Chapters from the history of astronomy
Python computer exercises

Marc van der Sluys
Radboud University Nijmegen
The Netherlands
http://astro.ru.nl/~sluys/

June 15, 2020

Contents

0 Computer exercise 0: Code template

Code listing 0.1 shows a Python template that reads our tailored extract from the Hipparcos catalogue [1] **BrightStars-1.1.csv** with stars up to seventh magnitude, sets the plot boundaries, selects the stars within those boundaries, and plots them to screen. Since you can use this as a code template for the next Computer exercises, you should make sure that you understand every line before you continue. The result of the code can be found in Figure 1. Before you continue, you should run the code and verify whether it produces the correct result.

Listing 0.1: plot_hipparcos.py: a Python code template to read the Hipparcos catalogue and plot a star map. The result is displayed in Fig. 1.

```
1  #!/bin/env python3
2
3  import math as m
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  r2d = m.degrees(1)
8  d2r = 1.0/r2d
9
10 # Read columns 2-4 from the input file, skipping the first 11 lines:
11 hip = np.loadtxt('BrightStars-1.1.csv', skiprows=11, delimiter=',',
12                  usecols=(1,2,3))
13
14 # Use comprehensible array names:
15 mag   = hip[:,0]
16 ra    = hip[:,1]
17 dec   = hip[:,2]
18 limMag = 7.0
19 sizes = 30*(0.5 + (limMag-mag)/3.0)**2
20
21 # Set the plot boundaries:
22 raMin  = 26.0*d2r
23 raMax  = 50.0*d2r
24 decMin = 10.0*d2r
25 decMax = 30.0*d2r
26
27 # Select the stars within the boundaries:
28 sel = (ra > raMin) & (ra < raMax) & (dec > decMin) & (dec < decMax)
29 # sel = ra < 1e6  # Select all stars
30
31 plt.figure(figsize=(9,7))                  # Set png size to 900x700 (dpi=100)
32
33 # Make a scatter plot.  s contains the *surface areas* of the circles:
34 plt.scatter(ra[sel]*r2d, dec[sel]*r2d, s=sizes[sel])
35
36 plt.axis('scaled')
37 plt.axis([raMax*r2d,raMin*r2d, decMin*r2d,decMax*r2d])
38 plt.xlabel(r'$\alpha_{2000}$ ($^\circ$)')
39 plt.ylabel(r'$\delta_{2000}$ ($^\circ$)')
40
41 plt.tight_layout()
42 plt.show()
43 # plt.savefig("hipparcos.pdf")
44 plt.close()
```

Line 1 of the code is called the *shebang*. This tells the script which interpreter to use, in this case Python 3 on a Linux system. You can adapt this to your system, or run the code instead with *e.g.*

`python3 plot_hipparcos_template.py`.

In lines 3–5, we import three Python modules we need:

- The standard **math** module for mathematical functions and constants.
- The **numpy** module for powerful mathematical functions, in particular involving arrays [2]. We use for example the `np.loadtxt()` function to read the data file and store its contents in the 2D array `hip`.
- The **pyplot** module from the **matplotlib** library for plotting [3]. For example, we use the function `plt.scatter()` to create a scatter plot with circles at the coordinates of the stars.

The `as` statement allows us to henceforth invoke the three modules as **m**, **np** and **plt** respectively.

0.1 Questions

1. What is the value of `r2d` in line 7? What does this mean?
2. How would the code in line 11 change if we would also read the Hipparcos number of each star (the first column in the file) into the `hip` array? Which other code lines should be adapted in order to keep the code working?
3. Explain the square power (`**2`) when computing star sizes from their magnitudes in line 18.
4. What is the shape of the array `sel`? What are its elements?
5. Write the plot to hipparcos.pdf instead of to screen. What is the file size of the resulting pdf file?
6. How does the size of the pdf file change when you uncomment code line 30 in order to select all stars? How is the run time of your program affected? Explain the changes.
7. What does line 37 (`plt.axis('scaled')`) do? Why is this necessary? Comment it out to see what happens. Replace `'scaled'` in line 37 with `'equal'`. How does this solve the same issue?
8. What happens if you print the axis labels without the leading **r**?
9. What happens if you comment out line 42 (`plt.tight_layout()`)?

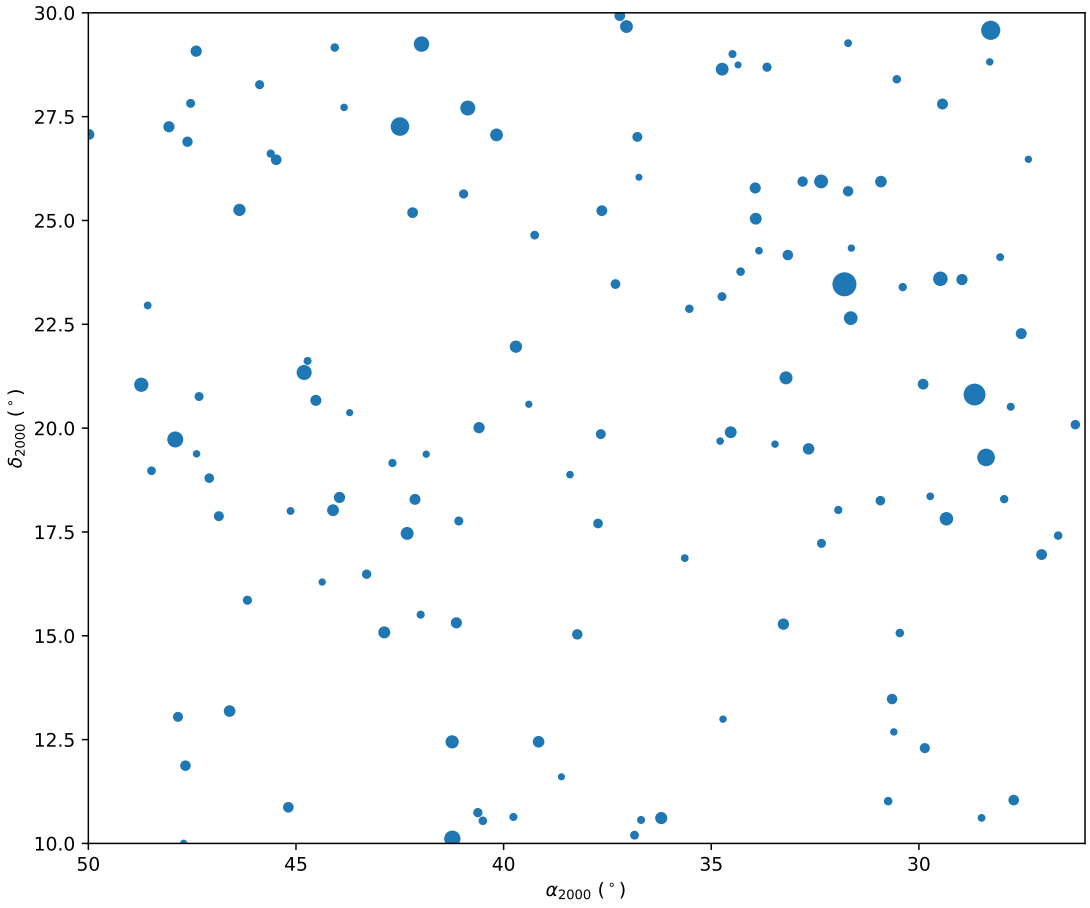

Figure 1: The sky map resulting from Code listing 0.1.

1 Computer exercise 1: Ecliptical map

Adapt the code from the previous exercise to make a plot of the constellation Aries in **ecliptic coordinates**. In order to do so, write the function `eq2ecl()` that takes the right ascension, declination and the obliquity of the ecliptic as input parameters and returns ecliptic latitude and longitude. A function in Python can be defined as follows:

```
def eq2ecl(ra,dec, eps):
    <code to convert ra, dec & eps to lon & lat>
    return lon,lat
```

We can call the function with:

```
lon,lat = eq2ecl(ra,dec, eps)
```

It is important to realise that this function will be called with *NumPy arrays* as input parameters and return values. Hence, the code that computes `lat,lon` should use the NumPy versions of *e.g.* the trigonometric functions — for example `np.sin(ra)` rather than `m.sin(ra)`.

Design and implement a way to compute and set the plot limits of the ecliptic map, using the function `eq2ecl()`.

Finally, ensure that you select the stars to plot according to the newly computed map limits.

2 Computer exercise 2: Horizontal map

Adapt your code from the previous exercises to reproduce the map of Figure 2. First, work out what the local stellar time is for this instance and compute the hour angle of your stars. Second, write a Python function called **`par2horiz()`** to transform parallactic coordinates (hour angle and declination) and the geographical latitude of the observer to azimuth and altitude, using NumPy functions. Use a limiting magnitude of 4.5 m to reduce the number of stars you plot and to scale the plot symbols. Ignore proper motion and precession.

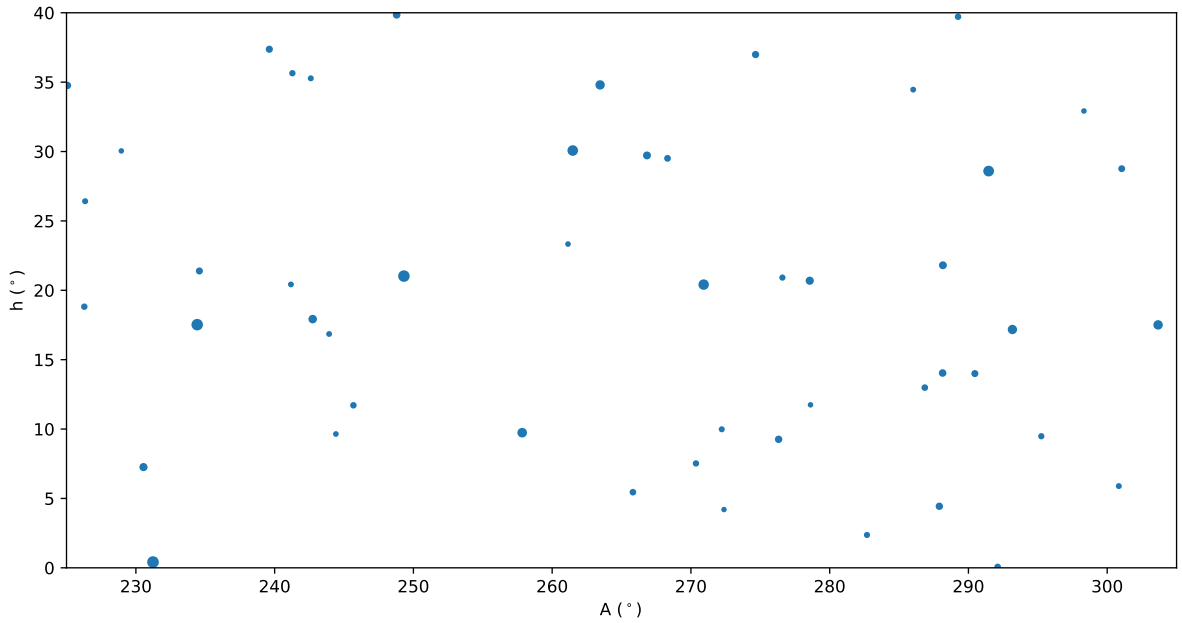

Figure 2: A sky map in horizontal coordinates. Stars near the horizon at sunrise for $\phi_G = 51.178°$ at the vernal equinox 2020. The stars just above $A = 270°$ belong to the constellation Pisces, the (near) square of bright stars is Pegasus/Andromeda.

3 Computer exercise 3: Proper motion and precession

In this exercise, we will recreate Figure 2, but for different epochs and equinoxes.

3.1 Julian day

We will need a linear time variable below. Write the Python function `julianDay()` that takes `year`, `month` and (decimal) `day` as input and returns the Julian day. You can use the `floor()` function from the `math` module. Assume that the Julian calendar is used when the year is 1582 or earlier. Verify that the noons (UT) of the new-year's days of the years -3000, 1000 and 2000 have JDs 625308.0, 2086308.0 and 2451545.0, respectively. Compare your answer to that of Exercise 5a. Compute the number of days between the years 1100 and 1200, and between 1500 and 1600, and explain the difference.

3.2 Proper motion

Write the Python function `properMotion()` that takes the start JD and target JD, as well as (arrays containing) equatorial coordinates and the corresponding proper motions, and returns the positions for the target epoch. Remember to use NumPy functions. Test your function against the position of Vega from Exercise 5a. Replot Figure 2, and use a second `scatter` call to plot the same selection of stars, for the equinox of 2000 but epoch -10 000, in the same plot.

3.3 Precession

Write the Python function `precessHip()` that takes a target JD, as well as (arrays containing) equatorial coordinates, and returns the precessed positions for the target equinox using the method described in Section 2.8.4 of the lecture notes. Since we use the Hipparcos catalogue, the initial JD is a constant and we can use the simplified equations. Remember to use NumPy functions and make sure the right ascensions end up in the correct quadrant. Test your function

against the position of Vega from Exercise 5b. Replot Figure 2, and use a second `scatter` call to plot the same selection of stars, but for the equinox and epoch of the year 1000, in the same plot.

3.4 Does the Bear bathe in the Ocean?

Today, the Big Dipper (Ursa Maior) is not circumpolar when seen from Athens, Greece. The ancient Greek poet Homeros however, claimed that the (Great) Bear never bathes in the ocean. We will make a **polar plot** of the region of 60° around the Celestial North Pole to see whether these claims are true. A polar scatter plot can be set up in Matplotlib using

```
plt.figure(figsize=(7,7))                    # Set plot size to 700x700 (dpi=100)
ax = plt.subplot(111, projection='polar')    # Set up a polar plot
ax.scatter(theta, r, s=sizes)                # Call scatter
ax.set_ylim(0, rMax)                         # Set the radius range
```

Here, the arrays `r` and `theta` contain the polar coordinates of the stars. Plot a selection of your star catalogue for the current epoch up to 60° from the pole. In addition, plot a circle around the pole which contains all circumpolar stars as seen from Athens, Greece (latitude = 38°) and show that the tail of the Bear is not circumpolar. Then, make the same plot for the epoch of 800 BCE and show that the constellation now *is* circumpolar. You can create and draw a (red) circle with radius `radius` in the polar plot by *e.g.*:

```
rCirc = np.ones(101)*radius          # Fill an array with ones and multiply them
thCirc = np.arange(101)/100*m.pi*2   # Fill an array with the range 0 - 2 pi
ax.plot(thCirc, rCirc, 'r')          # Draw a red ('r') circle
```

4 Computer exercise 4: Computing planet positions and magnitudes

4.1 VSOP

The VSOP87 theory [4] describes the heliocentric (or barycentric) ecliptical coordinates (or orbital elements) as periodic terms. There are six different versions; we will use VSOP87D, which provides heliocentric ecliptic spherical coordinates for the equinox of the day. The necessary files can be downloaded from the cited URL by clicking on "FTP". Each file contains the data for a single planet. We will need the `VSOP87D.*` files.

4.1.1 Heliocentric ecliptical coordinates

The function `readVSOP()` reads the periodic terms to compute a heliocentric ecliptical planet position from a VSOP87D file and returns a 2D array for each of the ecliptical longitude, latitude and distance. The arrays have different numbers of rows, and each row i contains the variables

$$[p_i,\, a_i, b_i, c_i]\,,$$

where p corresponds to the variable `power` in Code listing 4.1. Note that you will need to install the Python package/module `fortranformat`.

Listing 4.1: readVSOP.py: reads a VSOP87D file and returns the periodic terms.

```
def readVSOP(fileName):
    inFile = open(fileName,'r')

    import fortranformat as ff
    formatHeader = ff.FortranRecordReader('(40x,I3, 16x,I1,I8)')          # Header
```

```
    formatBody   = ff.FortranRecordReader('(79x,F18.11,F14.11,F20.11)')  # Body

    lonTerms=[]; latTerms=[]; radTerms=[]

    for iBlock in range(3*6):  # 3 variables (l,b,r), up to 6 powers (0-5)
        line = inFile.readline()
        var,power,nTerm = formatHeader.read(line)
        #print(var,power,nTerm)
        if line == '': break  # EoF

        for iLine in range(nTerm):
            line = inFile.readline()
            a,b,c = formatBody.read(line)
            #print(iLine, var,power, a,b,c)

            if var == 1: lonTerms.append([power, a,b,c])  # var=1: ecl. lon.
            if var == 2: latTerms.append([power, a,b,c])  # var=2: ecl. lat.
            if var == 3: radTerms.append([power, a,b,c])  # var=3: distance

    return lonTerms,latTerms,radTerms
```

From these terms, one can compute the heliocentric ecliptical longitude, latitude (in radians) and distance (in AU):

$$L, B, R = \sum_i t_{\mathrm{Jm}}^{p_i}\ a_i\ \cos\left(b_i\ +\ c_i\, t_{\mathrm{Jm}}\right), \tag{1}$$

where t_{Jm} is the time since 2000, expressed in Julian millennia:

$$t_{\mathrm{Jm}} = \frac{\mathrm{JDE} - 2451545}{365250}. \tag{2}$$

Write a Python function **`computeLBR()`** that takes the JD(E) and **`lonTerms`**, **`latTerms`** and **`radTerms`** as input, and returns L, B and R. Verify that on 1 Jan -1000, $L, B, R \approx 1.5925\,\mathrm{rad}$, $4.15 \times 10^{-7}\,\mathrm{rad}$ and 0.98608 AU, respectively, for the Earth, and 0.5462 rad, $-0.01612\,\mathrm{rad}$ and 5.0967 AU for Jupiter.

4.1.2 Geocentric ecliptical coordinates

To convert the heliocentric coordinates to geocentric ones, we compute the heliocentric positions of both the planet of interest and of the Earth and subtract the second from the first. This is easier when using rectangular coordinates:

$$x = R \cos B \cos L \tag{3}$$
$$y = R \cos B \sin L \tag{4}$$
$$z = R \sin B \tag{5}$$

After subtraction, we need to convert the resulting geocentric rectangular ecliptic coordinates (x, y, z) back to spherical, using the inverse of Eqs. 3–5.

Write a Python function **`hc2gc()`** that takes the heliocentric coordinates (L, B, R) of a planet and of the Earth as input parameters, and returns the geocentric spherical ecliptical coordinates (l, b, r) of the planet. On 1 Jan -1000, Jupiter's l, b, r should be $\sim 0.363\,\mathrm{rad}$, $-0.018\,\mathrm{rad}$ and 4.68 AU.

4.1.3 Conversion of ecliptical to equatorial coordinates

In order to make *local* sky maps, we need to be able to convert (geocentric) ecliptical coordinates to equatorial ones (and thence to the azimuthal system). For historical calculations, we need to use the contemporary value of the obliquity of the ecliptic.

1. Write a Python function `obliquity()` that computes ε from the JD. Show that the obliquity in the years 2000, 1000 and -3000 is (approximately) equal to 23.439°, 23.569° and 24.02° respectively.

2. Write a Python function `ecl2eq()` that converts a geocentric planet position from ecliptical to equatorial coordinates, where the obliquity is one of the input parameters. Show that on 1 Jan -1000, Jupiter's $\alpha, \delta \approx 0.341, 0.128$ rad.

4.2 Magnitudes

In order to compute the visual magnitude of a planet, we need its *phase angle*[1] ϕ:

$$\phi = \arccos\left(\frac{R^2 + r^2 - r_\odot^2}{2\,R\,r}\right), \tag{6}$$

where $\phi \in [0°, 180°] \geq 0°$, R and r are the heliocentric and geocentric distances of the planet, respectively, and $r_\odot$ is the heliocentric distance of the Earth.

The visual magnitude of a planet is then given by

$$V = 5\ \log_{10}(R\,r)\ +\ a_0\ +\ a_1\,\phi\ +\ a_2\,\phi^2\ +\ a_3\,\phi^3, \tag{7}$$

with ϕ in **degrees**. The a_i are provided by

	Mer.	**Ven.1**	**Ven.2**	**Mars**	**Jup.**	**Sat.**	**Ur.**	**Nep.**
a_0	−0.60	−4.47	+0.98	−1.52	−9.40	−8.88	−7.19	−6.87
a_1	0.0498	0.0103	−0.0102	0.016	0.005	0.044	0.002	—
a_2	-4.88×10^{-4}	5.7×10^{-5}	—	—	—	—	—	—
a_3	3.02×10^{-6}	1.3×10^{-7}	—	—	—	—	—	—

The results for Mercury are valid in the range $2° < \phi < 170°$. For Venus, two expressions are available — the first is used for $2.2° < \phi < 163.6°$ and the second for $163.6° < \phi < 170.2°$. Outside these ranges, Venus is very close to the Sun, and the expressions are not valid.

This method has been used by the Astronomical Almanac since 2007 [5] (taking into account the errata [6]).

Write a Python function `magnPlanet()` that takes the planet number (1=Mercury, 8=Neptune), the distances to the Sun and the Earth and the distance between the Earth and the Sun as input parameters, and returns the planet's magnitude. Show that on 1 Jan -1000, Jupiter's magnitude is about −2.46.

4.2.1 The magnitude of Saturn's rings

For Saturn, we need to add the brightness of its rings to the magnitude of the planet. A simplified method[2] starts by computing the inclination of the ring i and the longitude of the ascending node Ω

$$i = 0.49; \tag{8}$$

$$\Omega = 2.96 + 0.024\,t_{\mathrm{Jc}}, \tag{9}$$

where t_{Jc} is the time in Julian centuries since 2000 and where i and Ω are expressed in radians. Next, compute the sine of the latitude B of the Earth, as seen from Saturn and with respect to

[1] The phase angle is the angle Sun–planet–Earth, *i.e.* the angular separation in the sky between the Sun and the Earth as seen from the planet.

[2] The mean and maximum absolute deviations from a more accurate algorithm for 10^5 random trials over the last 5 ka were 0.014 m and 0.041 m, respectively.

the plane of the rings:[3]

$$\sin B = \sin i \, \cos\beta \, \sin(\lambda - \Omega) \; - \; \cos i \, \sin\beta, \tag{10}$$

where λ and β are the geocentric ecliptical longitude and latitude of Saturn, respectively. The visual magnitude of Saturn's rings can then be approximated by

$$V_{\rm Sr} \; = \; -2.60\,|\sin B| \; + \; 1.25\,\sin^2 B. \tag{11}$$

This term should be *added* to the magnitude of Saturn computed above.

Write a Python function **`magnSatRing()`** that takes the JD and the geocentric ecliptical coordinates of Saturn and returns the magnitude of Saturn's rings. You can use Eq. 2 as inspiration to compute $t_{\rm Jc}$. Show that on 1 Jan -1000, Saturn's magnitude is ≈ -0.28.

4.3 Putting it all together

Reproduce the ecliptical maps in Figure 3.9 from the lecture notes (both in a single plot, if you like). Note that Jupiter moves $\sim 30^\circ$ per *year*. Of the grey lines in that Figure, only plot the ecliptic. Ensure that Jupiter's disc has the size that reflects its magnitude.

5 Computer exercise 5: Computing the position of the Moon

Computing a precise Moon position is more complex than computing that of the planets, due to *e.g.* the perturbations by the planets and the fact that the Earth is not a sphere. In addition, the Moon is so close to the Earth that different observers on different continents may see a different position of the Moon with respect to the background stars (*parallax*), which must be taken into account. Since this parallax changes with the Earth's rotation, we will also need to know the *exact* orientation of the Earth in its slowly decelerating rotation, for which we will need to compute the local sidereal time and use the quantity known as ΔT. We will ignore a slight wobble of the Earth's rotation axis called *nutation* (for the Moon and other celestial objects), since it is small and only affects the position of an object with respect to the horizon (or equator), not to other celestial objects.

5.1 ELP 2000-82/85

The lunar theories ELP 2000-82 and ELP 2000-85 were published in 1983 and 1988, respectively, by researchers from the same institute that brought forth the VSOP87 paper [7; 8]. The ELP theory is an elaborate fit, similar to the VSOP and has roughly the same number of periodic terms as the VSOP87 theory for all planets together. The data files can be downloaded, together with Fortran code to compute the position [7]. We will use an abridged version published by Meeus [9], but with arguments in radians rather than degrees.

The lab-class webpage provides the file `moonposMeeus.csv` with most of the data, which can be read with the Python function `readELP82bData()` below.

```
def readELP82bData(inFile):
    """Read the periodic terms for the ELP82B theory, selected by Meeus and
    return them in two arrays: one for longitude and distance, and one for
    latitude.
    """

    # Longitude and radius (6 columns: 4 args, 2 coefs):
    lrTerms = np.genfromtxt(inFile, delimiter=',', skip_header=1,  max_rows=60)
```

[3] If B is positive, we see the northern side of the rings from Earth, while $B = 0$ indicates a *ring-plane crossing*.

	$t^0_{\rm Jc}$	$t^1_{\rm Jc}$	$t^2_{\rm Jc}$	$t^3_{\rm Jc}$	$t^4_{\rm Jc}$
$\lambda_{\rm m}$	3.8103408236	8399.7091116339958	-2.755176757e-5	3.239043e-8	-2.6771e-10
D	5.1984665298	7771.377144834	-3.2845e-5	3.197347e-8	-1.5436512e-10
$M_\odot$	6.240060127	628.301955167	-2.681e-6	7.1267017e-10	
$M_{\leftmoon}$	2.355555637	8328.691424759	1.52566e-4	2.5041e-7	-1.18633e-9
F	1.627905158	8433.466158061	-6.3773e-5	-4.94988e-9	2.02167e-11
E	1	-0.002516	-0.0000074		
A_1	2.090032	2.301199			
A_2	0.926595	8364.7398477			
A_3	5.4707345	8399.6847253			

Table 1: Coefficients for the lunar-orbit arguments.

```
9
10      # Latitude (5 columns: 4 args, 1 coef):
11      bTerms  = np.genfromtxt(inFile, delimiter=',', skip_header=61, max_rows=60)
12
13      return lrTerms,bTerms
```

After reading the file, we will compute a number of quantities, which are all polynomials of the time in Julian centuries since 2000 ($t_{\rm Jc}$) with the coefficients shown in Table 1, where the first coefficient is the constant, the second is for the $t_{\rm Jc}$ term, etc. These are the Moon's mean longitude $\lambda_{\rm m}$, mean elongation D, mean anomaly $M_{\leftmoon}$ and argument of latitude F. $M_\odot$ is the Sun's mean anomaly and E is a correction factor to take into account the decreasing eccentricity of the Earth's orbit. A_1–A_3 are used to correct for perturbations by the planets. All but E are angles, expressed in radians and should be brought between 0 and 2π (and converted to degrees if desired) before you print them. We will put the four values called the Delauney arguments in an array: $\alpha = [D, M_\odot, M_{\leftmoon}, F]$.

Then for each line i in the ELP file, and separately for the variables λ and R on the one hand, and β on the other, compute the four arguments γ for the ELP theory using the four integer coefficients for the four Delauney arguments:

$$\gamma_{\rm i} = \sum_{j=0}^{3} b_{i,j} \cdot \alpha_j. \tag{12}$$

Next, for the longitude, latitude and distance, sum the periodic terms as follows:

$$\lambda, \beta = \sum_{i}^{N_{\rm terms}} \sin\gamma_i \cdot C_i \cdot E^{|b_{i,1}|}; \tag{13}$$

$$R = \sum_{i}^{N_{\rm terms}} \cos\gamma_i \cdot C_i \cdot E^{|b_{i,1}|}. \tag{14}$$

Note that the integers $b_{i,j}$ are identical for the longitude and distance and provided in the first four columns of the Python variable `lrTerms`, while the coefficients C_i for λ and R are in the fifth and sixth column respectively. As a consequence, the arguments γ_i are also identical for λ and R. For β, the integer arguments b and coefficients C are in columns 0-3 and 4 of `bTerms`, respectively.

To take into account perturbations by the planets, we compute the corrections (in radians)

$$\Delta\lambda = 6.908 \times 10^{-5} \sin A_1 + 3.4243 \times 10^{-5} \sin(\lambda_m - F) + 5.55 \times 10^{-6} \sin(A_2); \quad (15)$$

$$\begin{aligned}\Delta\beta = & -3.9008 \times 10^{-5} \sin(\lambda_m) + 6.667 \times 10^{-6} \sin(A_3) + 3.0543 \times 10^{-6} \sin(A_1 - F) \\ & +3.0543 \times 10^{-6} \sin(A_1 + F) + 2.2166 \times 10^{-6} \sin(\lambda_m - M_{\leftmoon}) \\ & -2.007 \times 10^{-6} \sin(\lambda_m + M_{\leftmoon}).\end{aligned} \quad (16)$$

Finally, we add up all components, including the mean values:

$$\lambda = (\lambda + \lambda_m + \Delta\lambda) \mod 2\pi; \quad (17)$$

$$\beta = \beta + \Delta\beta; \quad (18)$$

$$R = R + 385000.56. \quad (19)$$

The geocentric ecliptical coordinates λ and β are expressed in radians, the distance R in km.

Write a Python function called `moonLBR()` that takes the JD(E)[4] and the arrays `lrTerms` and `bTerms` as input parameters and returns the geocentric ecliptical coordinates of the Moon.

5.2 Greenwich and local sidereal time

To compute the Greenwich mean[5] sidereal time in radians for a given instance, we can use the polynomial fit [5, Eq. 6.66]:

$$\begin{aligned}\theta_0 = & 4.89496121088131 + 6.30038809894828323 \cdot t_{Jd} + 5.05711849 \times 10^{-15} \cdot t_{Jd}^2 \\ & - 4.378 \times 10^{-28} \cdot t_{Jd}^3 - 8.1601415 \times 10^{-29} \cdot t_{Jd}^4 - 2.7445 \times 10^{-36} \cdot t_{Jd}^5,\end{aligned} \quad (20)$$

where t_{Jd} is the time since 2000 in Julian days

$$t_{Jd} = JD - 2\,451\,545, \quad (21)$$

and you should use the JD to compute t_{Jd}, not the JDE.

To compute the *local* (mean) sidereal time (LST), we add the geographical longitude of the observer λ_{obs}, with $\lambda_{obs} > 0$ if east of Greenwich:

$$\theta = \theta_0 + \lambda_{obs}. \quad (22)$$

Before printing the variables θ_0 or θ, you should ensure their values lie between 0 and 2π.

Write a Python function `localSiderealTime()` that takes the JD and geographical longitude as input parameters and returns the local mean sidereal time in radians (between 0 and 2π).

5.3 The decelerating Earth rotation and ΔT

The Earth's rotation is slowly decelerating, mainly due to the tidal effects from the Moon, and at an unpredictable rate. For the past, historical observations are used to determine the difference between the *true* orientation of the Earth and the orientation the Earth would have if each day had been 86 400 s long and the phase was that of $\sim$ 1820. This difference is called ΔT and is expressed in seconds.

In this document, we will assume a constant rate of lengthening of the day of 1.8 ms/century, and that $\Delta T_0 = 12$ s in the year 1820. We are not interested in the length of a day at any time,

[4] See Sect. 5.3 for the JDE. For now, we will call `moonLBR()` with the JD, but this should be replaced with the JDE later on.

[5] *i.e.*, not corrected for nutation

but at the *cumulative* effect this amounts to over the centuries or millennia. The constant is therefore in fact a *deceleration* factor of 1.8 ms/day/century and depends on the time since 1820 *squared*. When expressed in seconds, this can be written as:

$$\begin{aligned}\Delta T &\approx \Delta T_0 + \frac{1}{2}\, a\, (t - t_{1820})^2 \\ &\approx 12 + \frac{1}{2} \cdot \frac{1.8 \times 10^{-3}\,\mathrm{s}}{86400\,\mathrm{s} \cdot (36525 \cdot 86400\,\mathrm{s})} \cdot ((\mathrm{JD} - \mathrm{JD1820}) \cdot 86400\,\mathrm{s})^2 \\ &\approx 12 + \frac{1}{2}\, \frac{1.8 \times 10^{-3}\,\mathrm{s}}{36525}\, (\mathrm{JD} - \mathrm{JD1820})^2. \end{aligned} \tag{23}$$

We will use ΔT to correct the Julian day to the *ephemeris* JD, or JDE:

$$\mathrm{JDE} = \mathrm{JD} + \frac{\Delta T}{86400}. \tag{24}$$

Write a Python function `DeltaT()` that takes the JD as input and returns ΔT in seconds. We will use this function from now on to compute both the JD and JDE for any instance we are interested in. When computing the dynamics of the Solar system, *e.g.* the position of Sun, Moon and planets, we will use the JDE instead of the JD. Hence, the (time associated with the) JDE is also referred to as "dynamical time".

Remember that the JD is in fact a "local" time which applies only to planet Earth since it takes into account the orientation of the Earth in its axis rotation. JDE concerns the time "out there" in the Solar system. When we compute the position of solar-system objects for a historical date at, say, noon in Greenwich, we know exactly the *number of days* (of variable length) that passed since that date occurred, but not the *amount of time*. Using ΔT, we can compute the latter and compute how many *seconds* ago this happened.

Note that ΔT was roughly zero around 1820, is currently positive and was also positive in the (distant) past — $\sim 10580\,\mathrm{s}$ ($\sim 3\,\mathrm{h}$) around the year 0. Also note that our assumption for the constant lengthening of the day is reasonable, but not perfect. In reality the changes are quite irregular, and since ΔT can be measured to some accuracy, one could use those values. See *e.g.* [10] for a graphical display of the measurements, a machine-readable table, and references.

5.4 Diurnal parallax and topocentric positions

The position for the Moon (and planets) that we have computed so far are *geocentric*, *i.e.* for an imaginary observer located in the centre of the Earth. Since most observers are on the Earth's *surface*, we need to compute the *topocentric* positions of these objects for the exact location of the observer.

The difference in apparent sky position of a given celestial object between the centre of the Earth and an observer on the surface (or between two observers at different locations) is known as the *parallax*. This effect is of course stronger for nearby objects like the Moon and is practically zero for nearly infinitely remote stars. For planets, the effects can usually be ignored, except for *e.g.* the accurate calculation of a Venus transit.

Since the parallax depends on the geometry between geocentre, observer location and celestial object, and the Earth rotates about its axis, the effect of the parallax is variable during the day. It is therefore known as the *diurnal parallax*. Due to the parallax, an object appears *lower* in the sky (closer to the horizon) than expected based on calculations using the geocentric position. The parallax in azimuth is very small, and would be zero if the Earth were a sphere.

We will follow the treatment of Meeus [9] in his Chapters 11 and 40 to convert geocentric positions to the topocentric system in ecliptical coordinates taking into account the flattening of the Earth and the elevation of the observer above sea level.

For the equatorial and polar radii of the Earth, we use

$$R_{\oplus,\mathrm{eq}} = 6378136.6; \tag{25}$$

$$\frac{R_{\oplus,\mathrm{pol}}}{R_{\oplus,\mathrm{eq}}} = 0.996647189335. \tag{26}$$

Then

$$\tan u = \frac{R_{\oplus,\mathrm{pol}}}{R_{\oplus,\mathrm{eq}}} \tan \varphi_{\mathrm{obs}}; \tag{27}$$

$$\rho \sin \varphi' = \frac{R_{\oplus,\mathrm{pol}}}{R_{\oplus,\mathrm{eq}}} \sin u + \frac{h_{\mathrm{obs}}}{R_{\oplus,\mathrm{eq}}} \sin \varphi_{\mathrm{obs}}; \tag{28}$$

$$\rho \cos \varphi' = \cos u + \frac{h_{\mathrm{obs}}}{R_{\oplus,\mathrm{eq}}} \cos \varphi_{\mathrm{obs}}, \tag{29}$$

where φ_{obs} is the geographical latitude of the observer, and h_{obs} her altitude above sea level in the same units as $R_{\oplus,\mathrm{eq}}$. The variable ρ is the geocentric distance of the observer expressed in equatorial radii, taking into account the non-sphericity of the Earth, and φ' would be equal to φ_{obs} if the Earth were a sphere.

We compute the *horizontal parallax*, *i.e.* the maximum possible difference in altitude, of an object with distance d as

$$\sin \pi = \frac{\sin\left(R_{\oplus,\mathrm{eq}}/\mathrm{AU}\right)}{d/\mathrm{AU}}. \tag{30}$$

Then compute

$$N = \cos \lambda \cos \beta - \rho \cos \varphi' \sin \pi \cos \theta, \tag{31}$$

where λ, β are the ecliptical coordinates and θ is the local sidereal time.

The topocentric ecliptical longitude and latitude and the topocentric semi-diameter of the object can then be computed from:

$$\tan \lambda' = \frac{\sin \lambda \cos \beta - \sin \pi \left(\rho \sin \varphi' \sin \varepsilon + \rho \cos \varphi' \cos \varepsilon \sin \theta\right)}{N}; \tag{32}$$

$$\tan \beta' = \frac{\cos \lambda' \left(\sin \beta - \sin \pi \left(\rho \sin \varphi' \cos \varepsilon - \rho \cos \varphi' \sin \varepsilon \sin \theta\right)\right)}{N}; \tag{33}$$

$$\sin r' = \frac{\cos \lambda' \cos \beta' \sin r}{N}. \tag{34}$$

In these expressions, a $'$ denotes the topocentric equivalent of a geocentric variable, ε is the obliquity of the ecliptic and r is the *apparent* radius of the object, as it appears in the sky (*i.e.*, an angle).

5.5 Nutation

Astronomical nutation is the slight wobble on the Earth's axis about a mean direction described by precession, on a range of short timescales. The main causes are the perturbations of the Sun and Moon on the non-spherical Earth, and the strongest period is ∼18.6 years, the timescale on which the orientation of the lunar orbit changes.

The effect of nutation is on the order of 18" at most, and only changes the orientation of the equatorial (and hence horizontal) coordinate systems with respect to the ecliptic, not the

positions between celestial bodies. Since a difference in position of 18" is not much when compared to the horizon[6] or variations in atmospheric refraction due to the weather, we will ignore nutation here. More information on nutation can be found on Wikipedia [11].

5.6 Spica and the Moon in the year 98

On 11 January of the year 98, the Moon was observed close to Spica (+0.98 m), as seen from Rome. Compute the position, distance and diameter of the Moon for a number of instances (*e.g.* every hour, and then refining it within the hour of interest) for this date and location, and compare the apparent distance between the two bodies to the diameter of the Moon. Devise a parameter that is an indication of the distance between the lunar limb and Spica, and that clearly shows at any time whether an occultation occurs or not. At what time (UT) was the closest approach? How close did the two bodies get? Did the Moon occult Spica, and if so, what were the start and end times? The geographical longitude and latitude of Rome are 12.4667°E and 41.8833°N.[7]

References

[1] **Perryman, M.A.C., Lindegren, L. *et al.*** *The Hipparcos Catalogue. A&A* 500:501, 1997.

[2] **Oliphant, T.** *NumPy: A guide to NumPy.* USA: Trelgol Publishing, 2006–. URL http://www.numpy.org. [Online; accessed 2019-01-26].

[3] **Hunter, J.D.** *Matplotlib: A 2D graphics environment. Computing In Science & Engineering* 9(3):90, 2007. URL https://matplotlib.org.

[4] **Bretagnon, P. & Francou, G.** *Planetary theories in rectangular and spherical variables - VSOP 87 solutions. A&A* 202:309, 1988. URL http://cdsarc.u-strasbg.fr/viz-bin/Cat?cat=VI/81.

[5] **Urban, S.E. & Seidelmann, P.K.** *Explanatory Supplement to the Astronomical Almanac (3rd Edition).* University Science Books, 2012.

[6] ——. *Errata to the Explanatory Supplement to the Astronomical Almanac (3rd Edition)*, 2018. URL https://aa.usno.navy.mil/publications/docs/exp_supp_errata.pdf.

[7] **Chapront-Touze, M. & Chapront, J.** *The lunar ephemeris ELP 2000. A&A* 124:50, 1983. URL ftp://ftp.imcce.fr/pub/ephem/moon/elp82b/.

[8] ——. *ELP 2000-85 - A semi-analytical lunar ephemeris adequate for historical times. A&A* 190:342, 1988.

[9] **Meeus, J.** *Astronomical algorithms.* 1998.

[10] **Extrapolation of Delta T**. *Extrapolation of* ΔT, 2015. URL http://hemel.waarnemen.com/Computing/deltat.html.

[11] **Wikipedia**. *Astronomical nutation — Wikipedia, The Free Encyclopedia*, 2019. URL https://en.wikipedia.org/w/index.php?title=Astronomical_nutation. [Online; accessed 24-May-2019].

[6] If the horizon is ∼5 km away, 18" corresponds to ∼44 cm of difference in elevation.

[7] The default convention is to treat longitudes east of Greenwich and latitudes north of the equator as positive.