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

ESTUDO DA PERTUBAÇÃO DA ÓRBITA DE SATÉLITES ARTIFICIAIS DEVIDO AO ARRASTO ATMOSFÉRICO

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA

(PIBIC/INPE/CNPq)

Rodolfo Lyu Shimotsu (UBC, Bolsista PIBIC/CNPq)

E-mail: lyu1989@hotmail.com

Dr. Hans-Ulrich Pilchowski (INPE-ETE/DMC, Orientador)

E-mail: hans.pilchowski@inpe.com

Julho de 2018

Dedico este trabalho à minha família e esposa pela paciência e total apoio.

AGRADECIMENTOS

Agradeço ao Dr. Hans-Ulrich Pilchowski por me aceitar e orientar com sua experiência o estudo deste trabalho, me incentivando a seguir este caminho de estudo e pesquisas.

À minha esposa Fernanda de Melo Cachoeira que com muita paciência me incentivou todos os dias.

À minha mãe Margarete Tiyomi Ueda e meu pai Ryushi Shimotsu me oferecendo todo o apoio nos estudos para nunca desistir.

Aos meus colegas Anderson Bartholomeu de Oliveira e Erick de Souza Fernandes que compartilharam a mesma sala de estudo e pela amizade.

Ao Dr. Evandro Marconi Rocco, que me forneceu dados de órbitas não perturbadas, para que eu pudesse iniciar o desenvolvimento do algoritmo.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pela bolsa concedida.

E principalmente ao Instituto Tecnológico de Pesquisas Espaciais (INPE), por fornecer o local de estudo e pesquisa realizado neste trabalho.

RESUMO

Este trabalho tem como objetivo desenvolver um algoritmo computacional que seja capaz de fornecer a perturbação da órbita de um satélite artificial terrestre devido à ação do arrasto atmosférico. Órbitas cada vez mais baixas causam um efeito de desaceleração maior no satélite devido ao atrito denominado força de arrasto. Uma órbita elíptica sofre a desaceleração ao passar pelo perigeu, consequentemente sua altitude no apogeu tende a diminuir e se transformar em uma órbita circular. Neste trabalho foi possível iniciar a elaboração dos algoritmos utilizando equações newtonianas e keplerianas, para que fosse possível obter os elementos keplerianos, de órbitas de satélites artificiais terrestres, a partir de sua velocidade e sua posição e vice versa. Com os algoritmos, de determinação de órbita, aqui desenvolvidos, foi possível elaborar um algoritmo de propagação orbital, a partir do uso do conjunto dos métodos utilizados, sendo possível visualizar a propagação orbital do satélite em função do tempo. Para o desenvolvimento do algoritmo com a perturbação do arrasto atmosférico, foi necessário determinar o coeficiente de arrasto frontal, lateral e posterior tendo que levar em consideração a densidade do meio em relação à altitude baseando na tabela US Standard Atmosphere 1976 (USSA76), permitindo determinar a força de arrasto sobre o satélite ao longo de seu trajeto e ser aplicado no algoritmo de propagação para que seja possível visualizar a propagação orbital de um satélite considerando à ação do arrasto atmosférico.

Palavras-chave: Determinação da órbita de satélites, perturbação da órbita l, arrasto atmosférico.

SUMÁRIO

LISTAS E ILUSTRAÇÕES

LISTA DE SÍMBOLOS E ABREVIATURAS

μ – Constante gravitacional

r – Posição

v – Velocidade

ω_E – Velocidade angular de rotação da Terra

A – Área do satélite

m – Massa do satélite

C_D – Coeficiente de arrasto

ρ – Densidade Atmosférica em relação a altitude

D – Força de arrasto

p – Aceleração da perturbação

p – Raio do Perigeu

r_a – Raio do Apogeu

a – Semi-eixo maior da elipse

h – Momento Angular

e – Excentricidade

Ω – Ascensão Reta do Nodo Ascendente

ω – Argumento do Perigeu

θ – Anomalia Verdadeira

T – Período

1. INTRODUÇÃO

Satélites artificiais terrestres são influenciados por diversas forças perturbadoras, fazendo com que saia de sua órbita. Uma dessas forças estudada neste trabalho é a força de arrasto atmosférico. A determinação de órbita é necessária para determinar a posição e velocidade do satélite artificial em um instante de tempo.

2. OBJETIVOS

O objetivo deste trabalho é desenvolver um programa computacional que calcule a perturbação da órbita de um satélite artificial terrestre devido ao arrasto atmosférico, fornecendo os dados da perturbação orbital. Visando alcançar o objetivo de um algoritmo que possa ser inserido em programas computacionais mais abrangentes, destinados ao controle de órbitas.

3. FUNDAMENTAÇÃO TEÓRICA

3.1. PROBLEMA DOS DOIS CORPOS

O problema dos dois corpos é a representação do momento das forças em relação ao centro de massas, podendo ser reduzido ao problema de um corpo de massa desprezível em que o movimento de um corpo em relação a outro, é devido à atração de um campo gravitacional exercida por uma massa pontual:

$$\frac{d^2r}{dt^2} = -\mu\frac{r}{r^3}$$

3.2. VELOCIDADE RELATIVA

A velocidade relativa é calculada como $\boldsymbol{v_{rel} = v - v_{atm}}$,e considerando que a velocidade atmosférica seja a mesma que a velocidade angular de rotação da Terra então:

$$v_{rel} = v - \omega_E \times r$$

3.3. FORÇA DE ARRASTO

A força de arrasto é a força resistente do movimento de um objeto sólido através de um fluído, atuando em sentido oposto ao movimento do satélite, fazendo com que o satélite perca velocidade em sua posição próximo ao perigeu, consequentemente perdendo energia e não atingindo a altitude do apogeu anterior. No entanto, a altitude do perigeu se mantém quase a mesma (Figura 1.).

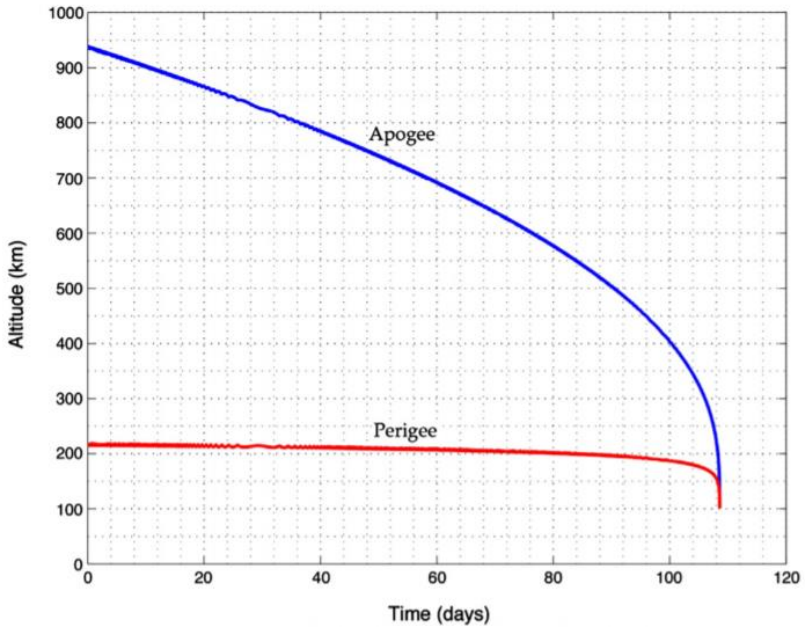

Figura 1- Variação da altitude do perigeu e apogeu devido ao arrasto atmosférico.

Fonte: CURTIS 2014.

A força de arrasto pode ser definida como:

$$D = \frac{1}{2}\rho v_{rel} C_D A$$

E a aceleração da perturbação da força de arrasto como ***p=D/m***, então:

$$p = \frac{-1}{2}\rho v_{rel}\left(\frac{C_D A}{m}\right) v_{rel}$$

Onde ρ é a densidade do ar em relação a altitude usando o modelo USSA76, A é área efetiva, C_D é o coeficiente de arrasto e m massa do satélite.

Para determinar a perturbação do movimento de dois corpos é adicionado o termo **p**.

$$\frac{d^2 r}{dt^2} = -\mu\frac{r}{r^3} + p$$

Com a equação diferencial de segunda como condição inicial pode-se resolver r e v para um intervalo de tempo.

$$\frac{d}{dt}\begin{Bmatrix} r \\ v \end{Bmatrix} = \begin{Bmatrix} v \\ a \end{Bmatrix} = \begin{Bmatrix} v \\ -\mu\dfrac{r}{r^3} + p \end{Bmatrix}$$

4. METODOLOGIA

O cálculo da perturbação da órbita se dará através de uma rotina computacional que calcula as forças de arrasto que atua sobre o satélite artificial. Inicialmente foram feitos estudos dos métodos de posicionamento de satélites artificiais, que possibilita calcular as propriedades da órbita de um satélite, tal que seja possível determinar essa órbita. Em seguida foi necessário o estudo dos sistemas de tempo, como conversões de data Juliana, e a utilização do conjunto dos métodos como Problema Direto, Problema Inverso e Método das Três Posições para determinar a órbita. Por fim o estudo do coeficiente de arrasto considerando a densidade do meio em relação à altitude baseando na tabela *US Standard Atmosphere 1976* (USSA76), permitindo determinar a força de arrasto sobre o satélite a partir do uso do método de Processo Direto, para determinar a posição e velocidade do satélite no instante de tempo, utilizando a equação do problema dos dois corpos para determinar a aceleração junto com a perturbação, a partir do vetor posição. Por fim a utilização de métodos numéricos de Integração de Órbita para determinar a propagação da órbita do satélite artificial com a perturbação.

5. ANÁLISES E RESULTADOS

O algoritmo do estudo foi desenvolvido em linguagem Python onde foi possível realizar testes com base em dados encontrados por CURTIS (2014).

Resultados encontrados a partir do método do Problema Direto (Figura 2):

```
Dados fornecidos:
Momento angular:   80000  Km^2/s
Excentricidade:   1.4
Inclinação:   0.5235987755982988  rad
Ascensão Reta do Nodo Ascendente:   1.0471975511965976
rad
Argumento do Perigeu:   0.6981317007977318  rad
Anomalia Verdadeira:   0.5235987755982988  rad

------------------------------------------------
Dados Calculados:
Posição:   [-4039.8959232   4814.56048018  3628.62470217]
Velocidade:   [-10.38598762  -4.77192164   1.743875  ]
```

Figura 2- Problema Direto.
Fonte: O autor

Resultados encontrados a partir do método do Problema Inverso (Figura 3):

```
Dados fornecidos:
Posição:   [-6045 -3490  2500]
Velocidade:   [-3.457  6.618  2.533]
------------------------------------------------
Dados Calculados:
Momento angular:   8393.5  Km²/s
Excentricidade:   0.17121234628445342
Inclinação:   153.2492285182475  rad
Ascensão do Reta do Nodo Ascendente:   104.72071466560382
  rad
Argumento do Perigeu:   20.068316650582467  rad
Anomalia Verdadeira:   28.445628306614967  rad
```

Figura 3- Problema Inverso.
Fonte: O autor

Resultados encontrados a partir do Método das Três Posições (Figura 4):

```
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
```

Figura 4- Método das Três Posições.

Fonte: O autor

Resultados encontrados da propagação orbital dados valores iniciais r_p , r_a, Ω, ω e θ.

Raio do Perigeu: r_p = 6593 km (altitude 215 km)

Raio do Apogeu: r_a = 7317 km (altitude 939 km)

Ascensão Reta do Nodo Ascendente: $\Omega = 340°$

Argumento do Perigeu: $\omega = 65.1°$

Anomalia Verdadeira: $\theta = 332°$

Determinação dos elementos *e, a, h, T* (Figura 5).

```
excentricidade (e):  0.05204888569374551
Semi-eixo maior da elipse (a):  6955.0  km
Momento angular (h):  52580.915736860414  km²/s
Período (T):  1.7157954810833342e-08  min
```

Figura 5- Parâmetros Orbitais.
Fonte: O autor

Determinação da Posição e Velocidade Inicial (Figura 6).

```
Momento angular (h):  52580.915736860414  km²/s
Período (T):  1.7157954810833342e-08  min
Posição Inicial:  [ 4109.93730389  2658.07613964 -4474.
2091795 ]
Velocidade Inicial:  [1.11846057 6.10992337 4.93178362]
```

Figura 6- Posição e Velocidade inicial.
Fonte: O autor

Propagação da altitude e velocidade no tempo (Figura 7).

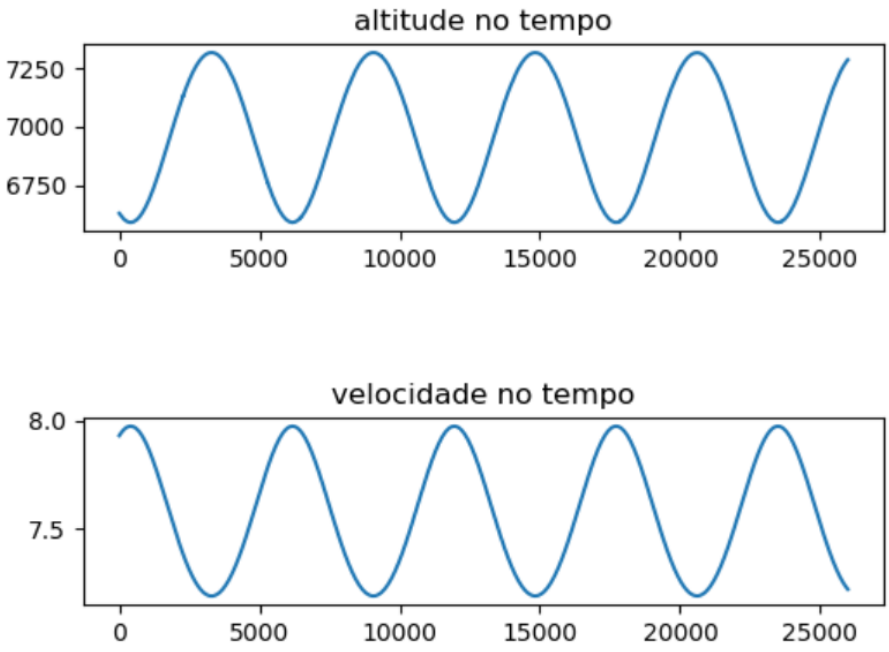

Figura 7-Variação da altitude no tempo e variação da velocidade no tempo sem a perturbação.

Fonte: O autor

Variação altitude devido à ação do arrasto atmosférico frontal (Figura 8).

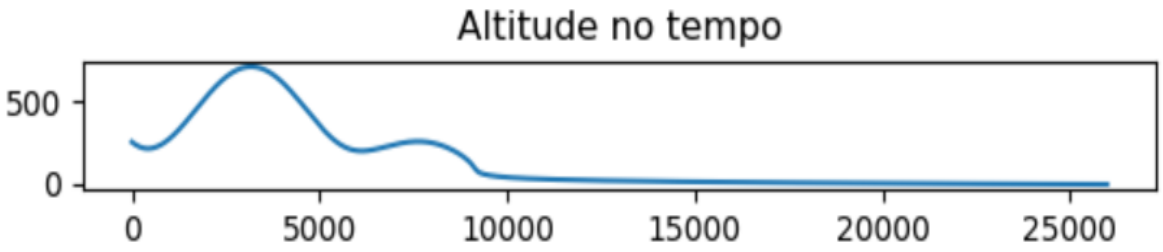

Figura 8- Variação da altitude devido à ação do arrasto atmosférico frontal.

Fonte: O autor

6. CONCLUSÃO

Os resultados obtidos pelos algoritmos desenvolvido neste estudo foram gratificantes tendo resultados com precisão de até quinze casas decimais, tendo a aproximação dos resultados encontrados por CURTIS (2014) nos métodos de Processo Direto, Inverso e Método das Três Posições. O desenvolvimento do algoritmo da ação do arrasto atmosférico frontal (Figura 8), nota-se que a altitude no apogeu diminui rapidamente comparada a altitude do perigeu levando a conclusão de que a força de arrasto atmosférico está sendo considerada na propagação da órbita.

REFERÊNCIAS BIBLIOGRÁFICAS

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 lista os algoritmos em python desenvolvidos e utilizado neste estudo, focando no algoritmo sem utilizar técnicas de interfaceamento gráfico, tendo a necessidade de entender a linguagem python e utilizar algumas bibliotecas do python como: numpy, scipy e matplotlib, para que seja possível utilizar os algoritmos.

ALGORITMO PROCESSO DIRETO

```
    def pos_dir(self):

    '''
    Este método encontra o vetor posição e velocidade a partir dos elementos keplerianos
    '''
    #Elementos Keplerianos
    h = self.elkep[0]   #momento angular (km²/s)
    e = self.elkep[1]   #excentricidade
    I = self.elkep[2]   #Inclinação (rad)
    w = self.elkep[3]   #Argumento do Perigeu (rad)
    RA = self.elkep[4]  #Ascensão Reta do Nodo Ascendente (rad)
    TA = self.elkep[5]  #Anomalia verdadeira (rad)
    #Vetores posição e velocidade no quadro perifocal (km e km/s)
    rx = pow(h, 2)/self.mi*(1/(1+e*np.cos(TA)))*np.cos(TA)
    ry = pow(h, 2)/self.mi*(1/(1+e*np.cos(TA)))*np.sin(TA)
    vx = self.mi/h*-np.sin(TA)
    vy = self.mi/h*(e + np.cos(TA))
    #Transformação das coordenadas
    Qx_1 = np.array([[np.cos(RA), np.sin(RA), 0],
                     [-np.sin(RA), np.cos(RA), 0],
                     [0, 0, 1]])
    Qx_2 = np.array([[1, 0, 0],
                     [0, np.cos(I), np.sin(I)],
                     [0, -np.sin(I), np.cos(I)]])
    Qx_3 = np.array([[np.cos(w), np.sin(w), 0],
                     [-np.sin(w), np.cos(w), 0],
                     [0, 0, 1]])
    Qx = np.dot(Qx_2, Qx_3)
    Qx_T = np.matrix.transpose(np.dot(Qx_1, Qx))
    #Vetor posição e velocidade no quadro equatorial geocêntrico (km e km/s)
    R = np.dot(Qx_T, np.array([rx, ry, 0]))
    V = np.dot(Qx_T, np.array([vx, vy, 0]))
    return R, V
```

ALGORITMO PROCESSO INVERSO

```
    def pos_inv(self):

    '''
    Este método encontra os elementos keplerianos a partir do vetor posição e velocidade
    '''
    r = np.sqrt(self.pos[0] ** 2 + self.pos[1] ** 2 + self.pos[2] ** 2) #módulo da posição
    v = np.sqrt(self.vel[0] ** 2 + self.vel[1] ** 2 + self.vel[2] ** 2) #módulo da velocidade
    vr = np.dot(self.pos, self.vel) / r #componente da velocidade radial
    H = np.cross(self.pos, self.vel)    #momento angular
    h = np.sqrt(H[0] ** 2 + H[1] ** 2 + H[2] ** 2)  #módulo do momento angular
    I = np.arccos(H[2] / h) #inclinação
    N_vet = np.cross(np.array([0, 0, 1]), H)  #Linha do nodo
    N = np.sqrt(N_vet[0] ** 2 + N_vet[1] ** 2 + N_vet[2] ** 2) #módulo da linha do nodo
    if N_vet[2] > 0:
        # print("O nodo ascendente esta no quadrante 1 ou 4")
        if N_vet[1] >= 0:
            n_ascendente = np.arccos(N_vet[2] / N)
            # print("\033[1;32mO nodo ascendente esta no quadrante 1\033[m")
        else:
            nodo_ascendente = 360 - np.arccos(N_vet[2] / N)
            # print("\033[1;32mO nodo ascendente esta no quadrante 4\033[m")
    else:
        # print("O nodo ascendente esta no quadrante 2 ou 3")
        if N_vet[2] >= 0:
            nodo_ascendente = np.arccos(N_vet[0] / N)
            # print("\033[1;32mO nodo ascendente esta no quadrante 2\033[m")
        else:
            nodo_ascendente = 360 - np.arccos(N_vet[0] / N)
            # print("\033[1;32mO nodo ascendente esta no quadrante 3\033[m")
    e_vet = (1 / self.mi) * ((((v**2 - (self.mi / r)) * self.pos) - r * vr * self.vel)) #excentricidade
    e = np.sqrt(e_vet[0] ** 2 + e_vet[1] ** 2 + e_vet[2] ** 2)  #módulo da excentricidade
    if (np.sum(N_vet) * np.sum(e_vet)) > 0:
        # print("O argumento do perigeu esta no quadrante 1 ou 4")
        if e_vet[2] >= 0:
            argumento_perigeu = np.arccos(np.dot(N_vet, e_vet) / (N * e))
            # print("\033[1;32mO argumento do perigeu esta no quadrante 1\033[m")
        else:
            argumento_perigeu = 360 - np.arccos(np.dot(N_vet, e_vet) / (N * e))
            # print("\033[1;32mO argumento do perigeu esta no quadrante 4\033[m")
    else:
        # print("O argumento do perigeu esta no quadrante 2 ou 3")
        if e_vet[2] >= 0:
            argumento_perigeu = np.arccos(np.dot(N_vet, e_vet) / (N * e))
            # print("\033[1;32mO argumento do perigeu esta no quadrante 2\033[m")
        else:
            argumento_perigeu = 360 - np.arccos(np.dot(N_vet, e_vet) / (N * e))
            # print("\033[1;32mO argumento do perigeu esta no quadrante 3\033[m")
    if (np.sum(e_vet) * np.sum(self.pos)) > 0:
        # print("O satelite esta no quadrante 1 ou 4")
        if vr >= 0:
            anomalia_verdadeira = np.arccos(np.dot(e_vet, self.pos) / (e * r))
            # print("\033[1;32mO satelite esta voando longe do perigeu\033[m")
        else:
            anomalia_verdadeira = 360 - np.arccos(np.dot(e_vet, self.pos) / (e * r))
            # print("\033[1;32mO satelite esta voando proximo ao perigeu\033[m")
    else:
        # print("O satelite esta no quadrante 2 ou 3")
        if vr >= 0:
```

```
            anomalia_verdadeira = np.arccos(np.dot(e_vet, self.pos) / (e * r))
            # print("\033[1;32mO satelite esta voando longe do perigeu\033[m")
        else:
            anomalia_verdadeira = 360 - np.arccos(np.dot(e_vet, self.pos) / (e * r))
            # print("\033[1;32mO satelite esta voando proximo ao perigeu\033[m")
    rp = ((h ** 2) / self.mi) * (1 / (1 + e * np.cos(0)))
    ra = ((h ** 2) / self.mi) * (1 / (1 + e * np.cos(180)))
    if (e > 1 + 10 ** 5):  # ORBITA HIPERBÓLICA
        print("\033[1;32mÓrbita hiperbólica\033[m")
        a = (h ** 2 / self.mi) * (1 / ((e ** 2) - 1))
    if (1 + 10 ** -5 > e > 1 - 10 ** -5):  # ORBITA PARABÓLICA
        print("\033[1;32mÓrbita Parabólica\033[m")
    if (1 - 10 ** -5 > e > 0):  # ELIPTICA OU CIRCULAR
        # print("\033[1;32mÓrbita Elíptica ou Circular\033[m")
        a = (rp + ra) / 2  # = (h**2/self.mi)*(1/(1-e**2))
        T = ((2 * self.pi) / np.sqrt(self.mi)) * np.sqrt(a ** 3)
    periodo = (((2 * self.pi) / np.sqrt(self.mi)) * np.sqrt(a ** 3)) / 3600
    anomalia_media = 2 * np.arctan(anomalia_verdadeira / (2 * np.arctan(np.sqrt((1 + e) / (1 - e)))))
    elkep = [h, e, np.degrees(I), np.degrees(nodo_ascendente), np.degrees(argumento_perigeu),
np.degrees(anomalia_verdadeira)]
    return elkep
```

ALGORITMO MÉTODO DAS TRÊS POSIÇÕES

```
def três_posições(self):

    Rx = self.pos[0]    #posição em x
    Ry = self.pos[1]    #posição em y
    Rz = self.pos[2]    #posição em z
    rx = np.sqrt(Rx[0] ** 2 + Rx[1] ** 2 + Rx[2] ** 2)#módulo da posição em x
    ry = np.sqrt(Ry[0] ** 2 + Ry[1] ** 2 + Ry[2] ** 2)#módulo da posição em y
    rz = np.sqrt(Rz[0] ** 2 + Rz[1] ** 2 + Rz[2] ** 2)#módulo da posição em z
    C23 = np.cross(Ry, Rz)
    c23 = np.sqrt(C23[0] ** 2 + C23[1] ** 2 + C23[2] ** 2)
    if abs(np.dot(Rx, C23)/rx/c23) > self.tol:
        print('\n  Os vetores não são coplanares.\n\n')
    else:
        N = rx*C23 + ry*np.cross(Rz, Rx) + rz*np.cross(Rx, Ry)
        n = np.sqrt(N[0] ** 2 + N[1] ** 2 + N[2] ** 2)
        D = np.cross(Rx, Ry) + C23 + np.cross(Rz, Rx)
        d = np.sqrt(D[0] ** 2 + D[1] ** 2 + D[2] ** 2)
        S = Rx * (ry - rz) + Ry * (rz - rx) + Rz * (rx - ry)
        Vel2 = np.sqrt(self.mi / n / d) * (np.cross(D, Ry) / ry + S)
        print('\nVelocidade para a posição (dois):', Vel2,
              "\n\nOs elementos orbitais que representa a posição e a velocidade (dois)")
        inverso(Ry, Vel2).saida_inv()
```

ALGORITMO DERIVAÇÃO DA POSIÇÃO E VELOCIDADE

```
def deriv(t,f): #Essa função calcula a aceleração do satélite do vetor posição e
velocidade no tempo
    R = f[0:3]
    self.List_R.append(R)
    self.List_t.append(t/self.days)
    r1 = np.sqrt(R[0] ** 2 + R[1] ** 2 + R[2] ** 2) #
    self.alt = r1 - self.RE     #altitude do satélite sem a interferência do
arrasto atmosférico
    #print(self.alt)
    rho = atmosfera(self.alt)   #Densidade atmosférica do ar em relação a
altitude
    V = f[3:6]  #Velocidade do satélite
    Vrel = V - np.cross(self.wE, R) #Velocidade relativa = Velocidade do
satélite - velocidade angular de rotação da Terra
    vrel = np.sqrt(Vrel[0] ** 2 + Vrel[1] ** 2 + Vrel[2] ** 2)  #Módulo da
velocidade relativa
    uv = Vrel / vrel    #Versor da velocidade relativa
    ap = -self.CD * self.A / self.m * rho * (1000 * vrel) ** 2 / 2 * uv
#acaleração pertubada
    a0 = -self.mi * R / r1 ** 3      #Fórmula reduzida do problema dos dois
corpos
    a = a0 + ap              #Aceleração da órbita + aceleração pertubada
    self.List_ar.append(a[0])
    self.List_as.append(a[1])
    self.List_aw.append(a[2])
    self.List_acel.append(np.sqrt(a[0] ** 2 + a[1] ** 2 + a[2] ** 2))
    #print(V,a)
    return [V[0], V[1], V[2], a[0], a[1], a[2]]
```

ALGORITMO DENSIDADE ATMOSFÉRICA EM RELAÇÃO À ALTITUDE

```
def atmosfera(z):# essa função calcula a densidade do ar em relação à altitude
do satélite


    # altitude geometrica (km)
    h = [0, 25, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 180,
200, 250, 300, 350, 400, 450,
         500, 600, 700, 800, 900, 1000]
    # densidade correspondente de USSA76 (kg/m³)
    r = [1.225, 4.008e-2, 1.841e-2, 3.996e-3, 1.027e-3, 3.097e-4, 8.283e-5,
1.846e-5, 3.416e-6, 5.606e-7,
         9.708e-8, 2.222e-8, 8.152e-9, 3.831e-9, 2.076e-9, 5.194e-10, 2.541e-10,
6.073e-11, 1.916e-11,
         7.014e-12, 2.803e-12, 1.184e-12, 5.215e-13, 1.137e-13, 3.070e-14,
1.136e-14, 5.759e-15, 3.561e-15]
    # altura scalar (km)
    H = [7.310, 6.427, 6.546, 7.360, 8.342, 7.583, 6.661, 5.927, 5.553, 5.703,
6.782, 9.973, 13.243, 16.332,
         21.652, 27.974, 34.934, 43.342, 49.755, 54.513, 58.019, 60.980, 65.654,
76.377, 100.587, 147.203,
         208.020]
    if (z > 1000):
        z = 1000
    elif (z < 0):
        z = 0
```

```
    for j in range(0, 27):
        if (z >= h[j] and z < h[j + 1]):
            i = j
    if (z == 1000):
        i = 26
    densidade = r[i] * np.exp(-(z - h[i]) / H[i])
    return densidade
```