MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

ESTUDO DA PERTURBAÇÃO DA ÓRBITA DE SATÉLITES ARTIFICIAIS DEVIDO À AÇÃO DA RADIAÇÃO SOLAR

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA
(PIBIC/INPE/CNPq)

Anderson Bartholomeu de Oliveira (UBC, Bolsista PIBIC/CNPq)
E-mail: anderson-azz@hotmail.com

Hans-Ulrich Pilchowski (INPE – ETE/DMC, Orientador)
E-mail: hans.pilchowski@inpe.br

Julho de 2018

Dedico este trabalho primeiramente a Deus, ao meu pai Thome A. Oliveira, minha mãe Idalina B. Oliveira, aos meus amigos e meu coordenador que sempre esteve me apoiando.

AGRADECIMENTOS

Agradeço primeiramente a Deus e aos pais por
Ao meu coordenador Dr. Hans-Ulrich Pilchowski por sua orientação seu grande desprendimento em ajudar e amizade sincera.
Aos meus amigos Erick e Rodolfo pelo incentivo e grande ajuda. Agradeço ao INPE E CNPq que me proporcionaram realizar esse trabalho

RESUMO

Foi efetuado um estudo teórico, com a finalidade de modelar a força perturbadora, devida à pressão de radiação solar, que age sobre os satélites artificiais terrestres. Assim, baseado em códigos específicos, para a busca de componentes periódicos e lineares, desenvolveu-se um algoritmo computacional que fornece a perturbação de órbitas de satélites artificiais terrestres, devido à ação da radiação solar direta e indireta. A radiação solar indireta, por sua vez é composta da radiação solar refletida pela Terra, isto é, o albedo, da re-irradiação, ou seja, a radiação absorvida e reemitida pela Terra. Onde, para os dois casos, a atmosfera é considerada como um meio ótico refrativo. Porém, antes de estabelecer o efeito, dessa perturbação, na órbita de um satélite artificial, deve considera-se o geopotencial, aqui considerado até o nível J2. Assim, determinou-se órbita de um satélite artificial e sua propagação ao longo do tempo, considerando o geopotencial e a perturbação devida à radiação solar, a qual é fornecida em termos dos elementos orbitais Keplerianos, ou seja, o algoritmo fornece a perturbação em cada elemento orbital individual e simultaneamente. Finalmente, o algoritmo fornece a órbita simulada livre de perturbações externas e com a perturbação devida à radiação solar de forma automaticamente. O algoritmo está construído e está na primeira etapa onde só considera se o satélite sofrendo radiação direta na forma de sub-rotina, para que possa ser inserida em algoritmos mais abrangentes, de forma que seja possível seus resultados serem somados a outras perturbações orbitais e utilizados na correção orbital, sempre que for necessária.

Palavras-chave: Determinação da orbita de satélites, perturbações na orbita, geopotencial , radiação solar direta, albedo e re-irradiação.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE SIMBOLOS E ABREVIATURAS

a – Semi-Eixo maior

h – Momento angular

i – Inclinação

e – Excentricidade

ω – Argumento do perigeu

θ – Anomalia verdadeira

Ω – Ascensão da reta

PSR – Pressão da radiação solar

μ – Constante gravitacional

c – Velocidade da luz

S - Intensidade de radiação na orbita da terra

φ – coeficiente de sombra

A_s – Área de absorção

F – Força perturbadora

1 - INTRODUÇÃO

O posicionamento de satélites artificiais é efetuado através de pontos de referência no espaço, cuja posição e velocidade são conhecidas ou encontradas através de métodos da mecânica celeste. É necessário monitorar os satélites artificiais constantemente, desde seu lançamento até o término da vida sua útil. Através do monitoramento constante dá para se verificar se o equipamento está em seu funcionando adequadamente, ou não, e determinar sua órbita está correta. Para não correr o risco de perder a comunicação com o satélite, e por assim o próprio satélite, faz-se necessário conhecer-se seus elementos orbitais e ser capaz de prever sua órbita com precisão. Considerando, inicialmente, que a órbita seja constante, em relação aos semieixos e à excentricidade em um plano fixo, as forças perturbadoras influem no movimento orbital, de modo que, podem deformar ou mudar a trajetória do satélite. Assim, a pressão de radiação solar, entre outras, exerce efeitos perturbadores sobre a órbita de um satélite terrestre. Assim, o presente trabalho tem seu processo dividido por duas partes:

- A primeira parte consistiu no estudo de técnicas e métodos de determinação de orbitas de satélites artificiais. Ou seja, o estudo do problema direto, que consiste em: conhecendo-se os elementos keplerianos, utilizando-se a transformação de coordenadas orbitais em cartesianas e as equações de posicionamento e velocidade, o que permite obterem-se os vetores deslocamento. Subsequentemente, é efetuado o estudo do processo inverso que utiliza os dados obtidos no posicionamento direto e faz-se o caminho oposto para obterem-se os elementos keplerianos. E finalmente é efetuado o estudo das forças perturbadoras, o método das três posições e as funções temporais.

- A segunda parte consistiu em, por meio do conhecimento adquirido, por meio do estudo das técnicas e métodos, do item anterior, criar e desenvolver algoritmos que possibilitam a obtenção e das órbitas de satélites artificiais, com e sem as forças perturbadoras consideradas, as quis foram ser analisadas e comparadas, com outras conhecidas.

Foi utilizada, a linguagem python, por ser de uso livre, e apresentar ótimos resultados no tratamento e na modelagem de equações, sendo uma linguagem rápida e robusta para a manipulação de dados massivos.

2 - OBJETIVOS

2.1 - OBJETIVO GERAL

Objetivo deste trabalho foi desenvolver um algoritmo computacional que forneça a perturbação da órbita de satélites artificiais terrestres devido à ação da radiação solar direta, indireta (albedo) e re-irradiação da Terra. Para poder estabelecer-se a perturbação na órbita, de satélites artificiais, devido à radiação solar. Faz-se necessário, primeiramente, considerando o geopotencial, pelo menos até o nível J2, determinar a órbita desses satélites e determinar sua propagação ao longo do tempo. Assim, o desenvolvimento do algoritmo computacional que fornece a perturbação da órbita de um satélite, deve ser efetuado em termos dos elementos orbitais keplerianos dessa, ou seja, este deve fornecer a perturbação em cada elemento orbital individual e simultaneamente. O algoritmo fornecer essas perturbações automaticamente podendo ser colocado em forma de rotina, isto é, rotina que possa ser inserida em programas computacionais mais abrangentes, para que os resultados possam ser somados a outras perturbações orbitais e utilizados na correção orbital sempre que for necessária.

2.2 - OBJETIVOS ESPECIFICOS

Este projeto de pesquisa tem como objetivo desenvolver um algoritmo computacional que forneça dados da perturbação orbital de satélites artificiais terrestres, devida à ação da radiação solar direta. Assim, visou-se alcançar o objetivo da obtenção do algoritmo que possa ser inserido em programas computacionais mais abrangentes, destinados ao controle de órbitas.

3 - FUNDAMENTAÇÃO TEÓRICA

Conhecer as forças que atuam sobre um satélite artificial, queS mantém e perturbam um satélite em órbita, partindo do pressuposto que o leitor tenha conhecimentos básicos de mecânica celeste, Os principais métodos para o cálculo da determinação de órbitas perturbadas devido a radiação solar, serão:

3.1 - FLUXO DINÂMICO

Este é o fluxo de energia mediada pelo tempo e área que percorrida, transportada por fótons através de uma superfície normal ao sentido da radiação é a pressão de radiação solar p_{SR} tendo a fórmula descrita:

$$p_{SR} = \frac{S\ (intensidade\ de\ radiação\ na\ orbita\ da\ terra)}{c(velocidade\ da\ luz\ \approx 2.998\ \times 10^{-8}\ m/s)}$$

Supondo um modelo de simples, isto é, adotando o modelo de uma bala de canhão para a radiação solar e assumindo que o satélite seja uma esfera de raio R. Então, a força perturbadora F no satélite, devido à radiação p_{SR} é:

$$\boldsymbol{F} = -\varphi \frac{S}{c} C_r A_s \boldsymbol{\hat{u}}$$

Onde **û** é vetor unitário do satélite em direção ao sol, já sinal negativo demonstra que força da radiação solar é direcionada para longe do sol, φ é a função da sombra, que tem o valor 0 se o satélite estiver na sombra e também nessa primeira etapa considera penumbra como 0 e caso contrário 1 onde implica que satélite sofre ação direta, A_s determina a área de absorção do satélite posição do satélite em relação sol, onde nesse caso é total, onde a plena exposição à radiação solar, como demonstra a figura 1. Já o C_r é coeficiente de pressão de radiação que fica entre 1 e 2, sendo 1 quando o satélite se encontra absorvendo todo o momento de fóton, ou seja sofreno toda radiação solar

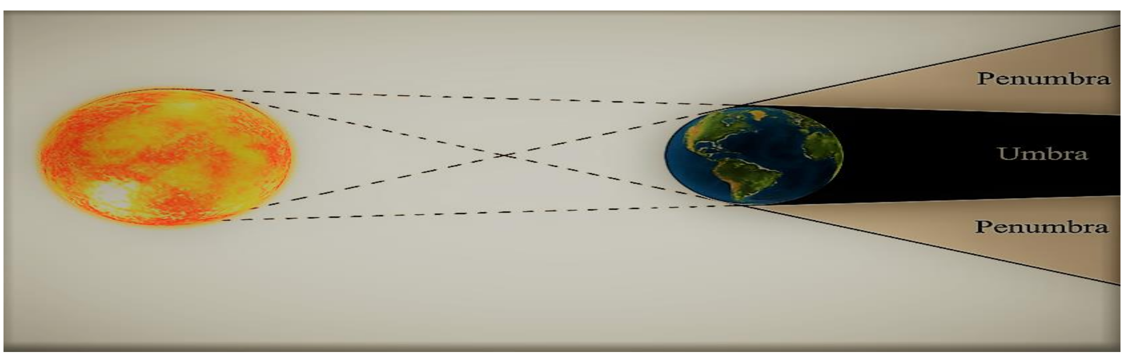

Figura 1 - Diagrama da Umbra e Penumbra
Fonte: o Autor

O Sol estando longe da terra, o ângulo entre a linha Terra – Sol e a satélite – Sol é menor que 0,02 graus mesmo para satélite GEO, será suficientemente preciso se assumir que vetor unitário seja em direção Sol – Terra e assim rastreia o movimento relativo entre eles e assim temos que aceleração perturbadora ***p*** devido à radiação solar ***F/m*** ou

$$\boldsymbol{p} = -p_{SR}\hat{\boldsymbol{u}}$$

E a magnitude da perturbação pode ser representada

$$p_{SR} = \varphi \frac{S}{c}\frac{C_r\, A_s}{m}$$

3.2 - VARIAÇÃO DE GAUSS

A perturbação da radiação solar afeta os elementos orbitais, ou seja, criando uma variação no tempo dos elementos que antes eram constantes, com a função temporal pode-se determinar a posição e a velocidade do satélite uma orbita perturbada, sabendo que a órbita tende a ficar degenerada devido a aceleração ao longo do tempo, e assim temos que:

$$\frac{dh}{dt} = -p_{SR}$$
$$\frac{de}{dt} = -p_{SR}\left\{\frac{h}{\mu} sin\theta u_r + \frac{1}{\mu h}\left[(h^2 + \mu r)\cos\theta + \mu e r\right]u_s\right\}$$
$$\frac{d\theta}{dt} = \frac{h}{r^2} - \frac{p_{SR}}{eh}\left[\frac{h^2}{\mu} cos\theta u_r - \left(r + \frac{h^2}{\mu}\right) sin\theta u_s\right]$$

$$\frac{d\Omega}{dt} = -p_{SR}\frac{r}{h\sin i}\sin(\omega + \theta)u_w$$
$$\frac{di}{dt} = -p_{SR}\frac{r}{h} cos(\omega + \theta)u_w$$
$$\frac{d\omega}{dt} = -p_{SR}\left\{\frac{1}{eh}\left[\frac{h^2}{\mu} cos\theta u_r - \left(r + \frac{h^2}{\mu}\right) sin\theta u_s\right] - \frac{r\sin(\omega+\theta)}{h\tan i}u_w\right\}$$

Para integrar numericamente as equações acima, é necessário conhecer a variação temporal da obliquidade e longitude eclíptica solar. Nós também precisamos do histórico de tempo da distância da Terra ao Sol, a fim de calcular o equatorial geocêntrico vetor de posição do sol, a saber, juntamente com o vetor de posição geocêntrica do satélite nos permite determinar quando o satélite está na sombra da Terra, de acordo com o Almanaque

Astronômico (National Almanac Office, 2013), a energia solar aparente longitude eclíptica (em graus) é dada pela a forma que se encontra no mesmo.

4 - MATERIAIS E MÉTODOS

Para poder estabelecer-se a perturbação na órbita, de satélites artificiais, devido à radiação solar, faz-se necessário, primeiramente, considerando o geopotencial, pelo menos até o nível J2, determinar a órbita desses satélites e determinar sua propagação ao longo do tempo. Assim, o desenvolvimento do algoritmo computacional que fornece a perturbação da órbita de um satélite, deve ser efetuado em termos dos elementos orbitais keplerianos dessa, ou seja, este deve fornecer a perturbação em cada elemento orbital individual e simultaneamente. O algoritmo desenvolvido em python3.6 utilizando as bibliotecas de manipulação de dados para área cientifica sendo elas: Numpy, Matplotlib e Scipy. Assim foram criados os algoritmos que determinam a orbita de um satélite com ou sem a perturbação devido a radiação solar, e assim pode facilmente ser colocado em forma de rotina, isto é, rotina que possa ser inserida em programas computacionais mais abrangentes, para que os resultados possam ser somados a outras perturbações orbitais e utilizados na correção orbital sempre que for necessária.

5 - ANÁLISES E RESULTADOS

Os resultados obtidos foram comparados com dados encontrados por CURTIS (2014), assim foram possíveis validar os algoritmos de determinação de órbitas, onde foram comparados um por vez para uma validação mais precisas dos dados obtidos.

Os métodos direto e inverso comparado com os resultados já existentes, como os dados mostram resultados semelhantes podendo ser considerados validos como demonstra figura 2, onde a primeira parte são os dados ser analisados e a segunda são os dados obtidos.

```
Dados fornecidos:
Momento angular:   80000  Km^2/s
Excentricidade:   1.4
Inclinação:   0.523598775598  rad
Ascensão do Nodo Ascendente:   1.0471975512
grad
Argumento do Perigeu:   0.698131700798  rad
Anomalia Verdadeira:   0.523598775598  rad

---------------------------------------------------
Dados Calculados:
Posição:   [-4039.8959232   4814.56048018
3628.62470217]
Velocidade:   [-10.38598762  -4.77192164   1.
743875  ]

Process finished with exit code 0
```

Figura 2 - Método Direto
Fonte: o Autor.

Método inverso utiliza vetor de estado para se obter os elementos keplerianos o resultado desse algoritmo vide figura 3.

```
Dados fornecidos:
Posição:   [-6045 -3490  2500]
Velocidade:   [-3.457  6.618  2.533]
---------------------------------------------------
Dados Calculados:
Momento angular:   8393.5  Km^2/s
Excentricidade:   0.171212346284
Inclinação:   153.249228518  rad
Ascensão do Reta do Nodo Ascendente:   104.
720714666  rad
Argumento do Perigeu:   20.0683166506  rad
Anomalia Verdadeira:   28.4456283066  rad


Process finished with exit code 0
```

Figura 3 - Método Inverso
Fonte: o Autor.

O algoritmo do método das três posições figura 4 e propagação foram testados figura 5 e validados e se assemelham aos dados de referência CURTIS (2014):

```
⬆Dados fornecidos:
 Posição 1: [-294.32, 4265.1, 5986.7]
 Posição 2: [-1365.5, 3637.6, 6346.8]
 Posição 3: [-2940.3, 2473.7, 6555.8]

Resultados
Velocidade para a posição (dois): [-6.21740189 -4.01216524  1.59898473]

Os elementos orbitais que representa a posição e a velocidade (dois)
Semi-eixo maior :   7813.0  Km
Excentricidade:   0.10010369281339042
Inclinação:   60.000470277369566  graus
Ascensão do nodo:   40.00144177286778  graus
Argumento do Perigeu:   30.074116831547773  graus
Anomalia verdadeira:   49.92565926551782  graus


Process finished with exit code 0

```

Figura 4 - Método das três posição
Fonte: o Autor.

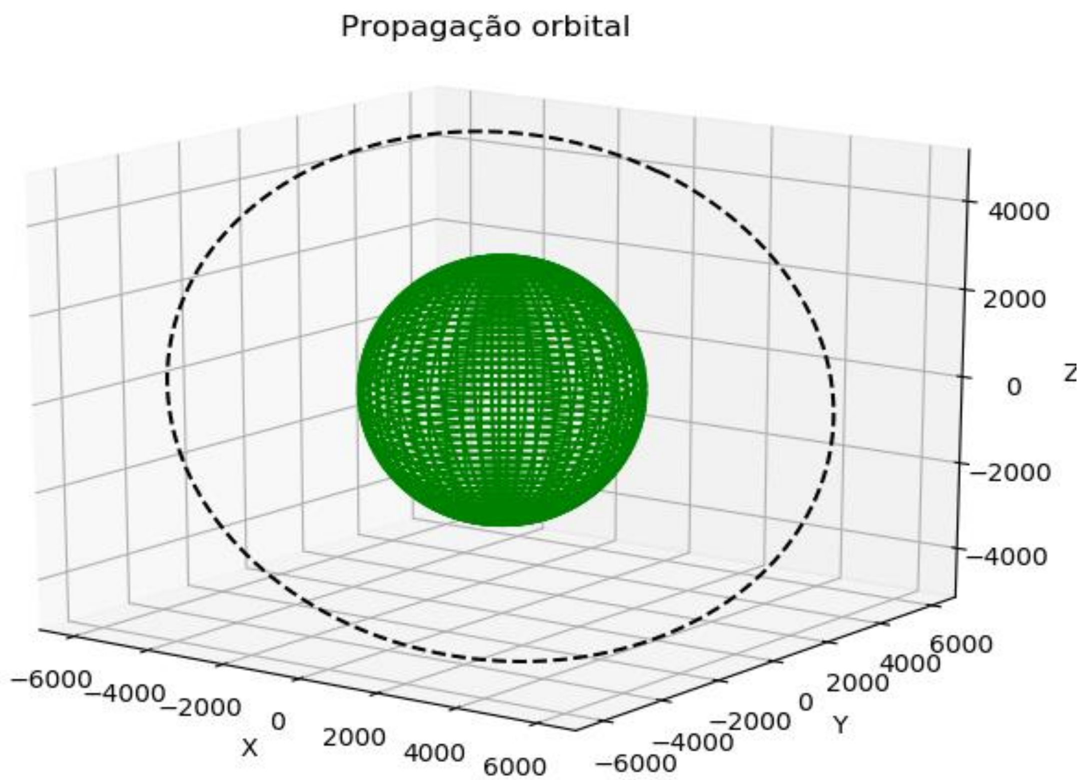

Figura 5 - Propagação Orbital
Fonte: o Autor.

Para simulação das perturbações devido à ação da radiação solar foram utilizados os algoritmos já citados, os resultados obtidos foram validados pela comparação com dados fornecidos por CURTIS (2014). Note-se que existem divergências entre os gráficos obtidos e os fornecidos por CURTIS (2014), devido a quantidade de casas decimais utilizadas. Entretanto os dados obtidos correspondem com os dados de validação, está sendo o utilizado o exemplo a seguir para obtenção dos dados:

Um satélite terrestre esférico tem uma relação área absorvente massa (As / m) de 2m² / kg. No tempo t0 (data juliana JD0 = 2.438.400,5) seus parâmetros orbitais são

Momento angular: 63383.4 km²/s
Excentricidade: 0.025422
Ascensão reta: 45.3812°
Inclinação:88.3924°
Argumento do perigeu:227.493°
Anomalia verdadeira:343.427°

O primeiro conjunto de gráficos, de dados, são da referência de CURTIS (2014), e o segundo conjunto devido ao autor.

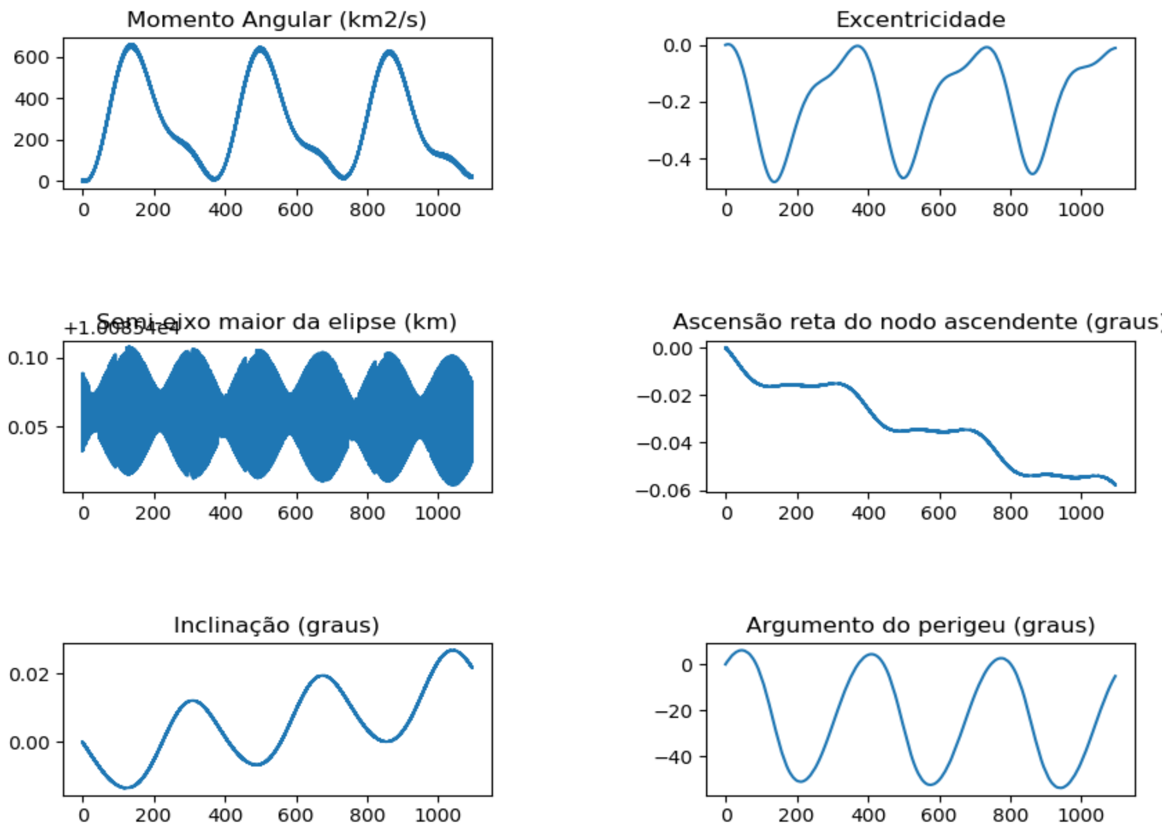

Figura 6 - Dados obtidos ação da radiação solar
Fonte: o Autor.

6 - CONCLUSÃO

Este trabalho apresenta o desenvolvimento do estudo de determinação de orbitas com a perturbação da radiação solar direta em sua fase um final de desenvolvimento, considerando os objetivos na qual se diz respeito aos estudos e as formulações de algoritmos usando funções de mecânica celeste, para o desenvolvimento e a compreensão dos métodos usados para determinar orbitas tanto com ou sem perturbações, em si foram alcançados.

Os últimos testes para otimizar o algoritmo tornando o mais eficiente e rápido serão implementadas na segunda parte onde terá o algoritmo final, com tudo pode se dizer que em seu estado atual ele atende os requisitos e com uma facilidade e implementação em outros algoritmos mais complexos devido a linguagem abordada e os métodos de formulação do código.

Este projeto ainda deixa lugar a uma possível continuação e melhoras, a fim de otimizar o algoritmo, podendo até ser desenvolvida uma interface gráfica mais amigável, com fornecimento de imagens mais detalhadas e opções de deixar ele automático ou manual dando uma maior liberdade e variedade de escolha para o usuário decidir melhor como utilizar o projeto.

REFERÊNCIA

PILCHOWSKI, H.U.; SILVA, W.C.C. & FERREIRA, L.D.D. **Introdução à mecânica celeste**. São José dos Campos, São Paulo, Brasil, 1981. (INPE-2126-RPE/350)

FERREIRA, L.D.D.; SILVA, W.C.C. & PILCHOWSKI, H.U. **Notas sobre sistemas de coordenadas e tempo**. São José dos Campos, São Paulo, Brasil, 1979. (INPE-1634-RPE/039)

CURTIS, Howard D. **Orbital mechanics for engineering students** 2. ed. Amsterdam, The Netherlands: Elsevier, 2009. 722 p. (Elsevier Aerospace Engineering Series) ISBN 978-0-1237-4778-5

NOAA/NASA/USAF, 1976. **U.S. Standard Atmosphere**, 1976. GPO.

BROUWER, D. & CLEMENCE, G.M. **Methods of celestial mechanics**. New York, N.Y., Academic, 1961.

ESCOBAL, P.R. **Methods of orbital determination**. New York, N.Y., John Wiley & Sons, 1965.

PRUSSING, J. E.; CONWAY, B. A. **Orbital Mechanics**. Oxford, University Press, 1993.

BOND, V. R.; ALLMAN, M. C. **Modern Astrodynamics: Fundamentals and Perturbation Methods**. Princeton University Press, 1996.

KAPLAN, M. H. **Modern spacecraft dynamic & control**. New York: John Wiley & Sons, 1976.

LARSON, W. J.; WERTZ, J. R. **Space mission analysis and design**. Torrance, California: Space Technology Series, 1992.

WIE, B. **Space vehicle dynamics and control**. Reston, Virginia: AIAA Education Series, 1998.

APÊNDICE

Este apêndice é a lista dos scripts do Python que foram implementados dos algoritmos necessário para Estudo da perturbação da órbita de satélites artificiais devido à ação da radiação solar

```
import numpy as np
######## POSICIONAMENTO DIRETO ##############
class direto(object):
    R: np.ndarray
    V: np.ndarray
    def __init__(self, elk, mi):
        self.elk = elk
        self.mi = mi

    def pos_dir(self):
        momA = self.elk[0]
        exce = self.elk[1]
        incl = self.elk[2]
        AscR = self.elk[3]
        ArgP = self.elk[4]
        AnoV = self.elk[5]

        """ posição """
        rx = pow(momA, 2)/self.mi*(1/(1+exce*np.cos(AnoV)))*np.cos(AnoV)
        ry = pow(momA, 2)/self.mi*(1/(1+exce*np.cos(AnoV)))*np.sin(AnoV)
        rz = pow(momA, 2)/self.mi*(1/(1+exce*np.cos(AnoV)))*0

        """ velocidade """
        vx = self.mi/momA*-np.sin(AnoV)
        vy = self.mi/momA*(exce + np.cos(AnoV))
        vz = self.mi/momA*0

        Qx_1 = np.array([[np.cos(ArgP), np.sin(ArgP), 0],
                         [-np.sin(ArgP), np.cos(ArgP), 0],
                         [0, 0, 1]])

        Qx_2 = np.array([[1, 0, 0],
                         [0, np.cos(incl), np.sin(incl)],
                         [0, -np.sin(incl), np.cos(incl)]])

        Qx_3 = np.array([[np.cos(AscR), np.sin(AscR), 0],
                         [-np.sin(AscR), np.cos(AscR), 0],
                         [0, 0, 1]])

        Qx1 = np.dot(Qx_2, Qx_3)
        Qx = np.dot(Qx_1, Qx1)
        Qx_T = np.matrix.transpose(Qx)
```

```python


###########           MÉTODO INVERSO ############
import numpy as np
import math as mth
#import pyexcel_ods3 as pods
from pos_direto import direto

class inverso(object):
    def __init__(self, pos, vel):
        self.pos = pos
        self.vel = vel
        self.mi = 398600

    def pos_vel(self):
        """ posição (r)"""
        r_e1 = np.asscalar(pow(self.pos[0], 2) + pow(self.pos[1], 2) + pow(self.pos[2], 2))
        self.r_e = np.sqrt(r_e1)

        """ velocidade (v)"""
        v_e1 = np.asscalar(pow(self.vel[0], 2) + pow(self.vel[1], 2) + pow(self.vel[2], 2))
        self.v_e = np.sqrt(v_e1)

        """ multiplicação vetorial de posição com a velocidade"""
        vr_0 = np.dot(self.vel, self.pos)
        self.vr = (vr_0 / self.r_e)

    def mom_ang(self):
        self.pos_vel()

        """ Magnetude do momento angular (h)"""
        self.h = np.cross(self.pos, self.vel)
        h_e0 = np.asscalar(pow(self.h[0], 2) + pow(self.h[1], 2) + pow(self.h[2], 2))
        return np.sqrt(h_e0)

    def incl(self):
        self.mom_ang()

        """ inclinação (i)"""
        i_1 = (self.h[2] / self.mom_ang())
        incl = np.arccos(i_1)
```

```
        '''
        if np.degrees(incl) > 90:
            print('\t\033[1;31mOrbita retrograda\033[m')
        '''
        """ multiplicação vetorial de K com h """
        k = np.array([0, 0, 1])
        self.n1 = np.cross(k, self.h)
        n1_e0 = np.asscalar(pow(self.n1[0], 2) + pow(self.
n1[1], 2) + pow(self.n1[2], 2))
        self.n1_e = np.sqrt(n1_e0)
        return incl

    def asc_ret(self):
        self.incl()

        """ ascenção da reta """
        return mth.acos((self.n1[0] / self.n1_e))

    def excen(self):
        self.asc_ret()

        """ excentricidade """
        self.e = (1 / self.mi * (((pow(self.v_e, 2) - self
.mi / self.r_e) * self.pos) - self.r_e * self.vr * self.
vel))
        e_e0 = np.asscalar(pow(self.e[0], 2) + pow(self.e[
1], 2) + pow(self.e[2], 2))
        return np.sqrt(e_e0)

    def arg_per(self):
        self.excen()

        """ argumento do perigeu """
        return mth.acos(np.dot(self.n1, self.e) / (self.
n1_e * self.excen()))

    def anom_verd(self):
        self.arg_per()

        """ anomalia Verdadeira """
        return mth.acos(np.dot(self.e, self.pos) / (self.
excen() * self.r_e))

    def sem_Ex_maior(self):
        """ Raio do Perigeu """
```

```
        rp = pow(self.mom_ang(), 2) / self.mi * (1 / (1 + self.excen() * np.cos(0)))
        """ Raio do Apogeu """
        ra = pow(self.mom_ang(), 2) / self.mi * (1 / (1 + self.excen() * np.cos(180)))
        """ Semi eixo maior """
        return 0.5 * (int(rp + ra))

    def anom_exce(self):
        """ Anomalia Excentrica """
        return 2 * mth.atan(np.tan(self.anom_verd() / 2) / np.sqrt((1 + self.excen()) / (1 - self.excen())))

    def anom_media(self):
        """ Anomalia Media """
        return self.anom_exce() - self.excen() * np.sin(self.anom_exce())

    def periodo(self):
        """ periodo """
        return 2 * np.pi / np.sqrt(self.mi) * pow(self.sem_Ex_maior(), 1.5)

    def mov_med(self):
        """ Movimento Médio """
        return 2 * np.pi / self.periodo()

    def anom_excI(self):
        """ Anomalia Excentrica Inicial """
        return mth.atan(np.sqrt((1 - self.excen()) / (1 + self.excen())) * (np.tan(self.anom_verd() / 2))) * 2

    def tempo_inic(self):
        """ Tempo Inicial """
        return (self.anom_excI() - self.excen() * np.sin(self.anom_excI())) / self.mov_med()


    def saida_inv(self):
        #direto(self.sem_Ex_maior(), self.excen(), self.incl(), self.arg_per(), self.asc_ret(), self.anom_media())
        print("\033[1;31mSemieixo Maior:\033[m ", "\033[1;30m", self.sem_Ex_maior(), "\033[m", "\033[1;31mKm\033[m")
```

```
        exce = self.pos_inv.excen()
        mov_med = self.pos_inv.mov_med()
        incl = self.pos_inv.incl()
        asc_ret = self.pos_inv.asc_ret()
        arg_per = self.pos_inv.arg_per()
        tempo_inic = self.pos_inv.tempo_inic()

        Tinicial = DJul(self.__ano, self.__mes, self.__dia, self.__hora, self.__min, self.__seg).TempI

        int_T = float(input('coloque o intervalo de tempo para a propagação orbital:'))

        while self.__seg < self.tempo:
            self.__seg += int_T

            data = DJul(self.__ano, self.__mes, self.__dia, self.__hora, self.__min, self.__seg).TempI
            delta_t = (data - Tinicial) * 86400
            self.ListT.append(["%.1f" % round(delta_t, 2)])

            temp_fin = tempo_inic + delta_t

            np2 = temp_fin / self.tempo
            t32 = (np2 - np.floor(np2)) * self.tempo

            """ Anomalia Média """
            anom_medF = mov_med * t32

            """ Anomalia Excentrica """
            erro = 1 * pow(10, -8)

            if anom_medF < mth.pi:
                E32 = anom_medF + exce / 2
            else:
                E32 = anom_medF - exce / 2

            ratio = 1

            while abs(ratio) > erro:
                ratio = (E32 - exce * np.sin(E32) - anom_medF) / (1 + exce * np.cos(E32))
                E32 = E32 - ratio

```

```
            """ Ascensão da Reta """
            asc1 = -1.5 * (np.sqrt(self.mi) * self.j2 * pow(self.R, 2) /
                           (((1 - pow(exce, 2)) ** 2) * pow(sem_eix, 3.5))) * np.cos(incl)

            g = np.degrees(asc1)
            asc_retF = np.radians(np.degrees(asc_ret) + g * delta_t)
            self.ListAscR.append(asc_retF)

            """ Argumento do Perigeu """
            arg1 = -1.5 * (np.sqrt(self.mi) *
                           self.j2 * pow(self.R, 2) /
                           ((1 - pow(exce, 2) ** 2) *
                            pow(sem_eix, 3.5))) * (2.5 * pow(np.sin(incl), 2) - 2)

            g2 = np.degrees(arg1)
            arg_perF = np.radians(np.degrees(arg_per) + g2 * delta_t)
            self.ListArgP.append(arg_perF)

            """ anomalia Verdadeira """
            anom_verdF = 2 * mth.atan(np.sqrt((1 + exce) / (1 - exce)) * np.tan(E32 / 2))
            self.ListAnoV.append(anom_verdF)

            """ posição """
            rx = pow(self.mom_ang, 2) / self.mi * (1 / (1 + exce * np.cos(anom_verdF))) * np.cos(anom_verdF)
            ry = pow(self.mom_ang, 2) / self.mi * (1 / (1 + exce * np.cos(anom_verdF))) * np.sin(anom_verdF)
            rz = pow(self.mom_ang, 2) / self.mi * (1 / (1 + exce * np.cos(anom_verdF))) * 0

            Npos = np.array([rx, ry, rz])

            """ velocidade """
            vx = self.mi / self.mom_ang * -np.sin(anom_verdF)
            vy = self.mi / self.mom_ang * (exce + np.cos(anom_verdF))
            vz = self.mi / self.mom_ang * 0

```

```
            Nvel = np.array([vx, vy, vz])

            Qx_1 = np.array([[np.cos(arg_perF), np.sin(arg_perF), 0],
                             [-np.sin(arg_perF), np.cos(arg_perF), 0],
                             [0, 0, 1]])

            Qx_2 = np.array([[1, 0, 0],
                             [0, np.cos(incl), np.sin(incl)],
                             [0, -np.sin(incl), np.cos(incl)]])

            Qx_3 = np.array([[np.cos(asc_retF), np.sin(asc_retF), 0],
                             [-np.sin(asc_retF), np.cos(asc_retF), 0],
                             [0, 0, 1]])

            Qx1 = np.dot(Qx_2, Qx_3)
            Qx = np.dot(Qx_1, Qx1)
            Qx_T = np.matrix.transpose(Qx)

            posF = np.dot(Qx_T, Npos)
            self.ListP.append(posF)

            posF_e = np.sqrt(posF[0]**2+posF[1]**2+posF[2]**2)
            self.ListPesc.append(posF_e)

    def saida_grafic(self):
        self.Inv_Direto()
        from gráficos import grafic
        grafic.grafic_propag(self.ListP, self.ListT)
```

```
def los(rsat, rsun):
    RE = 6378
    r_sat = np.sqrt(rsat[0] ** 2 + rsat[1] ** 2 + rsat[2] ** 2)
    r_sun = np.sqrt(rsun[0] ** 2 + rsun[1] ** 2 + rsun[2] ** 2)

    theta = np.arccos(np.radians(np.dot(rsat, rsun)/r_sat/r_sun))
    thetasat = np.arccos(np.radians(RE/r_sat))

    thetasun = np.arccos(np.radians(RE/r_sun))

    if thetasat + thetasun <= theta:
        ligth_switch = 0
    else:
        ligth_switch = 1

    return ligth_switch
```

```
    def ALS_solar_pos(self):

        # Mean anomaly(deg)
        M0 = 357.528 + 0.9856003 * self.n
        M = np.mod(M0, 360)

        # Mean longitude(deg)
        L0 = 280.460 + 0.98564736 * self.n
        L = np.mod(L0, 360)

        # Apparent ecliptic longitude(deg)
        lamda0 = L + 1.915 * np.sin(np.radians(M)) + 0.020 * np.sin(np.radians(2 * M))
        lamda = np.mod(lamda0, 360)

        # Obliquity of the ecliptic(deg)
        eps = 23.439 - 0.0000004 * self.n

        # Unit vector from earth to sun
        u = np.array([np.cos(np.radians(lamda)),
        np.sin(np.radians(lamda)) * np.cos(np.radians(eps)),
        np.sin(np.radians(lamda)) * np.sin(np.radians(eps))])

        # Distance from earth to sun(km)
        rS = (1.00014 - 0.01671 * np.cos(np.radians(M)) - 0.000140 * np.cos(np.radians(2 * M))) * self.AU
        # Geocentric position vector(km)
        return lamda, eps, rS * u
```

```
import numpy as np
from pos_inverso import inverso

class tree_pos(object):
    def __init__(self, mi, pos):
        self.mi = mi
        self.pos = pos
        self.tol = 1e-4

    def três_posições(self):
        Pos1 = self.pos[0]
        Pos2 = self.pos[1]
        Pos3 = self.pos[2]

        pos1 = np.sqrt(Pos1[0] ** 2 + Pos1[1] ** 2 + Pos1[2] ** 2)
        pos2 = np.sqrt(Pos2[0] ** 2 + Pos2[1] ** 2 + Pos2[2] ** 2)
        pos3 = np.sqrt(Pos3[0] ** 2 + Pos3[1] ** 2 + Pos3[2] ** 2)

        C12 = np.cross(Pos1, Pos2)
        C23 = np.cross(Pos2, Pos3)
        C31 = np.cross(Pos3, Pos1)

        c23 = np.sqrt(C23[0] ** 2 + C23[1] ** 2 + C23[2] ** 2)

        if abs(np.dot(Pos1, C23)/pos1/c23) > self.tol:
            print('\n  Os vetores posição não são coplanares.\n\n')

        else:
            N = pos1*C23 + pos2*C31 + pos3*C12
            n = np.sqrt(N[0] ** 2 + N[1] ** 2 + N[2] ** 2)

            D = C12 + C23 + C31
            d = np.sqrt(D[0] ** 2 + D[1] ** 2 + D[2] ** 2)

            S = Pos1 * (pos2 - pos3) + Pos2 * (pos3 - pos1) + Pos3 * (pos1 - pos2)

            Vel2 = np.sqrt(self.mi / n / d) * (np.cross(D, Pos2) / pos2 + S)

```

```

        def deriv(t, f):
            DJ = self.JD0 + int(t) / self.days
            Lam, ep, rs = atração_solar(DJ).ALS_solar_pos()

            lamda = np.radians(Lam)
            eps = np.radians(ep)

            h = f[0]
            e = f[1]
            i = f[2]
            RA = f[3]
            w = f[4]
            TA = f[5]
            phi = w + TA
            elk = np.array([h, e, i, RA, w, TA])

            posF, velF = direto(elk, self.mi).pos_dir()

            pos_e = np.sqrt(posF[0] ** 2 + posF[1] ** 2 + posF[2] ** 2)

            nu = los(posF, rs)

            pSR = nu*(self.S/self.c)*self.CR*self.As/self.m/1000

            sl = np.sin(lamda)
            se = np.sin(eps)
            sW = np.sin(RA)
            si = np.sin(i)
            su = np.sin(phi)
            sT = np.sin(TA)

            cl = np.cos(lamda)
            ce = np.cos(eps)
            cW = np.cos(RA)
            ci = np.cos(i)
            cu = np.cos(phi)
            cT = np.cos(TA)

            """ Perturbação da aceleração """
            ur = sl*ce*cW*ci*su + sl*ce*sW*cu - cl*sW*ci*su+ cl*cW*cu + sl*se*si*su
```

```

            us = sl*ce*cW*ci*cu - sl*ce*sW*su - cl*sW*ci*cu-cl*cW*su + sl*se*si*cu

            uw = - sl*ce*cW*si + cl*sW*si + sl*se*ci

            hdot = -pSR*pos_e*us

            edot = -pSR*(h/self.mi*sT*ur+1/self.mi/h*((h**2 + self.mi*pos_e)*cT + self.mi*e*pos_e)*us)

            RAdot = -pSR*pos_e/h/si*su*uw

            idot = -pSR*pos_e/h*cu*uw

            wdot = -pSR*(-1/e/h*(h**2/self.mi*cT*ur - (pos_e + h**2/self.mi)*sT*us)-pos_e*su/h/si*ci*uw)

            TAdot = h/pos_e**2 - pSR/e/h*(h**2/self.mi*cT*ur - (pos_e + h**2/self.mi)*sT*us)

            #self.lista.append((hdot**2/self.mi/(1-edot**2)))
            #self.ap = np.array([hdot**2/self.mi/(1-edot**2)])
            #print("lista",self.lista)
            #print("ap",self.ap)


            return [hdot, edot, idot, RAdot, wdot, TAdot]

        coe = Radau(deriv, [t0, tf], y0, max_step=nout, rtol=1.e-8, atol=1.e-8)

        h_f, e_f, i_f, RA_f, w_f, TA_f = coe.y
        t_f = coe.t
```