



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



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

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 órbita de satélites, perturbações na órbita, geopotencial, radiação solar direta, albedo e re-irradiação.



## SUMÁRIO

RESUMO .....	2
LISTA DE ILUSTRAÇÕES .....	6
LISTA DE SIMBOLOS E ABREVIATURAS .....	6
1 - INTRODUÇÃO .....	7
2 - OBJETIVOS .....	8
2.1 - OBJETIVO GERAL .....	8
2.2 - OBJETIVOS ESPECIFICOS .....	8
3 - FUNDAMENTAÇÃO TEÓRICA .....	8
3.1 - FLUXO DINÂMICO .....	9
3.2 - VARIAÇÃO DE GAUSS .....	10
4 - MATERIAIS E METODOS .....	11
5 - ANÁLISES E RESULTADOS .....	11
6 - CONCLUSÃO .....	15
REFERÊNCIA .....	16



## LISTA DE ILUSTRAÇÕES

Figura 1 - Diagrama da Umbra e Penumbra.....	9
Figura 2 - Método Direto.....	12
Figura 3 - Método Inverso .....	12
Figura 4 - Método das três posição.....	13
Figura 5 - Propagação Orbital .....	14
Figura 7 - Dados obtidos ação da radiação solar .....	15



## LISTA DE SIMBOLOS E ABREVIATURAS

$a$  – Semi-Eixo maior

$h$  – Momento angular

$i$  – Inclinação

$e$  – Excentricidade

$\omega$  – Argumento do perigeu

$\theta$  – Anomalia verdadeira

$\Omega$  – Ascensão da reta

PSR – Pressão da radiação solar

$\mu$  – Constante gravitacional

$c$  – Velocidade da luz

S - Intensidade de radiação na orbita da terra

$\varphi$  – 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 quais 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, que 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:

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

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:

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

Onde  $\hat{\mathbf{u}}$  é vetor unitário do satélite em direção ao sol, já sinal negativo demonstra que força da radiação solar é direcionada para longe do sol,  $\varphi$  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 sofrendo toda radiação solar

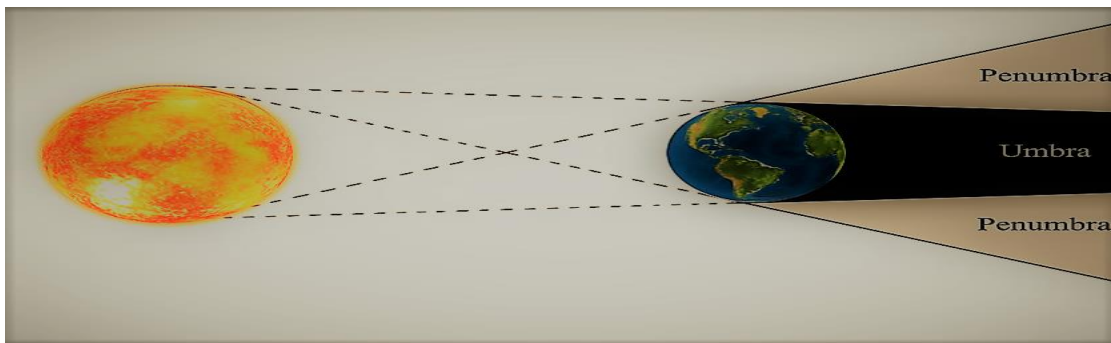


Figura 1 - Diagrama da Umbra e Penumbra  
Fonte: o Autor

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  $\mathbf{p}$  devido à radiação solar  $F/m$  ou

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

E a magnitude da perturbação pode ser representada

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

### 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:

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

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

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 científica 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.



```
9  ↑Dados fornecidos:
10 Momento angular:  80000  Km^2/s
11 Excentricidade:  1.4
12 Inclinação:  0.523598775598  rad
13 Ascensão do Nodo Ascendente:  1.0471975512
    grad
14 Argumento do Perigeu:  0.698131700798  rad
15 Anomalia Verdadeira:  0.523598775598  rad
16
17 -----
    ---
18 Dados Calculados:
19 Posição:  [-4039.8959232  4814.56048018
    3628.62470217]
20 Velocidade:  [-10.38598762  -4.77192164  1.
    743875  ]
21
22 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.

```
9  ↑Dados fornecidos:
10 Posição:  [-6045 -3490  2500]
11 Velocidade:  [-3.457  6.618  2.533]
12 -----
    ---
13 Dados Calculados:
14 Momento angular:  8393.5  Km^2/s
15 Excentricidade:  0.171212346284
16 Inclinação:  153.249228518  rad
17 Ascensão do Reta do Nodo Ascendente:  104.
    720714666  rad
18 Argumento do Perigeu:  20.0683166506  rad
19 Anomalia Verdadeira:  28.4456283066  rad
20
21
22 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):

```
10 ▲Dados fornecidos:
11 Posição 1: [-294.32, 4265.1, 5986.7]
12 Posição 2: [-1365.5, 3637.6, 6346.8]
13 Posição 3: [-2940.3, 2473.7, 6555.8]
14
15 Resultados
16 Velocidade para a posição (dois): [-6.21740189 -4.01216524 1.59898473]
17
18 Os elementos orbitais que representa a posição e a velocidade (dois)
19 Semi-eixo maior : 7813.0 Km
20 Excentricidade: 0.10010369281339042
21 Inclinação: 60.000470277369566 graus
22 Ascensão do nodo: 40.00144177286778 graus
23 Argumento do Perigeu: 30.074116831547773 graus
24 Anomalia verdadeira: 49.92565926551782 graus
25
26
27 Process finished with exit code 0
28
```

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

### Propagação orbital

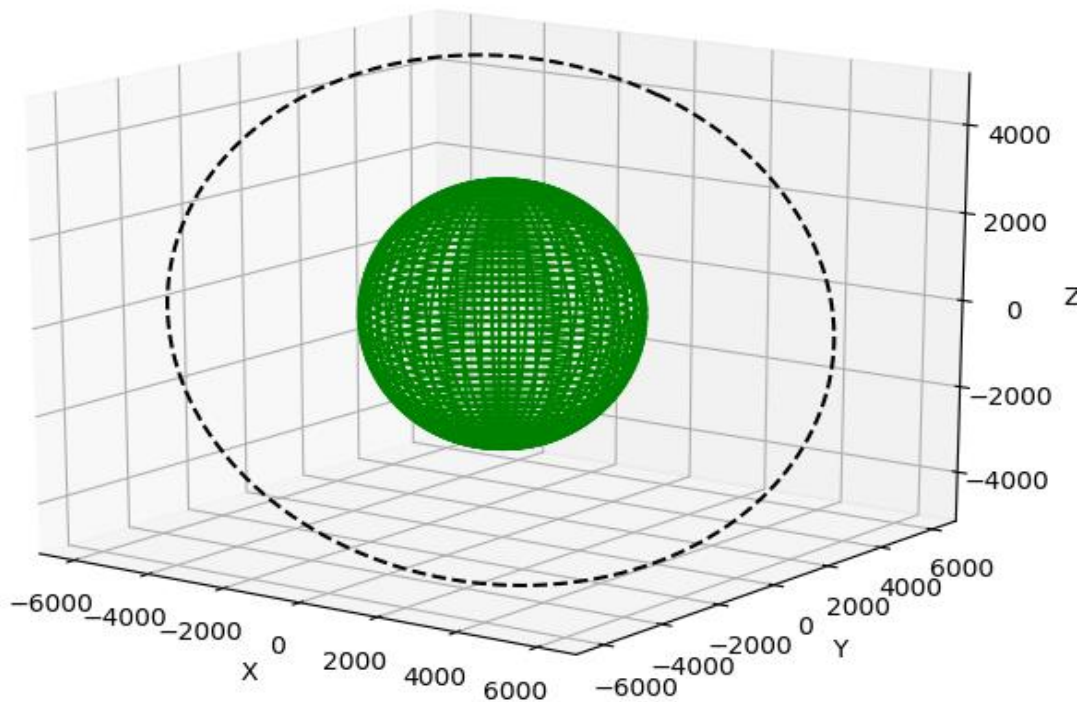


Figura 5 - Propagação Orbital  
Fonte: o Autor.

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 ( $A_s / m$ ) de  $2\text{m}^2 / \text{kg}$ . No tempo  $t_0$  (data juliana  $\text{JD}_0 = 2.438.400,5$ ) seus parâmetros orbitais são

Momento angular:  $63383.4 \text{ km}^2/\text{s}$

Excentricidade:  $0.025422$

Ascensão reta:  $45.3812^\circ$

Inclinação:  $88.3924^\circ$

Argumento do perigeu:  $227.493^\circ$

Anomalia verdadeira:  $343.427^\circ$

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

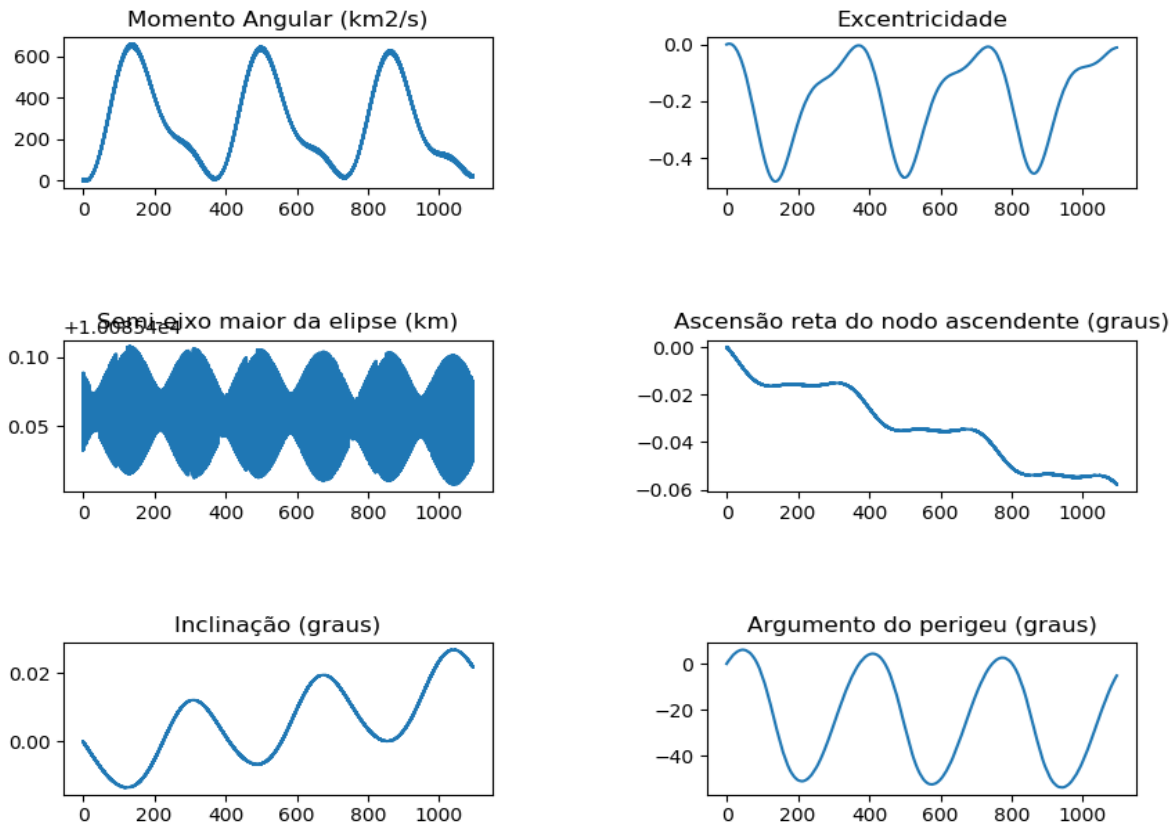


Figura 6 - Dados obtidos ação da radiação solar  
 Fonte: o Autor.

## 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



```
1 import numpy as np
2 ##### POSICIONAMENTO DIRETO #####
3 class direto(object):
4     R: np.ndarray
5     V: np.ndarray
6     def __init__(self, elk, mi):
7         self.elk = elk
8         self.mi = mi
9
10    def pos_dir(self):
11        momA = self.elk[0]
12        exce = self.elk[1]
13        incl = self.elk[2]
14        AscR = self.elk[3]
15        ArgP = self.elk[4]
16        AnoV = self.elk[5]
17
18        """ posição """
19        rx = pow(momA, 2)/self.mi*(1/(1+exce*np.cos(AnoV))
20)*np.cos(AnoV)
21        ry = pow(momA, 2)/self.mi*(1/(1+exce*np.cos(AnoV))
22)*np.sin(AnoV)
23        rz = pow(momA, 2)/self.mi*(1/(1+exce*np.cos(AnoV))
24)*0
25
26        """ velocidade """
27        vx = self.mi/momA*-np.sin(AnoV)
28        vy = self.mi/momA*(exce + np.cos(AnoV))
29        vz = self.mi/momA*0
30
31        Qx_1 = np.array([[np.cos(ArgP), np.sin(ArgP), 0],
32                        [-np.sin(ArgP), np.cos(ArgP), 0],
33                        [0, 0, 1]])
34
35        Qx_2 = np.array([[1, 0, 0],
36                        [0, np.cos(incl), np.sin(incl)],
37                        [0, -np.sin(incl), np.cos(incl)]])
38
39        Qx_3 = np.array([[np.cos(AscR), np.sin(AscR), 0],
40                        [-np.sin(AscR), np.cos(AscR), 0],
41                        [0, 0, 1]])
42
43        Qx1 = np.dot(Qx_2, Qx_3)
44        Qx = np.dot(Qx_1, Qx1)
45        Qx_T = np.matrix.transpose(Qx)
```



```
1
2
3 ##### MÉTODO INVERSO #####
4 import numpy as np
5 import math as mth
6 #import pyexcel ods3 as pods
7 from pos_direto import direto
8
9 class inverso(object):
10     def __init__(self, pos, vel):
11         self.pos = pos
12         self.vel = vel
13         self.mi = 398600
14
15     def pos_vel(self):
16         """ posição (r) """
17         r_e1 = np.asscalar(pow(self.pos[0], 2) + pow(self.
18 pos[1], 2) + pow(self.pos[2], 2))
19         self.r_e = np.sqrt(r_e1)
20
21         """ velocidade (v) """
22         v_e1 = np.asscalar(pow(self.vel[0], 2) + pow(self.
23 vel[1], 2) + pow(self.vel[2], 2))
24         self.v_e = np.sqrt(v_e1)
25
26         """ multiplicação vetorial de posição com a
27 velocidade """
28         vr_0 = np.dot(self.vel, self.pos)
29         self.vr = (vr_0 / self.r_e)
30
31     def mom_ang(self):
32         self.pos_vel()
33
34         """ Magnetude do momento angular (h) """
35         self.h = np.cross(self.pos, self.vel)
36         h_e0 = np.asscalar(pow(self.h[0], 2) + pow(self.h[
37 1], 2) + pow(self.h[2], 2))
38         return np.sqrt(h_e0)
39
40     def incl(self):
41         self.mom_ang()
42
43         """ inclinação (i) """
44         i_1 = (self.h[2] / self.mom_ang())
45         incl = np.arccos(i_1)
```



```
42     '''
43     if np.degrees(incl) > 90:
44         print('\t\033[1;31mOrbita retrograda\033[m')
45     '''
46     """ multiplicação vetorial de K com h """
47     k = np.array([0, 0, 1])
48     self.n1 = np.cross(k, self.h)
49     n1_e0 = np.asscalar(pow(self.n1[0], 2) + pow(self.
n1[1], 2) + pow(self.n1[2], 2))
50     self.n1_e = np.sqrt(n1_e0)
51     return incl
52
53     def asc_ret(self):
54         self.incl()
55
56         """ ascensão da reta """
57         return mth.acos((self.n1[0] / self.n1_e))
58
59     def excen(self):
60         self.asc_ret()
61
62         """ excentricidade """
63         self.e = (1 / self.mi * ((pow(self.v_e, 2) - self
.mi / self.r_e) * self.pos) - self.r_e * self.vr * self.
vel))
64         e_e0 = np.asscalar(pow(self.e[0], 2) + pow(self.e[
1], 2) + pow(self.e[2], 2))
65         return np.sqrt(e_e0)
66
67     def arg_per(self):
68         self.excen()
69
70         """ argumento do perigeu """
71         return mth.acos(np.dot(self.n1, self.e) / (self.
n1_e * self.excen()))
72
73     def anom_verd(self):
74         self.arg_per()
75
76         """ anomalia Verdadeira """
77         return mth.acos(np.dot(self.e, self.pos) / (self.
excen() * self.r_e))
78
79     def sem_Ex_maior(self):
80         """ Raio do Perigeu """
```



```
81         rp = pow(self.mom_ang(), 2) / self.mi * (1 / (1 +
self.excen() * np.cos(0)))
82         """ Raio do Apogeu """
83         ra = pow(self.mom_ang(), 2) / self.mi * (1 / (1 +
self.excen() * np.cos(180)))
84         """ Semi eixo maior """
85         return 0.5 * (int(rp + ra))
86
87     def anom_exce(self):
88         """ Anomalia Excentrica """
89         return 2 * mth.atan(np.tan(self.anom_verd() / 2)
/ np.sqrt((1 + self.excen()) / (1 - self.excen())))
90
91     def anom_media(self):
92         """ Anomalia Media """
93         return self.anom_exce() - self.excen() * np.sin(
self.anom_exce())
94
95     def periodo(self):
96         """ periodo """
97         return 2 * np.pi / np.sqrt(self.mi) * pow(self.
sem_Ex_maior(), 1.5)
98
99     def mov_med(self):
100         """ Movimento Médio """
101         return 2 * np.pi / self.periodo()
102
103     def anom_excI(self):
104         """ Anomalia Excentrica Inicial """
105         return mth.atan(np.sqrt((1 - self.excen()) / (1 +
self.excen()))) * (np.tan(self.anom_verd() / 2)) * 2
106
107     def tempo_inic(self):
108         """ Tempo Inicial """
109         return (self.anom_excI() - self.excen() * np.sin(
self.anom_excI())) / self.mov_med()
110
111
112     def saida_inv(self):
113         #direto(self.sem_Ex_maior(), self.excen(), self.
incl(), self.arg_per(), self.asc_ret(), self.anom_media
())
114         print("\033[1;31mSemieixo Maior:\033[m ", "\033[1
;30m", self.sem_Ex_maior(), "\033[m", "\033[1;31mKm\033[m
")
```



```
45     exce = self.pos_inv.excen()
46     mov_med = self.pos_inv.mov_med()
47     incl = self.pos_inv.incl()
48     asc_ret = self.pos_inv.asc_ret()
49     arg_per = self.pos_inv.arg_per()
50     tempo_inic = self.pos_inv.tempo_inic()
51
52     Tinicial = DJul(self.__ano, self.__mes, self.__dia
, self.__hora, self.__min, self.__seg).TempI
53
54     int_T = float(input('coloque o intervalo de tempo
para a propagação orbital:'))
55
56     while self.__seg < self.tempo:
57         self.__seg += int_T
58
59         data = DJul(self.__ano, self.__mes, self.__dia
, self.__hora, self.__min, self.__seg).TempI
60         delta_t = (data - Tinicial) * 86400
61         self.ListT.append(["%.1f" % round(delta_t, 2)]
)
62
63         temp_fin = tempo_inic + delta_t
64
65         np2 = temp_fin / self.tempo
66         t32 = (np2 - np.floor(np2)) * self.tempo
67
68         """ Anomalia Média """
69         anom_medF = mov_med * t32
70
71         """ Anomalia Excentrica """
72         erro = 1 * pow(10, -8)
73
74         if anom_medF < mth.pi:
75             E32 = anom_medF + exce / 2
76         else:
77             E32 = anom_medF - exce / 2
78
79         ratio = 1
80
81         while abs(ratio) > erro:
82             ratio = (E32 - exce * np.sin(E32) -
anom_medF) / (1 + exce * np.cos(E32))
83             E32 = E32 - ratio
84
```



```
85         """ Ascensão da Reta """
86         asc1 = -1.5 * (np.sqrt(self.mi) * self.j2 *
87         pow(self.R, 2) /
88         (((1 - pow(exce, 2)) ** 2) *
89         pow(sem_eix, 3.5))) * np.cos(incl)
90         g = np.degrees(asc1)
91         asc_retF = np.radians(np.degrees(asc_ret) + g
92         * delta_t)
93         self.ListAscR.append(asc_retF)
94         """ Argumento do Perigeu """
95         arg1 = -1.5 * (np.sqrt(self.mi) *
96         self.j2 * pow(self.R, 2) /
97         ((1 - pow(exce, 2)) ** 2) *
98         pow(sem_eix, 3.5))) * (2.5 *
99         pow(np.sin(incl), 2) - 2)
100        g2 = np.degrees(arg1)
101        arg_perF = np.radians(np.degrees(arg_per) +
102        g2 * delta_t)
103        self.ListArgP.append(arg_perF)
104        """ anomalia Verdadeira """
105        anom_verdF = 2 * mth.atan(np.sqrt((1 + exce)
106        / (1 - exce)) * np.tan(E32 / 2))
107        self.ListAnoV.append(anom_verdF)
108        """ posição """
109        rx = pow(self.mom_ang, 2) / self.mi * (1 / (1
110        + exce * np.cos(anom_verdF))) * np.cos(anom_verdF)
111        ry = pow(self.mom_ang, 2) / self.mi * (1 / (1
112        + exce * np.cos(anom_verdF))) * np.sin(anom_verdF)
113        rz = pow(self.mom_ang, 2) / self.mi * (1 / (1
114        + exce * np.cos(anom_verdF))) * 0
115        Npos = np.array([rx, ry, rz])
116        """ velocidade """
117        vx = self.mi / self.mom_ang * -np.sin(
118        anom_verdF)
119        vy = self.mi / self.mom_ang * (exce + np.cos(
120        anom_verdF))
121        vz = self.mi / self.mom_ang * 0
```





```
119         Nvel = np.array([vx, vy, vz])
120
121         Qx_1 = np.array([[np.cos(arg_perF), np.sin(
122             arg_perF), 0],
123                         [-np.sin(arg_perF), np.cos(
124             arg_perF), 0],
125                         [0, 0, 1]])
126
127         Qx_2 = np.array([[1, 0, 0],
128                         [0, np.cos(incl), np.sin(
129             incl)],
130                         [0, -np.sin(incl), np.cos(
131             incl)]]))
132
133         Qx_3 = np.array([[np.cos(asc_retF), np.sin(
134             asc_retF), 0],
135                         [-np.sin(asc_retF), np.cos(
136             asc_retF), 0],
137                         [0, 0, 1]])
138
139         Qx1 = np.dot(Qx_2, Qx_3)
140         Qx = np.dot(Qx_1, Qx1)
141         Qx_T = np.matrix.transpose(Qx)
142
143         posF = np.dot(Qx_T, Npos)
144         self.ListP.append(posF)
145
146         posF_e = np.sqrt(posF[0]**2+posF[1]**2+posF[2]
147             **2)
148         self.ListPesc.append(posF_e)
149
150     def saida_grafic(self):
151         self.Inv_Direto()
152         from gráficos import grafic
153         grafic.grafic_propag(self.ListP, self.ListT)
```



```
4 def los(rsat, rsun):
5     RE = 6378
6     r_sat = np.sqrt(rsat[0] ** 2 + rsat[1] ** 2 + rsat[2]
7     ** 2)
8     r_sun = np.sqrt(rsun[0] ** 2 + rsun[1] ** 2 + rsun[2]
9     ** 2)
10    theta = np.arccos(np.radians(np.dot(rsat, rsun)/r_sat/
11    r_sun))
12    thetasat = np.arccos(np.radians(RE/r_sat))
13    thetasun = np.arccos(np.radians(RE/r_sun))
14    if thetasat + thetasun <= theta:
15        lighth_switch = 0
16    else:
17        lighth_switch = 1
18
19    return lighth_switch
```



```
15     def ALS_solar_pos(self):
16
17         # Mean anomaly(deg)
18         M0 = 357.528 + 0.9856003 * self.n
19         M = np.mod(M0, 360)
20
21         # Mean longitude(deg)
22         L0 = 280.460 + 0.98564736 * self.n
23         L = np.mod(L0, 360)
24
25         # Apparent ecliptic longitude(deg)
26         lamda0 = L + 1.915 * np.sin(np.radians(M)) + 0.020
27         * np.sin(np.radians(2 * M))
28         lamda = np.mod(lamda0, 360)
29
30         # Obliquity of the ecliptic(deg)
31         eps = 23.439 - 0.0000004 * self.n
32
33         # Unit vector from earth to sun
34         u = np.array([np.cos(np.radians(lamda)),
35         np.sin(np.radians(lamda)) * np.cos(np.radians(eps))
36         ),
37         np.sin(np.radians(lamda)) * np.sin(np.radians(eps))
38         ])
39
40         # Distance from earth to sun(km)
41         rS = (1.00014 - 0.01671 * np.cos(np.radians(M)) -
42         0.000140 * np.cos(np.radians(2 * M))) * self.AU
43         # Geocentric position vector(km)
44         return lamda, eps, rS * u
```



```
1 import numpy as np
2 from pos_inverso import inverso
3
4 class tree_pos(object):
5     def __init__(self, mi, pos):
6         self.mi = mi
7         self.pos = pos
8         self.tol = 1e-4
9
10    def três_posições(self):
11        Pos1 = self.pos[0]
12        Pos2 = self.pos[1]
13        Pos3 = self.pos[2]
14
15        pos1 = np.sqrt(Pos1[0] ** 2 + Pos1[1] ** 2 + Pos1[
16    2] ** 2)
17        pos2 = np.sqrt(Pos2[0] ** 2 + Pos2[1] ** 2 + Pos2[
18    2] ** 2)
19        pos3 = np.sqrt(Pos3[0] ** 2 + Pos3[1] ** 2 + Pos3[
20    2] ** 2)
21
22        C12 = np.cross(Pos1, Pos2)
23        C23 = np.cross(Pos2, Pos3)
24        C31 = np.cross(Pos3, Pos1)
25
26        c23 = np.sqrt(C23[0] ** 2 + C23[1] ** 2 + C23[2]
27    ** 2)
28
29        if abs(np.dot(Pos1, C23)/pos1/c23) > self.tol:
30            print('\n Os vetores posição não são
31    coplanares.\n\n')
32
33        else:
34            N = pos1*C23 + pos2*C31 + pos3*C12
35            n = np.sqrt(N[0] ** 2 + N[1] ** 2 + N[2] ** 2)
36
37            D = C12 + C23 + C31
38            d = np.sqrt(D[0] ** 2 + D[1] ** 2 + D[2] ** 2)
39
40            S = Pos1 * (pos2 - pos3) + Pos2 * (pos3 - pos1
41    ) + Pos3 * (pos1 - pos2)
42
43            Vel2 = np.sqrt(self.mi / n / d) * (np.cross(D,
44    Pos2) / pos2 + S)
45
46
```



```
42
43     def deriv(t, f):
44         DJ = self.JD0 + int(t) / self.days
45         Lam, ep, rs = atração_solar(DJ).ALS_solar_pos(
46     )
47         lamda = np.radians(Lam)
48         eps = np.radians(ep)
49
50         h = f[0]
51         e = f[1]
52         i = f[2]
53         RA = f[3]
54         w = f[4]
55         TA = f[5]
56         phi = w + TA
57         elk = np.array([h, e, i, RA, w, TA])
58
59         posF, velF = direto(elk, self.mi).pos_dir()
60
61         pos_e = np.sqrt(posF[0] ** 2 + posF[1] ** 2 +
62     posF[2] ** 2)
63
64         nu = los(posF, rs)
65
66         pSR = nu*(self.S/self.c)*self.CR*self.As/self.
67     m/1000
68
69         sl = np.sin(lamda)
70         se = np.sin(eps)
71         sW = np.sin(RA)
72         si = np.sin(i)
73         su = np.sin(phi)
74         sT = np.sin(TA)
75
76         cl = np.cos(lamda)
77         ce = np.cos(eps)
78         cW = np.cos(RA)
79         ci = np.cos(i)
80         cu = np.cos(phi)
81         cT = np.cos(TA)
82
83         """ Perturbação da aceleração """
84         ur = sl*ce*cW*ci*su + sl*ce*sW*cu - cl*sW*ci*s
85     u+ cl*cW*cu + sl*se*si*su
```



```
83
84         us = sl*ce*cW*ci*cu - sl*ce*sW*su - cl*sW*ci*
      cu-cl*cW*su + sl*se*si*cu
85
86         uw = - sl*ce*cW*si + cl*sW*si + sl*se*ci
87
88         hdot = -pSR*pos_e*us
89
90         edot = -pSR*(h/self.mi*sT*ur+1/self.mi/h*((h*
      *2 + self.mi*pos_e)*cT + self.mi*e*pos_e)*us)
91
92         RAdot = -pSR*pos_e/h/si*su*uw
93
94         idot = -pSR*pos_e/h*cu*uw
95
96         wdot = -pSR*(-1/e/h*(h**2/self.mi*cT*ur - (p
      os_e + h**2/self.mi)*sT*us)-pos_e*su/h/si*ci*uw)
97
98         TAdot = h/pos_e**2 - pSR/e/h*(h**2/self.mi*cT
      *ur - (pos_e + h**2/self.mi)*sT*us)
99
100        #self.lista.append((hdot**2/self.mi/(1-edot**
      2)))
101        #self.ap = np.array([hdot**2/self.mi/(1-edot
      **2)])
102        #print("lista",self.lista)
103        #print("ap",self.ap)
104
105
106        return [hdot, edot, idot, RAdot, wdot, TAdot]
107
108        coe = Radau(deriv, [t0, tf], y0, max_step=nout, r
      tol=1.e-8, atol=1.e-8)
109
110        h_f, e_f, i_f, RA_f, w_f, TA_f = coe.y
111        t_f = coe.t
112
```