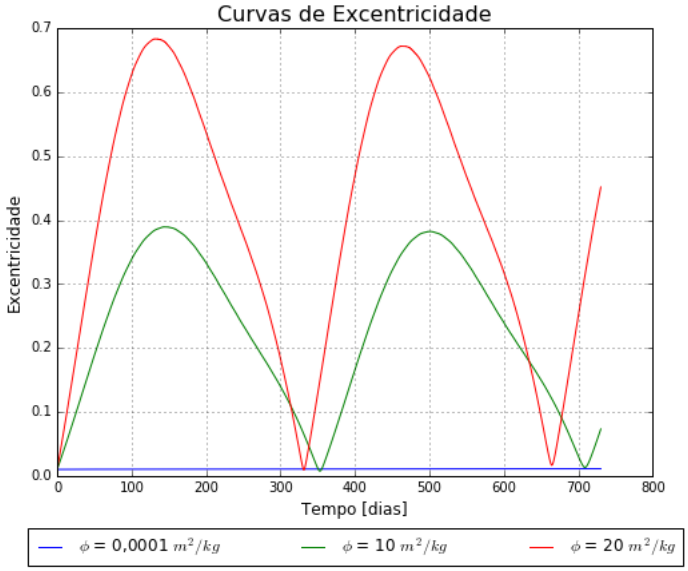

Figura 20 – Simulação do comportamento de e para diferentes ϕ iniciais.

A Figura 21 mostra o comportamento de i para valores diferentes de ϕ iniciais. Nota-se uma tendência crescente, apesar de pequena, no comportamento global das curvas de i. Os efeitos das perturbações são maiores para ϕ maiores.

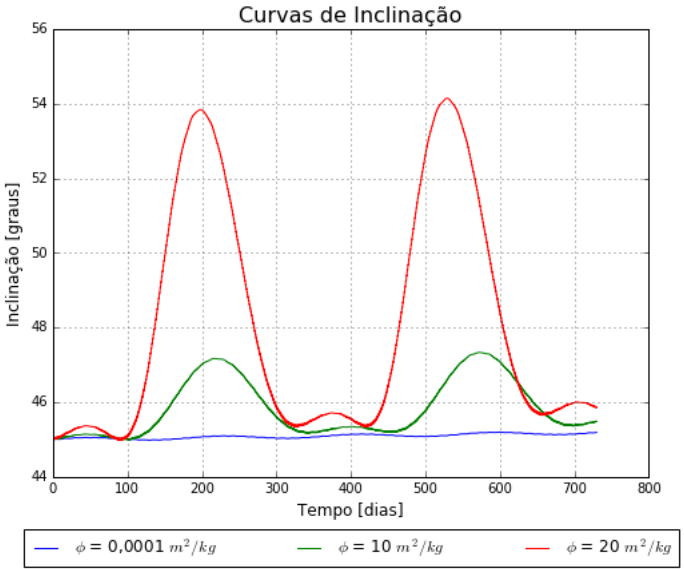

Figura 21 – Simulação do comportamento de i para diferentes ϕ iniciais.

A Figura 22 mostra o comportamento de Ω para valores diferentes de ϕ iniciais. É observado um comportamento decrescente em Ω ao longo do tempo. Os efeitos são maiores para maiores ϕ.

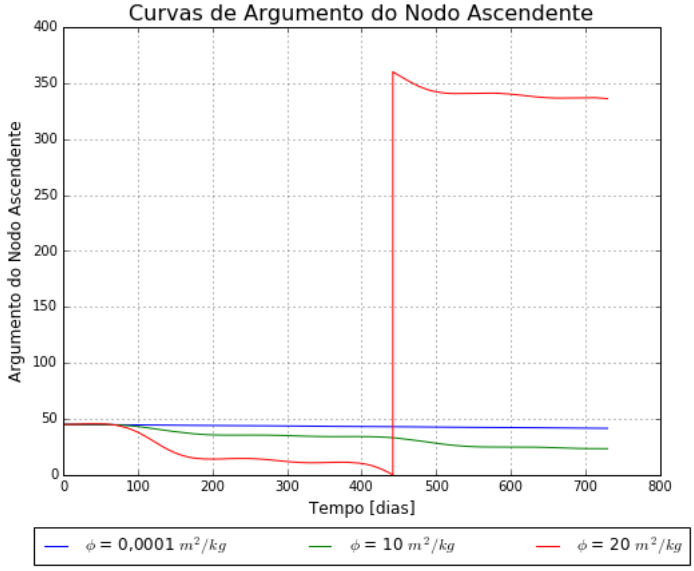

Figura 22 – Simulação do comportamento de Ω para diferentes ϕ iniciais.

A Figura 23 mostra o comportamento de ω para valores diferentes de ϕ iniciais. O valor do argumento do perigeu é crescente no tempo. Para maiores ϕ, há um aumento na velocidade da variação desse elemento. Nota-se, também, que essa velocidade aumenta ao longo da integração.

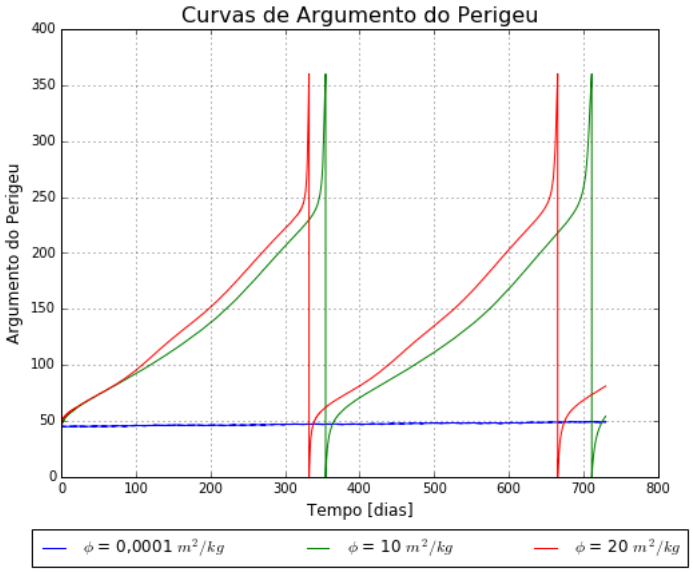

Figura 23 – Simulação do comportamento de ω para diferentes ϕ iniciais.

5.5 Simulação com Diferentes Inclinações Iniciais

Esta seção apresenta resultados para simulações utilizando diferentes valores iniciais de inclinação e um período de integração de 2 anos. A Tabela 15 apresenta os valores dos elementos fixados na simulação. A Tabela 16 mostra os valores iniciais dos i usados.

Tabela 15 – Condições iniciais fixas para simulação.

Grandezas com valores fixos	
ϕ (m^2/kg)	20
a (km)	42378
e	0,01
Ω $(graus)$	45
ω $(graus)$	45
f $(graus)$	0

Tabela 16 – Diferentes valores de i simulados.

Valores de inclinação	
i $(graus)$	30
i $(graus)$	45
i $(graus)$	60

A Figura 24 mostra o comportamento de a para valores diferentes de i iniciais. Aumentar a inclinação inicial provoca variações diferentes entre as curvas de comportamento de a, deslocando-as para cima e diminuindo a amplitude das oscilações.

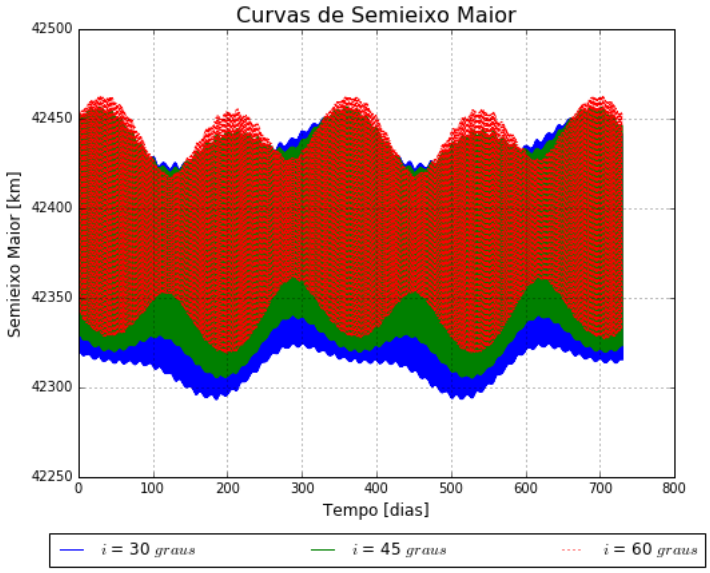

Figura 24 – Simulação do comportamento de a para diferentes i iniciais.

A Figura 25 mostra o comportamento de e para diferentes valores iniciais de i. A curva azul corresponde a $i = 30\ graus$ no início da integração. Aumentar o valor de i inicial reduz a amplitude da variação da inclinação e provoca primeiro um decréscimo na velocidade de queda do valor após os máximos locais, para depois sofrer um atraso, de forma que as curvas atingem seus mínimos locais em tempos similares.

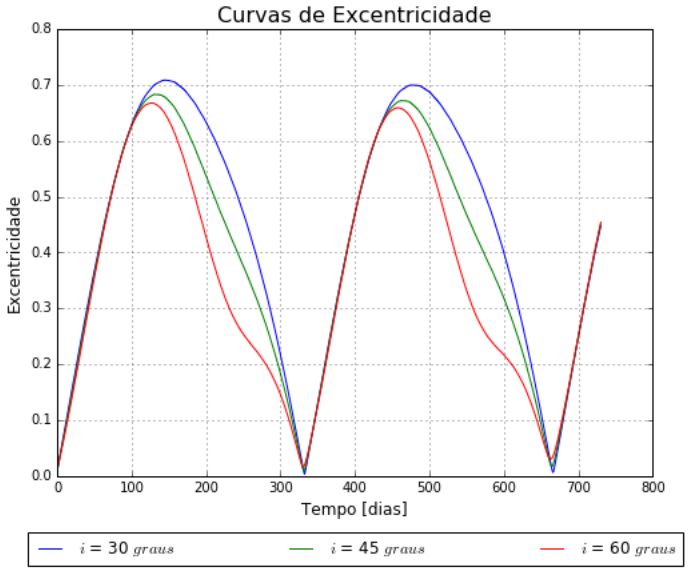

Figura 25 – Simulação do comportamento de e para diferentes i iniciais.

A Figura 26 mostra o comportamento de i para diferentes valores iniciais de i. Nota-se que aumentar o valor de i inicial desloca a curva do comportamento de i para cima e reduz as amplitudes das variações das curvas.

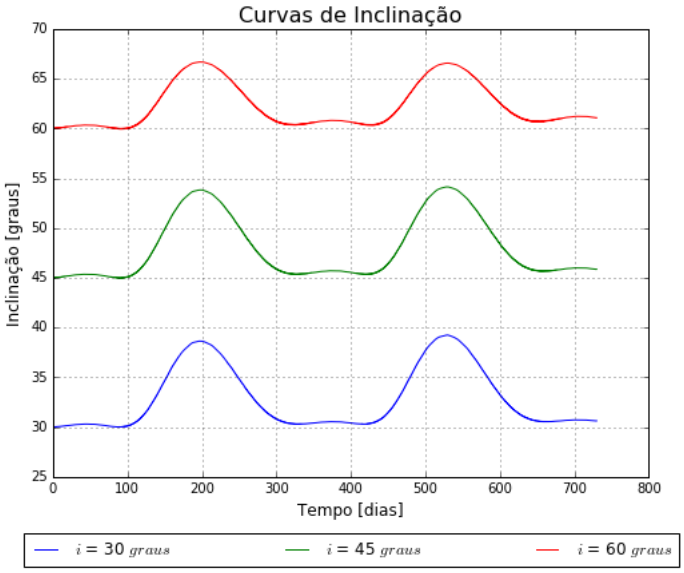

Figura 26 – Simulação do comportamento de i para diferentes i iniciais.

A Figura 27 mostra o comportamento de Ω para diferentes valores iniciais de i. Notam-se pequenas variações entre as curvas de Ω, mas nada significativo para a escala da figura.

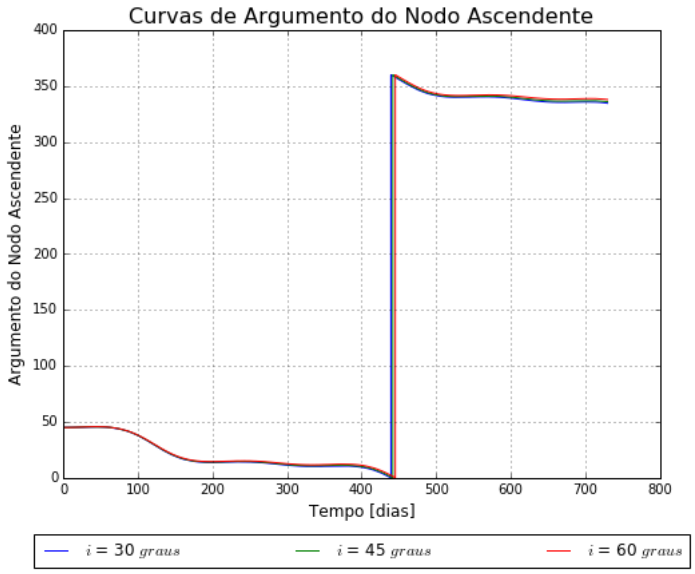

Figura 27 – Simulação do comportamento de Ω para diferentes i iniciais.

A Figura 28 mostra o comportamento de ω para diferentes valores iniciais de i. Notam-se variações maiores para maiores valores de i iniciais. Apesar disso, os períodos das curvas de ω são similares.

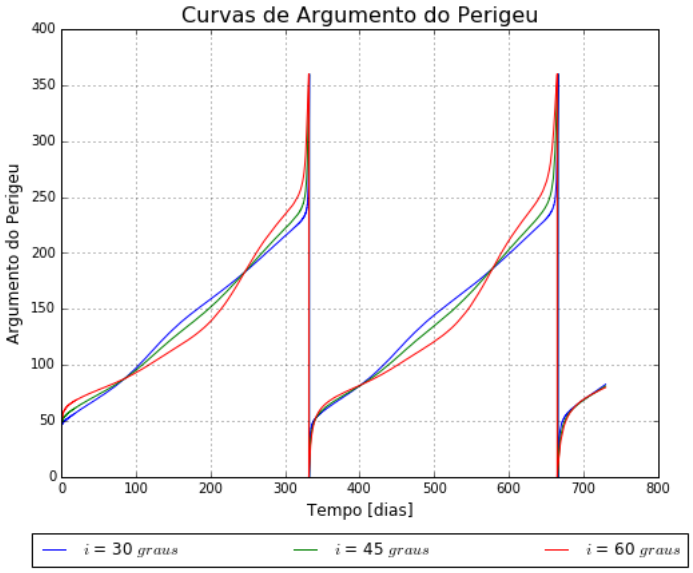

Figura 28 – Simulação do comportamento de ω para diferentes i iniciais.

Segundo Pardal et al. (2011), as perturbações luni-solares afetam especialmente Ω e ω, causando precessão nas órbitas. A PRS provoca perturbações periódicas e seculares nos elementos Ω, ω e M, além de perturbações seculares em a, e e i. Dessa forma, o comportamento das curvas obtidas para os elementos orbitais tanto nesta quanto na seção anterior são coerentes.

5.6 Aplicação do POC em um Caso Real: SGDC

Esta seção apresenta três simulações realizadas para o SGDC. A primeira considera a razão área-massa máxima que o satélite pode ter em seu estado atual. A segunda e a terceira supõem que é possível, por algum dispositivo externo, como uma vela solar, aumentar a razão área-massa. A integração foi feita por dois anos. O objetivo dessas simulações foi verificar se havia como retirar de órbita um satélite como o SGDC usando as perturbações modeladas e proramadas no POC.

Segundo dados da Visiona (2017), o SGDC pesa 5735 kg, tem envergadura de 37 m e altura de 7 m. Assumindo idealmente um formato retangular, o satélite teria uma área A de aproximadamente 259 m^2. Isso leva a uma razão área-massa de $\phi = 0{,}04516$ m^2/kg, que é a maior que o satélite pode ter sem auxílio de um equipamento externo, segundo as dimensões e hipóteses assumidas. Foi considerado, também, que o dispositivo tem coeficiente de reflectividade $r' = 1$, além de um controle que faz com que o ângulo entre o vetor normal da superfície que recebe os fótons e a linha de incidência dos fótons se mantivesse sempre igual a zero grau.

A Tabela 17 mostra os valores dos ϕ usados nas simulações. Além disso, a coluna A

refere-se às áreas que um dispositivo, como uma vela solar, precisaria ter para corresponder a cada ϕ, levando em conta a massa do SGDC. Neste caso, a massa do dispositivo externo foi desprezada. A coluna$\sqrt{A}$ é uma estimativa da dimensão do lado que uma vela quadrada deveria ter para corresponder a A. Ressalta-se que esses valores são hipotéticos, não consideram as tecnologias reais envolvidas na construção de uma vela solar e servem apenas para fins de simulação.

Tabela 17 – Razões área massa para o SGDC.

Teste	ϕ (m^2/kg)	A (m^2)	$\sqrt{A}$ (m)
1	0,04516	259	16,09
2	20	114700	338,67
3	50	286750	535,49

A Tabela 18 mostra as condições iniciais, em termos de elementos orbitais, que foram utilizadas para simular a evolução orbital do SGDC. Os elementos orbitais do satélite foram obtidos dos dados de *two-line elements* (TLE) do NORAD (KELSO, 2018), medidos no dia 24/08/2018, às 06:43 (UTC). As condições iniciais do Sol foram mantidas iguais a zero. As da Terra e da Lua foram usadas segundo os dados de Curtis (2005), com exceção de suas anomalias verdadeiras. A anomalia verdadeira da Terra foi calculada considerando-se o tempo passado desde o equinócio de outono do hemisfério sul, ocorrido em 20/03/2018. A anomalia verdadeira da Lua foi obtida com os dados de WolframAlpha (2018). Para cada ϕ apresentado foi realizada uma simulação com essas condições iniciais.

Tabela 18 – Condições iniciais para a simulação com o SGDC.

Elementos Orbitais às 06:43 (UTC) de 24/08/2018							
Corpo	a (km)	i $(graus)$	Ω $(graus)$	e	ω $(graus)$	f $(graus)$	n $(graus/s)$
Sol	0	0	0	0	0	0	0
Terra	149597870,6910	0	0	0	0	154,7433265	4,75321E-07
Lua	384400	5,1454	0	0,0549	0	179,6650	0,000152504
SGDC	42241,0801	0,0101	89	0,0002412	60	171,2009	0,0041666667

Os resultados das simulações do SGDC foram calculados em termos do raio orbital e da excentricidade para cada ϕ. As Figuras 29 e 30 mostram, respectivamente, esses resultados. Para $\phi = 0{,}04516\ m^2/kg$, o normal máximo do SGDC, as variações são imperceptíveis na escala da figura. Entretanto, os efeitos das perturbações orbitais são maiores, como esperado, para razões área-massa maiores.

O caso mais interessante é mostrado para a curva de razão área-massa de 50 m^2/kg. Note que há uma descontinuidade nas curvas correspondentes a esse valor de ϕ para as duas figuras, pouco antes de 100 dias, período em que a excentricidade, segundo a Figura 30, atinge valor 1, o que levaria a uma órbita parabólica.

Entretanto, antes de $e = 1$ acontecer, outro fenômeno pode ser observado. Com 55,73 dias de integração, o satélite atinge um raio orbital de 6564,9 km. Isso equivale a uma altitude de 186,9 km, ficando abaixo dos 200 km que definem a órbita LEO, provocando a reentrada do satélite e sua consequente queda. Isso sugere que seria possível retirar de órbita um satélite como o SGDC com as perturbações modeladas, se houvesse disponível um dispositivo que elevasse sua razão área-massa para 50 m^2/kg.

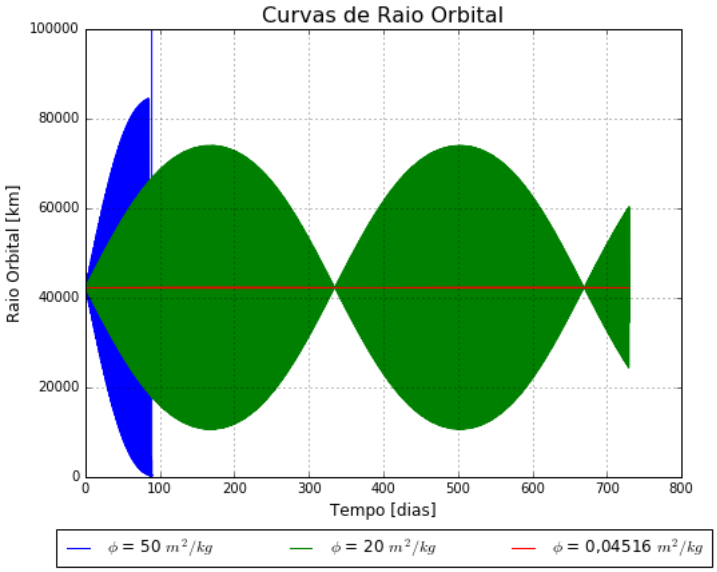

Figura 29 – Evolução do raio orbital do SGDC para diferentes ϕ.

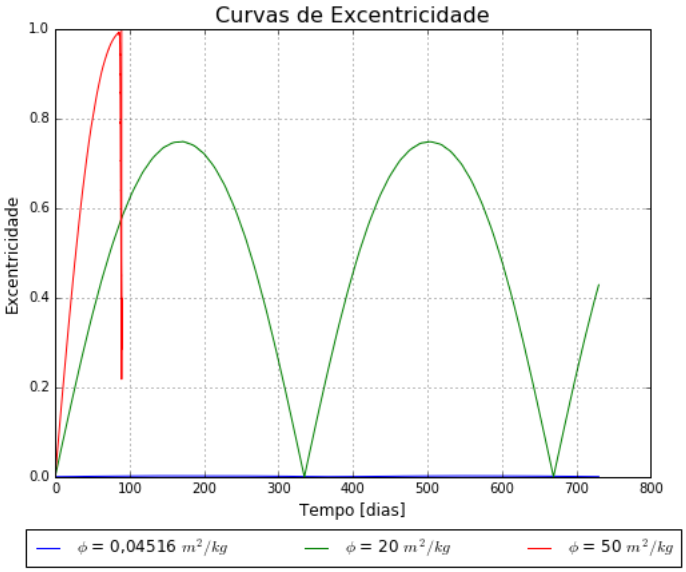

Figura 30 – Evolução da excentricidade do SGDC para diferentes ϕ.

6 Conclusões

Foi construído com sucesso um propagador orbital, na linguagem de programação C++ (referido neste trabalho como POC), capaz de simular, considerando o problema de quatro corpos, a órbita de um detrito ao redor da Terra sob perturbações luni-solares e de pressão de radiação solar. O simulador desenvolvido teve como referência o propagador orbital de Celestino (2005), escrito em Fortran (referido neste trabalho como POF). Este foi utilizado como referência na programação das entradas de simulação e para validar os resultados gerados pelo novo programa.

A compreensão do POF foi a primeira tarefa antes de iniciar o desenvolvimento numérico do POC. Isso foi essencial para compreender as relações entre cada etapa do programa, das variáveis e métodos utilizados. O funcionamento do POF foi demonstrado em vários diagramas neste trabalho.

Para desenvolver o POC, foram necessários métodos e bibliotecas para manipular matrizes em C++. Para isso, foi utilizada a biblioteca Eigen. Com ela, a programação das entradas da simulação, etapa em que o Eigen foi presente, levou menos tempo e ficou mais elegante que o código em C++ comum. Pode-se perguntar: e o porquê de a elegância de um código-fonte ser importante? Para facilitar a interpretação deste código, tanto pelo desenvolvedor quanto por futuros usuários, pois um código elegante pode ser compreendido, também, como organizado. A biblioteca Eigen, além disso, mostrou-se funcional, sem prejudicar o desempenho das simulações.

As entradas do POC foram construídas com base nas entradas do POF. Isso foi feito de forma a usar os mesmos formatos de equações de posição e velocidade, calculadas a partir dos elementos orbitais, nos dois simuladores, garantindo a construção de vetores de estado inicial iguais entre POC e POF. Isso foi importante porque o integrador numérico utilizado em POC não pôde ser o mesmo de POF, pois os simuladores operam de maneiras diferentes. Com um mesmo vetor de estados inicial nos dois, a validação da simulação, com relação aos resultados finais, foi mais simples de ser feita.

Desenvolver POC na linguagem de programação C++ tornou alguns passos do trabalho complicados, como a integração numérica comentada. Foi necessário encontrar uma biblioteca para o POC poder realizar a integração. A biblioteca adotada foi o Odeint e o integrador numérico o integrate_const, servindo ao fim desejado.

As equações luni-solares foram implementadas no POC e validadas comparando seus resultados com resultados de simulação com o POF. Essa parte do trabalho levou mais tempo que as demais, pois foi necessário implementar as equações dinâmicas e validá-las, ambas tarefas que exigiram empenho. Uma vez validado, a perturbação por pressão de

radiação solar foi implementada e, depois, validada com os resultados do POF.

A validação de resultados entre os simuladores foi feita via correlação de Pearson. Os resultados saídos dos simuladores têm formato matricial, em que cada coluna representa uma componente do vetor posição ou velocidade de um corpo e cada linha é o valor da componente calculado no tempo correspondente à linha. As correlações foram calculadas entre as colunas correspondentes nas duas matrizes. Para o passo de integração de 0,0001 dia, *stepper* Fehlberg 78 e integrador numérico integrate_const, detrito com razão área-massa de 50 m^2/kg e perturbações luni-solares e de PRS atuando, o menor valor de correlação foi de aproximadamente 0,962335461, todos os outros ficaram acima, atestando POC com os resultados de POF.

O propagador orbital POC foi aplicado para simular órbitas teste em diversas condições, variando elementos orbitais e a razão área-massa do detrito simulado. Foram obtidos resultados compatíveis com a literatura, observando que todos os elementos orbitais foram afetados pelas perturbações consideradas.

O POC foi utilizado também para um caso real: estudar o comportamento do SGDC sob diferentes valores de razão área-massa: a atual do satélite e duas supostas, caso fosse possível acoplar ao SGDC um dispositivo que aumentasse ϕ, como uma vela solar. Foi observado que para cerca de 55 dias com $\phi = 50\ m^2/kg$ seria possível retirar o satélite de órbita, fazendo-o reentrar na Terra. Para tanto, apenas as perturbações implementadas neste trabalho foram suficientes.

Trabalhos futuros incluem a possibilidade de transformar o POC num programa orientado a objetos, para melhorar a modelagem e a estrutura do programa. Podem ser, também, desenvolvidas novas funções equações dinâmicas e de perturbação, que podem ser inseridas no programa existente para gerar outros resultados. Espera-se que a ferramenta desenvolvida sirva a pesquisadores, estudantes, entusiastas, engenheiros e professores.

Referências

AHNERT, K.; MULANSKY, M. Controlled stepper. In: . Disponível em: <<https://www.boost.org/doc/libs/1_67_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/concepts/controlled_stepper.html>>. Acesso em 26 de julho de 2018: Boost.org, 2018. 50

AHNERT, K.; MULANSKY, M. Integrate functions. In: . Disponível em: <<https://www.boost.org/doc/libs/1_67_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/odeint_in_detail/integrate_functions.html>>. Acesso em 26 de julho de 2018: Boost.org, 2018. 50, 51, 52

AHNERT, K.; MULANSKY, M. Overview. In: . Disponível em: <<https://www.boost.org/doc/libs/1_67_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/getting_started/overview.html>>. Acesso em 26 de julho de 2018: Boost.org, 2018. 50, 53

AHNERT, K.; MULANSKY, M. Short example. In: . Disponível em: <<https://www.boost.org/doc/libs/1_67_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/getting_started/short_example.html>>. Acesso em 30 de julho de 2018: Boost.org, 2018. 55

AHNERT, K.; MULANSKY, M. Stepper. In: . Disponível em: <<https://www.boost.org/doc/libs/1_67_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/concepts/stepper.html>>. Acesso em 26 de julho de 2018: Boost.org, 2018. 50

AHNERT, K.; MULANSKY, M. Steppers. In: . Disponível em: <<https://www.boost.org/doc/libs/1_55_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/odeint_in_detail/steppers.html>>. Acesso em 30 de julho de 2018: Boost.org, 2018. 53, 66

AHNERT, K.; MULANSKY, M. System. In: . Disponível em: <<https://www.boost.org/doc/libs/1_55_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/concepts/system.html>>. Acesso em 30 de julho de 2018: Boost.org, 2018. 53

ALBATROSS. History of c++. In: . Disponível em: <<http://www.cplusplus.com/info/history/>. Acesso em 24 de junho de 2017: Cplusplus, 2016. 37

ALBATROSS. A brief description. In: . Disponível em: <<http://www.cplusplus.com/info/description/>. Acesso em 24 de junho de 2017: Cplusplus, 2017. 37

ASAITHAMBI, S. Why, how and when to scale your features. In: . Disponível em: <<https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e>. Acesso em 24 de junho de 2017: Medium, 2017. 47

ASANO, C. H.; COLLI, E. *Cálculo Numérico — Fundamentos e Aplicações*. São Paulo: Departamento de Matemática Aplicada – IME - USP, 2009. 30, 31

BRITO, T. P.; CELESTINO, C. C.; MORAES, R. V. A brief scenario about the "space pollution"around the earth. *Journal of Physics: Conference Series*, v. 465, n. 1, 2013. 16

BRYANT, C. Correlation. In: . Disponível em: <<https://pythonfordatascience.org/correlation-python/>>. Acesso em 28 de julho de 2018: Python for Data Science, 2018. 63

CELESTINO, C. C. Estudo da dinâmica de pequenos detritos espaciais e meteoroides. In: . [S.l.]: Instituto Nacional de Pesquisas Espaciais, 2005. 7, 8, 16, 17, 18, 19, 20, 29, 32, 33, 34, 35, 44, 46, 49, 53, 55, 60, 65, 67, 79

COLOMBO, C.; LUCKING, C.; MCINNES, C. R. Dynamics of high area-to-mass ratio spacecraft under the influence of j2 and solar radiation pressure. *62nd International Astronautical Congress*, 2011. 16

COLOMBO, C.; MCINNES, C. R. Orbital dynamics of "smart dust"devices with solar radiation. *Journal of Journal of Guidance, Control and Dynamics*, 2011. 16

CURTIS, H. Orbital mechanics for engineering students. In: . [S.l.]: Amsterdam: Elsevier Butterworth Heinemann, 2005. 16, 18, 20, 21, 23, 24, 25, 26, 27, 28, 29, 46, 77, 84, 86

DALININA, R. Introduction to correlation. In: . Disponível em: <<https://www.datascience.com/blog/introduction-to-correlation-learn-data-science-tutorials>>. Acesso em 28 de julho de 2018: Oracle + DataScience.com, 2017. 63

DAWES, B.; ABRAHAMS, D. Boost background information. In: . Disponível em: <<https://www.boost.org/users/>>. Acesso em 29 de julho de 2018: Boost.org, 2018. 50

DAWES, B.; ABRAHAMS, D. Getting started on unix variants. In: . Disponível em: <<https://www.boost.org/doc/libs/1_67_0/more/getting_started/unix-variants.html#header-only-libraries>>. Acesso em 29 de julho de 2018: Boost.org, 2018. 50

DAWES, B.; ABRAHAMS, D. Welcome to boost.org! In: . Disponível em: <<https://www.boost.org/>>. Acesso em 26 de julho de 2018: Boost.org, 2018. 50

FIESELER, P. D. A method for solar sailing in a low earth orbit. In: . [S.l.]: Jet Propulsion Laboratory, 1998. 17, 18, 30, 60

GAELZER, R. Introdução ao fortran 90/95. In: . Disponível em: <<https://ufsj.edu.br/portal2-repositorio/File/demat/PASTA-PROF/jorge/Fortran.pdf>. Acesso em 29 de julho de 2018: UFSJ, 2010. 36, 37

HEGGIE, D. C. The classical gravitational n-body problem. In: . [S.l.]: University of Edinburgh, 2005. 29

JACOB, B.; GUENNEBAUD, G. Main page. In: . Disponível em: <<http://eigen.tuxfamily.org/index.php?title=Main_Page>. Acesso em 29 de julho de 2018: Eigen, 2018. 39

KELSO, T. S. Norad two-line element sets current data. In: . Disponível em: <<https://www.celestrak.com/NORAD/elements/>. Acesso em 24 de agosto de 2018: CelesTrak, 2018. 77

MORAES, R. V. de. Combined solar radiation pressure and drag effects on the orbits of artificial satellites. *Celestial Mechanics*, v. 25, 1981. 17

MÖSLI, B. A comparison of c++, fortran 90 and oberon-2 for scientific programming. In: HUBER-WÄSCHLE, F.; SCHAUER, H.; WIDMAYER, P. (Ed.). *GISI 95*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. p. 740–748. 37

PARDAL, P. C. P. M.; MORAES, R. V. de; KUGA, H. K. Orbit determination modeling analysis by gps including perturbations due to geopotential coefficients of high degree and order, solar radiation pressure and luni-solar attraction. *Aerosp.Technol. Manag*, v. 1, 2011. 17, 29, 30, 76

RIEBEEK, H. Catalog of earth satellite orbits. In: . Disponível em: <<https://earthobservatory.nasa.gov/Features/OrbitsCatalog/printall.php>>. Acesso em 28 de junho de 2018: National Aeronautics and Space Administration, 2009. 26, 29, 32, 46

SHAMSUL, M. et al. A study of perturbation effect on satellite orbit using cowell's method. In: . [S.l.]: University Science Malaysia, 2017. 29

VALK, S.; LEMAITRE, L. A.; ANSELMO. Analytical and semi-analytical investigations of geosynchronous space debris with high area-to-mass ratios. In: . [S.l.]: Advances in Space Research, 2007. 17

VISIONA. Sistemas espaciais - o sgdc. In: . Disponível em: <<http://www.visionaespacial.com.br/sgdc>. Acesso em 24 de agosto de 2018: Visiona Tecnologia Espacial, 2017. 76

WOLFRAMALPHA. Moon true anomaly. In: . Disponível em: <<http://m.wolframalpha.com/input/;jsessionid=5776BFAA4468CF543E9AD982E5F18048?i=Moon+true+anomaly>. Acesso em 24 de agosto de 2018: Wolfram Alpha, 2018. 77

ZIPFEL, P. H. *Modeling and Simulation of Aerospace Vehicle Dynamics (Aiaa Education)*. 2. ed. [S.l.]: AIAA (American Institute of Aeronautics and Astronautics, 2007. 30

APÊNDICE A – Trabalhando com Elementos Orbitais

A.1 Cálculo dos Elementos Orbitais a Partir da Posição e Velocidade

Para avaliar o comportamento temporal de um corpo em órbita, uma abordagem interessante é analisar a variação dos elementos orbitais. É possível estudar como cada elemento da órbita se comporta durante a integração, observar o que as perturbações provocam e quais elementos da órbita são afetados. Os próximos parágrafos descrevem como obter os elementos orbitais a partir da posição $\mathbf{r} = (x, y, z)$ e velocidade $\mathbf{v} = (v_X, v_Y, v_Z)$ de um corpo (CURTIS, 2005).

Para calcular os elementos orbitais, primeiro deve-se obter a magnitude dos vetores de posição e velocidade. O módulo da posição orbital pode ser calculado pela Equação A.1. A variável $\mathbf{r}$ é o vetor da posição orbital e x, y e z são as componentes desse vetor em suas respectivas direções, medidas com relação a um referencial inercial.

$$r = \sqrt{\mathbf{r} \cdot \mathbf{r}} = \sqrt{x^2 + y^2 + z^2} \tag{A.1}$$

O módulo da velocidade orbital pode ser calculado pela Equação A.2. Nesta equação, $\mathbf{v}$ é o vetor da velocidade orbital, v_i $(i = X, Y, Z)$ são as componentes desse vetor nos eixos do sistema de coordenadas inercial $OXYZ$.

$$v = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{v_X^2 + v_Y^2 + v_Z^2} \tag{A.2}$$

Deve ser utilizada, então, a Equação A.3 para calcular o módulo da velocidade radial v_r na órbita, em função do módulo da posição orbital r, de seu vetor $\mathbf{r}$ e do vetor $\mathbf{v}$.

$$v_r = \frac{\mathbf{r} \cdot \mathbf{v}}{r} \tag{A.3}$$

Os vetores de posição e velocidade orbital são usados para calcular o vetor do momento angular específico da órbita $\mathbf{h}$, conforme a Equação A.4. Nela, h_i $(i = X, Y, Z)$ são as componentes desse vetor na direção dos versores $[\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}]$, respectivamente.

$$\mathbf{h} = [h_X\hat{\mathbf{i}} + h_Y\hat{\mathbf{j}} + h_Z\hat{\mathbf{k}}] = \mathbf{r} \times \mathbf{v} \tag{A.4}$$

O módulo do vetor de momento angular específico pode ser calculado com a Equação A.5.

$$h = \sqrt{\mathbf{h} \cdot \mathbf{h}} \tag{A.5}$$

Assim, a inclinação i da órbita pode ser calculada com a Equação A.6.

$$i = \cos^{-1}\left(\frac{h_Z}{h}\right) \tag{A.6}$$

Para determinar os argumentos do nodo ascendente e do perigeu, é necessário calcular o vetor da linha dos nodos $\mathbf{N}$. Ele é definido pela operação mostrada na Equação A.7, na qual $\hat{\mathbf{k}}$ é o versor que aponta na direção do eixo Z, $\mathbf{h}$ é o vetor de momento angular específico e N_i $(i = X,\ Y,\ Z)$ são as componentes desse vetor.

$$\mathbf{N} = [N_X\hat{\mathbf{i}} + N_Y\hat{\mathbf{j}} + N_Z\hat{\mathbf{k}}] = \hat{\mathbf{k}} \times \mathbf{h} \tag{A.7}$$

O módulo de $\mathbf{N}$ é dado pela Equação A.8.

$$N = \sqrt{\mathbf{N} \cdot \mathbf{N}} \tag{A.8}$$

Então, calcula-se o argumento do nodo ascendente, ou ascensão reta do nodo ascendente, Ω, pela Equação A.9.

$$\Omega = \cos^{-1}\left(\frac{N_X}{N}\right) \tag{A.9}$$

Em alguns elementos orbitais pode haver um problema de valor numérico nos resultados de alguns cossenos, causado por um problema de quadrantes. No caso de Ω, isso pode acontecer. Caso $N_Y < 0$, fazer o ajuste $\Omega \leftarrow 2\pi - \Omega$ (a seta $\leftarrow$ representa a operação "atribuir", ou "*assign*", em inglês, comumente usada em programação, e que instrui: atribuir à variável à qual a seta aponta o valor da operação realizada na raiz da seta).

Outra grandeza é necessária para os últimos elementos orbitais, o vetor excentricidade. Ele pode ser calculado pela Equação A.10.

$$\mathbf{e} = \frac{1}{\mu}\left[\mathbf{v} \times \mathbf{h} - \mu\frac{\mathbf{r}}{r}\right] \tag{A.10}$$

O módulo do vetor excentricidade pode ser calculado com a Equação A.11.

$$e = \sqrt{\mathbf{e} \cdot \mathbf{e}} \tag{A.11}$$

A Equação A.12 calcula o argumento da periapse ω.

$$\omega = \cos^{-1}\left(\frac{\mathbf{N}\cdot\mathbf{e}}{Ne}\right) \tag{A.12}$$

Análogo ao que foi feito com Ω, se $e_Z < 0$, fazer $\omega \leftarrow 2\pi - \omega$.

Por fim, a anomalia verdadeira pode ser calculada pela Equação A.13. Com a anomalia verdadeira, são calculados os 6 elementos orbitais a partir da posição e da velocidade.

$$f = \cos^{-1}\frac{\mathbf{e}\cdot\mathbf{r}}{er} \tag{A.13}$$

O valor de f deve ser corrigido também, assim como Ω e ω. Se $v_r < 0$, fazer $f \leftarrow 2\pi - f$.

A.2 Cálculo da Posição e Velocidade a Partir dos Elementos Orbitais

É possível, segundo Curtis (2005), calcular as componentes da posição e da velocidade a partir dos elementos orbitais. As Equações A.14 e A.15 calculam a posição orbital no referencial da órbita, em função do semieixo maior a, da anomalia excêntrica E e da excentricidade e. Por ser calculada no referencial da órbita, o valor da posição orbital não possui componente diferente de zero em z (Equação A.16).

$$x = a(\cos E - e) \tag{A.14}$$

$$y = a\sqrt{1-e^2}\,\mathrm{sen}E \tag{A.15}$$

$$z = 0 \tag{A.16}$$

As Equações A.17 e A.18 calculam a velocidade orbital no referencial da órbita, em função do semieixo maior a, da anomalia excêntrica E e da excentricidade e. Neste caso, também por ser calculada no referencial da órbita, a velocidade orbital não possui componente em z que seja diferente de zero (Equação A.19).

$$\dot{x} = -\frac{na^2}{r}\,\mathrm{sen}E \tag{A.17}$$

$$\dot{y} = \frac{na^2}{r}\sqrt{1-e^2}\cos E \tag{A.18}$$

$$\dot{z} = 0 \tag{A.19}$$

Os vetores de posição e velocidade formados por essas componentes são

$$\mathbf{r} = [x\hat{\mathbf{i}}\ y\hat{\mathbf{j}}\ z\hat{\mathbf{k}}]^T$$

$$\mathbf{v} = [\dot{x}\hat{\mathbf{i}}\ \dot{y}\hat{\mathbf{j}}\ \dot{z}\hat{\mathbf{k}}]^T$$