Trabalho Final - Astronomia de posição

Thiago Laidler Vidal Cunha

2024/1

- O presente trabalho tem o objetivo de programar o cálculo da posição de um corpo celeste no céu com todas as correções necessárias. Nosso objetivo foi o de aproximar os resultados o máximo possível das efemérides retornadas pelo JPL. Nesse caso, utilizamos Eris como corpo exemplo.

Drive com Eris:

https://drive.google.com/drive/folders/1oDKUBtn3KVEEUTbQeL7PoYuC-sOvBIRq?usp=sharing

In [1]:
```
path_eris = 'Eris.bsp' #'/content/drive/MyDrive/Astronomia_de_posição/Eris.bsp'
```

> A função a seguir (rad_to_hours_convert) converte um ângulo medido em radianos para uma representação em horas, minutos e segundos (útil para RA).

In [3]:
```
def rad_to_hours_convert(angle):
  hours_angle = ((angle*180/np.pi)/15)
  rest_h = hours_angle - int(hours_angle)#((angle*180/np.pi))//15
  minutes_angle = rest_h*60
  seconds_angle = (minutes_angle - int(minutes_angle))*60
  print(f"{int(hours_angle)}:{(int(minutes_angle))}:{(seconds_angle)}") #Imprime as

rad_to_hours_convert(0.7000436538411695+np.pi) #Precisa ainda considerar o angulo c
```

```
14:40:26.29124160388514
```

- A função terra calcula os vetores de posição e velocidade de um observador terrestre em relação ao geocentro (centro da Terra) utilizando coordenadas retangulares equatoriais (se trata de uma conversão direta de uma das subrotinas do código em fortran para o python).

In [5]:
```
## Convertendo as sub-rotinas para python:

def terra(glon, glat, ht, st):   #glon: Longitude do observador em graus.
                                 #glat: Latitude geodésica do observador em graus.
                                 #ht: Altura do observador em metros.
                                 #st: Tempo sideral aparente local no meridiano de
    """
    Compute the position and velocity vectors of a terrestrial observer with respec

    Parameters:
```

```
    glon : float
        Longitude of observer with respect to reference meridian (East +) in degree
    glat : float
        Geodetic latitude (North +) of observer in degrees.
    ht : float
        Height of observer in meters.
    st : float
        Local apparent sidereal time at reference meridian in hours.

    Returns:
    pos : numpy array
        Position vector of observer with respect to geocenter, equatorial rectangul
        referred to true equator and equinox of date, components in AU.
    vel : numpy array
        Velocity vector of observer with respect to geocenter, equatorial rectangul
        referred to true equator and equinox of date, components in AU/day.
    """

    # Declaração de Constantes
    PI = 3.14159265358979324
    SECCON = 180.0 * 3600.0 / PI #Constante de conversão de segundos de arco para r


    ERAD = 6378.137  # Raio equatorial da Terra em quilômetros.
    F = 1.0 / 298.25642 #298.257223563  # Fator de achatamento do elipsoide terrest
    OMEGA = 7.2921150e-5  # Velocidade angular da Terra em radianos por segundo.
    AUKM = 1.49597870700e8  # Unidade astronômica em quilômetros


    DF2 = (1.0 - F)**2  #Fator de achatamento ao quadrado.
    PHI = glat * 3600.0 / SECCON  #Latitude em radianos.
    SINPHI = math.sin(PHI) #Seno e cosseno da latitude.
    COSPHI = math.cos(PHI)
    C = 1.0 / math.sqrt(COSPHI**2 + DF2 * SINPHI**2)  #Fatores de correção para a c
    S = DF2 * C     #Fatores de correção para a curvatura da Terra.
    ACH = ERAD * C + ht / 1000.0  #Raio equatorial e polar corrigidos pela altura d
    ASH = ERAD * S + ht / 1000.0


    STLOCL = (st * 54000.0 + glon * 3600.0) / SECCON  #Calcula o tempo sideral loca
    SINST = math.sin(STLOCL) #Calcula o seno (SINST) e o cosseno (COSST) do tempo s
    COSST = math.cos(STLOCL)

#Cálculo do vetor de posição (Calcula as componentes do vetor de posição em coorden

    pos = np.zeros(3)
    pos[0] = ACH * COSPHI * COSST
    pos[1] = ACH * COSPHI * SINST
    pos[2] = ASH * SINPHI

# Cálculo do vetor de velocidade (Calcula as componentes do vetor de velocidade em

    vel = np.zeros(3)
    vel[0] = -OMEGA * ACH * COSPHI * SINST
    vel[1] = OMEGA * ACH * COSPHI * COSST
    vel[2] = 0.0
```

```
    # Converter os vetores de posição e velocidade de quilômetros para unidades astronô
    pos = pos / AUKM
    vel = vel / AUKM * 86400.0

    return pos, vel

# Exemplo
glon = 0.0   # Longitude in degrees
glat = 51.5  # Latitude in degrees
ht = 0.0     # Height in meters
st = 0.0     # Sidereal time in hours

pos, vel = terra(glon, glat, ht, st)
print("Position vector (AU):", pos)
print("Velocity vector (AU/day):", vel)
```

```
Position vector (AU): [2.65956229e-05 0.00000000e+00 3.32114513e-05]
Velocity vector (AU/day): [-0.          0.00016756  0.        ]
```

- Normalização do ângulo (também uma conversão direta de uma das subrotinas):

In [6]:
```
def sla_DRANRM(angle):
  """Normalização para ângulo ficar entre 0~2pi"""
    D2PI = 6.28318530717958647692528676655  # 2 * pi

    result = angle % D2PI
    if result < 0.0:
        result += D2PI

    return result

# Exemplo
angle = -1.0
normalized_angle = sla_DRANRM(angle)
print("Normalized angle:", normalized_angle)
```

```
Normalized angle: 5.283185307179586
```

- Converter um vetor tridimensional em coordenadas cartesianas para coordenadas esféricas

In [7]:
```
def sla_DCC2S(v):
    x = v[0]
    y = v[1]
    z = v[2]
    r = math.sqrt(x * x + y * y)

    if r == 0.0:
        a = 0.0
    else:
        a = math.atan2(y, x)

    if z == 0.0:
```

```
        b = 0.0
    else:
        b = math.atan2(z, r)

    return a, b

# Exemplo
v = [1, 1, 1]
a, b = sla_DCC2S(v)
print("Longitude (A):", a)
print("Latitude (B):", b)

# Retorna a longitude(A) e a latitude (B).
```

```
Longitude (A): 0.7853981633974483
Latitude (B): 0.6154797086703873
```

In [8]:
```
from math import modf, floor

def sla_DD2TF(ndp, days):
    '''
    Função sla_DD2TF: Esta função converte um valor em dias para horas, minutos, se
    Função sla_DR2TF: Esta função converte um ângulo em radianos para "turns" (volt
    '''
    sign = '+' if days >= 0 else '-'
    days = abs(days)

    total_seconds = days * 86400  # 1 day = 86400 seconds (Converte o valor de dias
    fraction, total_seconds = modf(total_seconds)
    total_seconds = int(total_seconds)

    hours = total_seconds // 3600
    total_seconds %= 3600
    minutes = total_seconds // 60
    seconds = total_seconds % 60

    # Adjust the fraction part
    fraction_seconds = round(fraction * 10**ndp)

    ihmsf = [hours, minutes, seconds, fraction_seconds]

    return sign, ihmsf
```

In [9]:
```
def sla_DR2TF(ndp, angle):
    T2R = 6.28318530717958647692528  # 2 * pi

    # Convert radians to turns
    turns = angle / T2R

    # Call the auxiliary function
    sign, ihmsf = sla_DD2TF(ndp, turns)

    return sign, ihmsf

# Exemplo de uso:
ndp = 2
```

```
angle = 1.0  # em radianos
sign, ihmsf = sla_DR2TF(ndp, angle)
print("Sign:", sign)
print("IHMSF:", ihmsf)
```

```
Sign: +
IHMSF: [3, 49, 10, 99]
```

- A seguir está um script que desenolvi no intuito de facilitar a busca pelas efemérides sem a necessidade de abrir o site do JPL. Embora possa ajudar, o programa presente não utiliza esse script para buscar a efeméride. Pode ser utilizado e testes futuros.

In [10]:
```
#importação das bibliotecas

from astroquery.jplhorizons import Horizons
import matplotlib.pyplot as plt
import datetime as dt
from dateutil.relativedelta import relativedelta
from pytz import UTC as utc

class Efemeride:
    '''
    Classe para buscar efeméride direto do JPL de maneira fácil e rápida
    '''
    def __init__(self, name, start, end, step='d'):
        self.name = name  #nome do objeto celestre
        self.start = start #data de inicio do período para o qual se deseja obter a
        self.end = end  #data do fim do ériodo
        self.step = step

    def datetime_to_epochs_days(self, dt, days=1):
        """
        Converte um objeto datetime em epochs compatíveis com o Astroquery.

        Parâmetros:
            dt (datetime): O objeto datetime a ser convertido.

        Retorna:
            epochs (dict): Um dicionário contendo as informações de epochs no forma
                           adequado para o Astroquery.
        """
        start = dt.strftime('%Y-%m-%d %H:%M:%S')

        end = (dt + relativedelta(days=days)).strftime('%Y-%m-%d %H:%M:%S')

        epochs = {'start': start, 'stop': end, 'step': '1' + self.step}

        return epochs
#obtenção das Efemérides

    def get_ephemerides_by_astropy(self, epoch=None):
        if epoch is None:
            obj = Horizons(id=self.name)
            eph = obj.ephemerides()
        else:
```

```
            obj = Horizons(id=self.name, epochs=epoch)
            eph = obj.ephemerides()
        return eph

# Obtém e retorna as efemérides como um "DataFrame"
    def fetch_ephemerides_dataframe(self):
        start = self.start.strftime('%Y-%m-%d %H:%M:%S')
        end = self.end.strftime('%Y-%m-%d %H:%M:%S')
        epochs = {'start': start, 'stop': end, 'step': '1' + self.step}
        eph = self.get_ephemerides_by_astropy(epoch=epochs)
        print(f"\nEfemérides de {self.name} para até {self.end.year} anos:")
        return eph

#Plotagem do RA e DEC (Obtém as efemérides e plota a evolução de RA e DEC ao longo
    def plotar_RA_e_DEC_JPL(self):
        start = self.start.strftime('%Y-%m-%d %H:%M:%S')
        end = self.end.strftime('%Y-%m-%d %H:%M:%S')
        epochs = {'start': start, 'stop': end, 'step': '1' + self.step}
        eph = self.get_ephemerides_by_astropy(epoch=epochs)
        print(f"\nEfemérides de {self.name} para até {self.end.year} anos:")

        # Plot
        plt.figure(figsize=(12, 6))
        plt.subplot(211)
        plt.title(f"Evolução de RA com o tempo ({self.name})")
        plt.plot(eph['datetime_str'], eph['RA'], label='RA')
        plt.xlabel('Tempo')
        plt.ylabel('RA')
        plt.legend()

        plt.subplot(212)
        plt.title(f"Evolução de DEC com o tempo ({self.name})")
        plt.plot(eph['datetime_str'], eph['DEC'], label='DEC')
        plt.xlabel('Tempo')
        plt.ylabel('DEC')
        plt.legend()

        plt.tight_layout()
        plt.show()
```

In [11]:
```
#Exemplo:
efem = Efemeride(name='Eris (system barycenter)', start = dt.datetime(2010, 1, 12,
```

In [12]:
```
#O output desse script é um data frame. Uma tabela com as informações da efeméride
efem.fetch_ephemerides_dataframe()
```

```
Efemérides de Eris (system barycenter) para até 2015 anos:
```

Out[12]: *Table masked=True length=1815*

targetname	datetime_str	datetime_jd	solar_presence	lunar_presence	RA	
---	---	d	---	---	deg	
str32	str20	float64	str1	str1	float64	flo
Eris (system barycenter) (201361	2010-Jan-12 20:13:36	2455209.342777778			24.30282	-4.5
Eris (system barycenter) (201361	2010-Jan-13 20:13:36	2455210.342777778			24.30258	-4.5
Eris (system barycenter) (201361	2010-Jan-14 20:13:36	2455211.342777778			24.3025	-4.5
Eris (system barycenter) (201361	2010-Jan-15 20:13:36	2455212.342777778			24.3026	-4.5
Eris (system barycenter) (201361	2010-Jan-16 20:13:36	2455213.342777778			24.30287	-4.5
Eris (system barycenter) (201361	2010-Jan-17 20:13:36	2455214.342777778			24.3033	-4.5
Eris (system barycenter) (201361	2010-Jan-18 20:13:36	2455215.342777778			24.30391	-4.5
Eris (system barycenter) (201361	2010-Jan-19 20:13:36	2455216.342777778			24.30468	-4.5
Eris (system barycenter) (201361	2010-Jan-20 20:13:36	2455217.342777778			24.30563	-4.5
...	...	...	...	...	...	
Eris (system barycenter) (201361	2014-Dec-22 20:13:36	2457014.342777778			24.97986	-3.3
Eris (system barycenter) (201361	2014-Dec-23 20:13:36	2457015.342777778			24.97601	-3.3
Eris (system barycenter) (201361	2014-Dec-24 20:13:36	2457016.342777778			24.97231	-3.3

targetname	datetime_str	datetime_jd	solar_presence	lunar_presence	RA	
Eris (system barycenter) (201361	2014-Dec-25 20:13:36	2457017.342777778			24.96876	-3.3
Eris (system barycenter) (201361	2014-Dec-26 20:13:36	2457018.342777778			24.96538	-3.3
Eris (system barycenter) (201361	2014-Dec-27 20:13:36	2457019.342777778			24.96215	-3.3
Eris (system barycenter) (201361	2014-Dec-28 20:13:36	2457020.342777778			24.95909	-3.3
Eris (system barycenter) (201361	2014-Dec-29 20:13:36	2457021.342777778			24.95619	-3.3
Eris (system barycenter) (201361	2014-Dec-30 20:13:36	2457022.342777778			24.95345	-3
Eris (system barycenter) (201361	2014-Dec-31 20:13:36	2457023.342777778			24.95087	-3.3

- calcular a correção de tempo (tau) entre um observador e um alvo celeste considerando a relatividade geral, utilizando as rotinas fornecidas pelo SPICE toolkit através do "spiceypy"

In [13]:
```
def TAU(etobs, targ, tp):
    #spkez: Rotina do SPICE para obter a posição e velocidade de um corpo em relaçã
    #399: ID da Terra.
    #10: ID do Sol.

    eb, ltim = spkez(399, etobs, 'J2000', 'NONE', 0)
    sb, ltim = spkez(10, etobs, 'J2000', 'NONE', 0)

    tau = 0
    cont = 0
    aux = 1

    while np.abs(tau - aux) > 1e-14:
    #while cont < 5:
      sbtau, ltim = spkez(10, etobs-tau, 'J2000', 'NONE', 0)
      qb, ltim = spkez(targ, etobs-tau, 'J2000', 'NONE', 0)

      p = qb[:3] - (eb[:3]+tp)
      e = (eb[:3]+tp) - sb[:3]
      q = qb[:3] - sbtau[:3]
```

```
        pn = np.linalg.norm(p)
        en = np.linalg.norm(e)
        qn = np.linalg.norm(q)

        aux = tau
        tau = (pn + (2*(G*M)/c**2)*np.log((en+pn+qn)/(en-pn+qn)))/c
        cont +=1

    return tau, sbtau

    #tau: Correção de tempo final.
    #sbtau: Posição do Sol no tempo corrigido por tau.
```

Abaixo temos o código principal. Ele que usará as funções criadas anteriormente para fazer o calculo da posição e velocidade do objeto celeste em uma data específica considerando diversas correções astronômicas como a deflexão da luz pelo Sol e a aberração diurna (causada pela velocidade do observador na superfície da Terra em rotação).

No código está sendo aplicado a correção pela **Deflexão** a partir de $\vec{p_1} = \vec{p} + (2GM_{sol}/c^2E)((\vec{p}\vec{q})\vec{e} - (\vec{e}\vec{p})\vec{q})/(1 + \vec{q}\vec{e})$, após o cáluclo do tempo luz (τ) que inclui o efeito gravitacional de retardamento devido ao sol.

Após isso é aplicado a correção pela **Aberração** pela formulação relativística, que é dada considerando a direção do objeto que se move numa velocidade v (que segundo as relaçõs de Lorentz $\gamma = (1 - v^2)^{-1/2}$). Temos que a velocidade da luz observada (pós aberração e deflexão) é:

$$\vec{U} = -c \cdot \vec{p}$$

$$\vec{U} = \frac{1}{(1-\frac{\vec{u}\vec{v}}{c^2})}[\gamma^{-1}\vec{u} - \vec{v} + \frac{\vec{u}\cdot\vec{v}\cdot\vec{v}}{(\gamma^{-1}+1)c^2}]$$

Logo:

$$-c\vec{p_2} = \frac{1}{(1+\frac{\vec{p}\vec{v}}{c})}[-c\vec{p}\gamma^{-1} + (1 + \frac{\gamma}{1+\gamma}\frac{c\vec{p}\vec{v}\vec{v}}{c^3})]$$

$$\vec{p_2} = \frac{1}{1+\vec{p}\vec{v}/c}[\gamma^{-1}\vec{p} + (1 + \frac{\gamma}{\gamma+1}\frac{\vec{p}\vec{v}}{c})\frac{\vec{v}}{c}]$$

As correções feitas nesse código seguem essa ordem pois ao longo do processo o referencial muda (ICRS para geocentro, geocentro para topocentro), e é importante manter um referencial comum.

In [54]:
```
#Reorganizei o código para ficar mais user friendly e mais organizado

getcontext().prec = 50  # Tentativa de aumentar a precisão dos cálculos

# Constantes
AU = 149597870.7  # Unidade Astronômica (km)
D2S = 86400.0  # Segundos em um dia
F = 4.84813681109536e-9
```

```
K = 0.210949526569699
DPI = np.pi
C = 299792.458  # Velocidade da luz (km/s)
G = 6.67430e-20  # Constante gravitacional em km^3 x kg^-1 x s^-2
M = 1.98847e+30  # Massa do Sol em kg
J2000 = 2451545.0  # Data Juliana para J2000

def convert_to_et(date):
    """
    Converte uma data em formato legível para o tempo ephemeris (ET).
    """
    et = str2et(date)
    return et

def alpha_decimal_hours_to_hms(decimal_hours):
      decimal_hours = (decimal_hours*180./np.pi) / 15.
      hours = int(decimal_hours)
      minutes = int((decimal_hours - hours) * 60.)
      seconds = (decimal_hours - hours - minutes / 60.) * 3600.
      int_sec = int(seconds)
      frc_sec = int(round(100000.*(seconds-int_sec),0))
      return hours, minutes, int_sec, frc_sec

def delta_decimal_hours_to_hms(decimal_hours):
      sinal = '+'
      if (decimal_hours < 0.):
        decimal_hours = abs(decimal_hours)
        sinal = '-'
      decimal_hours = (decimal_hours*180./np.pi)
      hours = int(decimal_hours)
      minutes = int((decimal_hours - hours) * 60.)
      seconds = (decimal_hours - hours - minutes / 60.) * 3600.
      int_sec = int(seconds)
      frc_sec = int(round(10000.*(seconds-int_sec),0))
      return sinal,hours, minutes, int_sec, frc_sec

def position_predict(date, leap, de, eph, glon, glat, ht):
    """
    Função principal para calcular a posição e velocidade do corpo celeste.
    """
    furnsh(leap)
    furnsh(de)
    furnsh(eph)

    etobs = convert_to_et(date)
    # Converte a data de entrada para tempo ephemeris (ET)

    # Cálculo do Tempo Sideral
    jdtt = float(etobs / D2S) + J2000
    jdutc = jdtt - (+32.184 + 37.0) / D2S

    tta = J2000
    ttb = jdtt - tta
    uta = J2000
    utb = jdutc - uta
```

```
    gast =  erfa.gst06a(uta, utb, tta, ttb)
    # Calcular o tempo sideral aparente em Greenwich (GAST)
    last = (gast + glon * DPI / 180.0) % (2.0 * DPI)
    if last < 0.0:
        last += 2.0 * DPI
    last = last * 180.0 / DPI / 15.0

    gast = gast % (2.0 * DPI)
    if gast < 0.0:
        gast += 2.0 * DPI
    gast = gast * 180.0 / DPI / 15.0

    # Calcula a posição e velocidade da Terra no ICRF
    # (International Celestial Reference Frame)
    pos, vel = terra(glon, glat, ht, gast)
    rbpn = erfa.pnm06a(tta, ttb)
    pos_icrf = np.dot(np.linalg.inv(rbpn), pos)
    vel_icrf = np.dot(np.linalg.inv(rbpn), vel)

    SPKID = int(get_spkid(eph))  # Obtém o SPKID do arquivo de efemérides

    tau, sbt = TAU(etobs, SPKID, AU * pos_icrf)
    bb, _ = spkez(SPKID, etobs - tau, 'J2000', 'NONE', 0)
    eb, _ = spkez(399, etobs, 'J2000', 'NONE', 0)
    sb, _ = spkez(10, etobs, 'J2000', 'NONE', 0)

    unload(leap)
    unload(de)
    unload(eph)

    # Calcula posição astrométrica do corpo
    pos1 = -pos_icrf * AU - eb[:3] + bb[:3]
    vel_tot_icrf = vel_icrf * AU / D2S + eb[3:]

#####Correções Astrométricas
    # Correção pela deflexão da luz pelo Sol
    p = pos1
    e = eb - sb
    q = bb - sbt

    norm_p = np.linalg.norm(p)
    pn = p / norm_p

    norm_q = np.linalg.norm(q)
    qn = q / norm_q

    norm_e = np.linalg.norm(e)
    en = e / norm_e

    pq = np.dot(pn[:3], qn[:3])
    ep = np.dot(en[:3], pn[:3])
    qe = np.dot(qn[:3], en[:3])
    p1 = np.zeros(3)

    for i in range(3):
        p1[i] = pn[i] + (2.0 * G * M / (C * C * norm_p)) *
```

```
        (pq * en[i] - ep * qn[i]) / (1.0 + qe)

    # Correção pela aberração
    p2 = np.zeros(3)
    pv = np.dot(p1, vel_tot_icrf)
    v2 = np.dot(vel_tot_icrf, vel_tot_icrf)
    gamma = 1 / np.sqrt(1 - v2 / (C * C))
    for i in range(3):
        p2[i] = 1.0 / (1.0 + pv / C) * (p1[i] / gamma +
             (1.0 + (gamma / (1.0 + gamma)) * pv / C) * vel_tot_icrf[i] / C)

    # Determinado a posição com respeito ao equador e equinócio verdadeiros
    p3 = np.dot(rbpn, p2)

    # Mudança para
    al = np.arctan2(p3[1], p3[0])
    de = np.arctan2(p3[2], np.sqrt(p3[0] * p3[0] + p3[1] * p3[1]))

    # Calcula a posição astrométrica final do corpo celeste considerando todas
    # as correções anteriores
    # Conversão de Coordenadas para Horas, Minutos e Segundos
    hours, minutes, int_sec, frc_sec = alpha_decimal_hours_to_hms(al)
    alpha = f"{hours:02}h {minutes:02}m {int_sec:02}.{frc_sec:05}s"

    sinal, hours, minutes, int_sec, frc_sec = delta_decimal_hours_to_hms(de)
    delta = f"{sinal}{hours:02}° {minutes:02}' {int_sec:02}.{frc_sec:04}"

    return alpha, delta

def get_spkid(eph_path):
    """
    Função auxiliar para obter o SPKID a partir do arquivo de efemérides.
    """
    import os
    os.system(f"./commnt -r {eph_path} | grep 'Target SPK ID   :' | awk '{{print $5
    with open('spkid', 'r') as file:
        SPKID = file.read().strip()
    return SPKID
```

In [55]:

```
# Utilizando a função position_predict para calcular a posição de Eris
if __name__ == "__main__":
    date = "2028-OCT-07 03:00:59.884"
    leap = 'naif0012.tls'
    de = 'de440.bsp'
    eph = path_eris
    glon = 314.4173
    glat = -22.5354318
    ht = 1817.54

    alpha, delta = position_predict(date, leap, de, eph, glon, glat, ht)
    print(f"alpha: {alpha}")
    print(f"delta: {delta}")
```

```
alpha: 01h 50m 59.28935s
delta: +00° 36' 22.0426
```

In [1]:
```
######## Conversor para PDF #################
!jupyter nbconvert --to webpdf --allow-chromium-download TrabalhoPosicao_ThiagoLaid
```

In [ ]:
```

```