EATS PL Report - Satellites

Template by André Moitinho

Student number: 49807

Name: Ricardo Matoza Pires

Instructions

- Fill in your name and number above
- This template provides blocks of instructions that guide you through the activity and report. Put your codes and comments below the corresponding blocks.
- Add as many code and markdown cells as necessary.
- The blue background colour in the markdown cells (like this one) is reserved for the instruction blocks. Use white cells for your comments.
- Write a clear report:
  - Provide justifications and comments.
  - Figures and labels (legends, axes, etc) should be easy to read.
  - Tables should be neatly formatted and include labels.

To submit your report:

You will need to have the nbconvert[webpdf] package installed for saving to pdf.
Anaconda: conda install -c conda-forge nbconvert-webpdf
General: pip install nbconvert[webpdf]

- Clean up and make sure all works well: In the "Kernel" menu select "Restart & Run All" (or use the "fast forward/double arrow" icon). Wait until the process is completed
- Save to a pdf file:
  - JupyterNotebook users: In the "File" menu -> "Download As" -> "**PDF via HTML (.html)**".
  - JupyterLab users: In the "File" menu -> "Save and Export Notebook As" -> "**Webpdf**".
- Check that the PDF file is fine (no truncated lines, nothing missing, etc)
- Name the file: NumberFirstnameLastname.pdf* (e.g. 10101ClaudeShannon.pdf)
- Upload the pdf file to the "PL Report - Satellites" assignement area.

Tips

- Run a cell - keyboard shortcut: shift-enter
- To create "markdown" cells like this one, For comments, not for code: create a new cell. In the menu above to the right of the "fast-forward" icon there's a dropdown button with the default value "Code". Change to "Markdown".
- To display an image from a file on disk (e.g. image.png): In a **markdown cell** you can use *`![some text](image.png)`*, or simply drag and drop!

Setup

Put all your imports (e.g. ephem, numpy, matplotlib, etc) in the cell bellow

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
import ephem
import datetime
import csv
```

PART 1 - Orbital elements basics

In this first exercise, you'll import a *Three Line Element* (TLE) for an Earth satellite, propagate its orbit with SGP4 (Simplified perturbations models) implemented in *ephem* and plot the results. In the process you will see which methods and attributes are provided by *ephem* for dealing with Earth satellites and their observation.

Set observer's coordinates and time

Start by setting the observer's coordinates and time in an *ephem Observer* object. Say which place and time you chose.

In [ ]:
```
## Set-up the observer's
obs = ephem.Observer()
## Its coordinates (latitude and longitude) for Lisbon
obs.lat = np.deg2rad(38.717)
obs.long = np.deg2rad(9.133)
## Define a date/time of observation (8th of November 2023 at midnight)
obs.date = datetime.datetime(2023,11,8,0)
```

What attributes and methods does the *Observer* object provide? Use the "help" command and show the output

In [ ]:
```
## Print observer's attriburtes and methods
help(obs)
```

```
Help on Observer in module ephem object:

class Observer(_libastro.Observer)
 |  A location on earth for which positions are to be computed.
 |
 |  An `Observer` instance allows you to compute the positions of
 |  celestial bodies as seen from a particular latitude and longitude on
 |  the Earth's surface.  The constructor takes no parameters; instead,
 |  set its attributes once you have created it.  Defaults:
 |
 |  `date` - the moment the `Observer` is created
 |  `lat` - zero latitude
 |  `lon` - zero longitude
 |  `elevation` - 0 meters above sea level
 |  `horizon` - 0 degrees
 |  `epoch` - J2000
 |  `temp` - 15 degrees Celsius
 |  `pressure` - 1010 mBar
 |
 |  Method resolution order:
 |      Observer
 |      _libastro.Observer
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __copy__ = copy(self)
 |
 |  __repr__(self)
 |      Return a useful textual representation of this Observer.
 |
 |  compute_pressure(self)
 |      Set the atmospheric pressure for the current elevation.
 |
 |  copy(self)
 |
 |  disallow_circumpolar(self, declination)
 |      Raise an exception if the given declination is circumpolar.
 |
 |      Raises NeverUpError if an object at the given declination is
 |      always below this Observer's horizon, or AlwaysUpError if such
 |      an object would always be above the horizon.
 |
 |  next_antitransit(self, body, start=None)
 |      Find the next passage of a body across the anti-meridian.
 |
 |  next_pass(self, body, singlepass=True)
 |      Return the next rising, culmination, and setting of a satellite.
 |
 |      If singlepass is True, return next consecutive set of
 |      ``(rising, culmination, setting)``.
 |
 |      If singlepass is False, return
 |      ``(next_rising, next_culmination, next_setting)``.
 |
 |  next_rising(self, body, start=None, use_center=False)
 |      Search for the given body's next rising, returning its date.
 |
 |      The search starts at the `date` of this `Observer` and is limited to
 |      the single circuit of the sky, from antitransit to antitransit, that
```

```
 |      the `body` was in the middle of describing at that date and time.
 |      If the body did not, in fact, cross the horizon in the direction you
 |      are asking about during that particular circuit, then the search
 |      must raise a `CircumpolarError` exception like `NeverUpError` or
 |      `AlwaysUpError` instead of returning a date.
 |
 |  next_setting(self, body, start=None, use_center=False)
 |      Search for the given body's next setting, returning its date.
 |
 |      The search starts at the `date` of this `Observer` and is limited to
 |      the single circuit of the sky, from antitransit to antitransit, that
 |      the `body` was in the middle of describing at that date and time.
 |      If the body did not, in fact, cross the horizon in the direction you
 |      are asking about during that particular circuit, then the search
 |      must raise a `CircumpolarError` exception like `NeverUpError` or
 |      `AlwaysUpError` instead of returning a date.
 |
 |  next_transit(self, body, start=None)
 |      Find the next passage of a body across the meridian.
 |
 |  previous_antitransit(self, body, start=None)
 |      Find the previous passage of a body across the anti-meridian.
 |
 |  previous_rising(self, body, start=None, use_center=False)
 |      Search for the given body's previous rising, returning its date.
 |
 |      The search starts at the `date` of this `Observer` and is limited to
 |      the single circuit of the sky, from antitransit to antitransit, that
 |      the `body` was in the middle of describing at that date and time.
 |      If the body did not, in fact, cross the horizon in the direction you
 |      are asking about during that particular circuit, then the search
 |      must raise a `CircumpolarError` exception like `NeverUpError` or
 |      `AlwaysUpError` instead of returning a date.
 |
 |  previous_setting(self, body, start=None, use_center=False)
 |      Search for the given body's previous setting, returning its date.
 |
 |      The search starts at the `date` of this `Observer` and is limited to
 |      the single circuit of the sky, from antitransit to antitransit, that
 |      the `body` was in the middle of describing at that date and time.
 |      If the body did not, in fact, cross the horizon in the direction you
 |      are asking about during that particular circuit, then the search
 |      must raise a `CircumpolarError` exception like `NeverUpError` or
 |      `AlwaysUpError` instead of returning a date.
 |
 |  previous_transit(self, body, start=None)
 |      Find the previous passage of a body across the meridian.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  elev
 |      Elevation above sea level in meters
 |
 |  name
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _libastro.Observer:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  radec_of(...)
 |      compute the right ascension and declination of a point identified by its azimuth and altitude
 |
 |  sidereal_time(...)
 |      compute the local sidereal time for this location and time
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from _libastro.Observer:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _libastro.Observer:
 |
 |  date
 |      Date
 |
 |  elevation
 |      Elevation above sea level in meters
 |
```

```
 |  epoch
 |      Precession epoch
 |
 |  horizon
 |      The angle above (+) or below (-) the horizon at which an object should be considered at the moment of ri
sing or setting as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  lat
 |      Latitude north of the Equator as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  lon
 |      Longitude east of Greenwich as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  long
 |      Longitude east of Greenwich as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  pressure
 |      atmospheric pressure in millibar
 |
 |  temp
 |      alias for 'temperature' attribute
 |
 |  temperature
 |      atmospheric temperature in degrees Celsius
```

Read a TLE

From the TLE file provided for this exercise, select a satellite and load its orbital elements into an ephem *EarthSatellite* object (readtle).

Para o propósito seguinte do exercício, em que vamos ver o "caminho" de um satélite ao longo de um dia, vamos desde já escolher um satélite geoestácionário, para vermos um período completo do seu movimento.

In [ ]:
```
## Select GEO satellite
l1 = '0 INTELSAT 18'
l2 = '1 37834U 11056A   23310.53660828  .00000059  00000-0  00000-0 0  9999'
l3 = '2 37834   0.0164 273.0485 0002003 311.6182 194.0234  1.00273532 44244'
## Load its orbital elements
orbitElement = ephem.readtle(l1,l2,l3)
```

What attributes and methods does the *EarthSatellite* object provide? Use the "help" command and show the output

In [ ]:
```
## Print satellite's attributes and methods
help(orbitElement)
```

```
Help on EarthSatellite object:

class EarthSatellite(Body)
 |  A satellite in orbit around the Earth, usually built by passing the text of a TLE entry to the `ephem.readtl
e()` routine. You can read and write its orbital parameters through the following attributes:
 |
 |  _ap -- argument of perigee at epoch (degrees)
 |  _decay -- orbit decay rate (revolutions per day-squared)
 |  _drag -- object drag coefficient (per earth radius)
 |  _e -- eccentricity
 |  _epoch -- reference epoch (mjd)
 |  _inc -- inclination (degrees)
 |  _M -- mean anomaly (degrees from perigee at epoch)
 |  _n -- mean motion (revolutions per day)
 |  _orbit -- integer orbit number of epoch
 |  _raan -- right ascension of ascending node (degrees)
 |
 |  Method resolution order:
 |      EarthSatellite
 |      Body
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  M
 |      mean anomaly (degrees from perigee at epoch)
 |
```

```
 |  ap
 |      argument of perigee at epoch (degrees)
 |
 |  catalog_number
 |      catalog number from TLE file
 |
 |  decay
 |      orbit decay rate (revolutions per day-squared)
 |
 |  drag
 |      object drag coefficient (per earth radius)
 |
 |  e
 |      eccentricity
 |
 |  eclipsed
 |      whether satellite is in earth's shadow
 |
 |  elevation
 |      height above sea level in meters
 |
 |  epoch
 |      reference epoch (mjd)
 |
 |  inc
 |      orbit inclination (degrees)
 |
 |  n
 |      mean motion (revolutions per day)
 |
 |  name
 |      object name
 |
 |  orbit
 |      integer orbit number of epoch
 |
 |  raan
 |      right ascension of ascending node (degrees)
 |
 |  range
 |      distance from observer to satellite in meters
 |
 |  range_velocity
 |      range rate of change in meters per second
 |
 |  sublat
 |      latitude beneath satellite as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  sublong
 |      longitude beneath satellite as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Body:
 |
 |  __copy__(...)
 |      Return a new copy of this body
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  compute(...)
 |      compute the location of the body for the given date or Observer, or for the current time if no date is s
upplied
 |
 |  copy(...)
 |      Return a new copy of this body
 |
 |  parallactic_angle(...)
 |      return the parallactic angle to the body; an Observer must have been provided to the most recent compute
() call, because a parallactic angle is always measured with respect to a specfic observer
 |
 |  writedb(...)
 |      return a string representation of the body appropriate for inclusion in an ephem database file
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from Body:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Body:
 |
```

```
 |  a_dec
 |      astrometric geocentric declination as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  a_epoch
 |      date giving the equinox of the body's astrometric right ascension and declination
 |
 |  a_ra
 |      astrometric geocentric right ascension as a float giving radians, or a string giving hours:minutes:secon
ds
 |
 |  alt
 |      altitude as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  az
 |      azimuth as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  circumpolar
 |      whether object remains above the horizon this day
 |
 |  dec
 |      declination as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  elong
 |      elongation as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  g_dec
 |      apparent geocentric declination as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  g_ra
 |      apparent geocentric right ascension as a float giving radians, or a string giving hours:minutes:seconds
 |
 |  ha
 |      hour angle as a float giving radians, or a string giving hours:minutes:seconds
 |
 |  mag
 |      magnitude
 |
 |  neverup
 |      whether object never rises above the horizon this day
 |
 |  ra
 |      right ascension as a float giving radians, or a string giving hours:minutes:seconds
 |
 |  radius
 |      visual radius as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  rise_az
 |      azimuth at which the body rises as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  rise_time
 |      rise time
 |
 |  set_az
 |      azimuth at which the body sets as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  set_time
 |      set time
 |
 |  size
 |      visual size in arcseconds
 |
 |  transit_alt
 |      transit altitude as a float giving radians, or a string giving degrees:minutes:seconds
 |
 |  transit_time
 |      transit time
```

List the attributes and methods in the *EarthSatellite* by directly inspecting its contents with the *dir* instruction.

In [ ]:
```
## Creates a list of the satellite's attributes and methods
dir(orbitElement)
```

Out[ ]:
```
['M',
 '_M',
 '__class__',
 '__copy__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '_ap',
 '_decay',
 '_drag',
 '_e',
 '_epoch',
 '_inc',
 '_n',
 '_orbit',
 '_raan',
 'a_dec',
 'a_epoch',
 'a_ra',
 'alt',
 'ap',
 'az',
 'catalog_number',
 'circumpolar',
 'compute',
 'copy',
 'dec',
 'decay',
 'drag',
 'e',
 'eclipsed',
 'elevation',
 'elong',
 'epoch',
 'g_dec',
 'g_ra',
 'ha',
 'inc',
 'mag',
 'n',
 'name',
 'neverup',
 'orbit',
 'parallactic_angle',
 'ra',
 'raan',
 'radius',
 'range',
 'range_velocity',
 'rise_az',
 'rise_time',
 'set_az',
 'set_time',
 'size',
 'sublat',
 'sublong',
 'transit_alt',
 'transit_time',
 'writedb']
```

Orbit integration.

Our goal is to visualise the satellite's positions (alititude and azimuth) for an array of times.

1. Create an array of times spanning a whole day
2. Compute the orbits for each time in the array

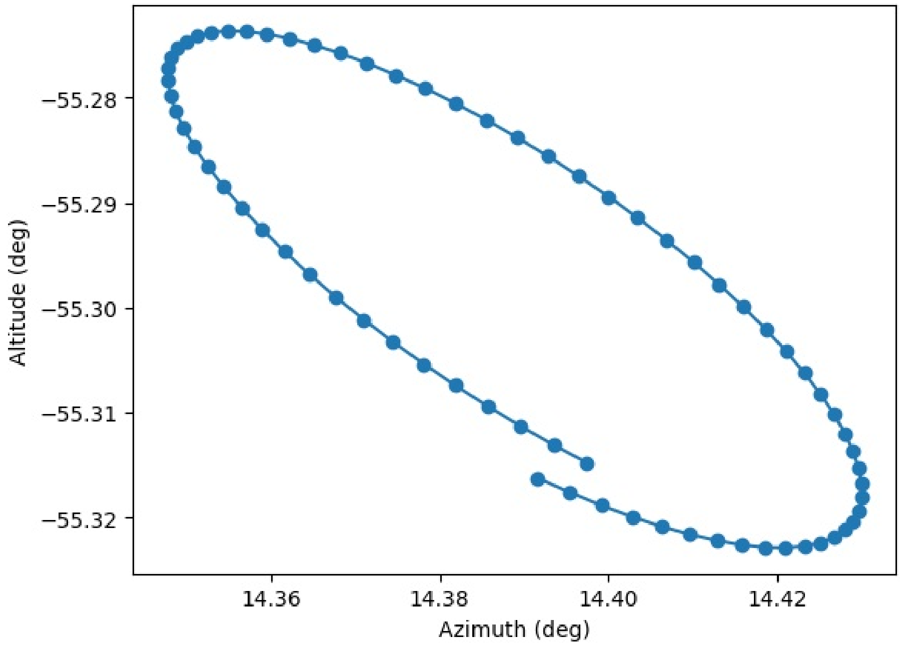
In [ ]:
```
## Array of times for a whole day in 20 minutes steps
date = datetime.datetime(2023, 11, 8, 0)
step = datetime.timedelta(minutes=20) ## enough resolution for a GEO
times = []
for _ in range(72): ## there are 72 times 20 minutes in one day
    times.append(date)
    date +=step

## Compute orbit for each time
alt = [] ## altitude list
az = []  ## azimute list
rng = []
for k in times:
    obs.date = k
    orbitElement.compute(obs)
    az.append(np.rad2deg(float(orbitElement.az)))    # degree
    alt.append(np.rad2deg(float(orbitElement.alt)))  # degree
```

Plot the satellite tracks for the times and positions you have computed above

1. Altitude vs time
2. Azimuth vs time
3. Satellite track in polar coordinates

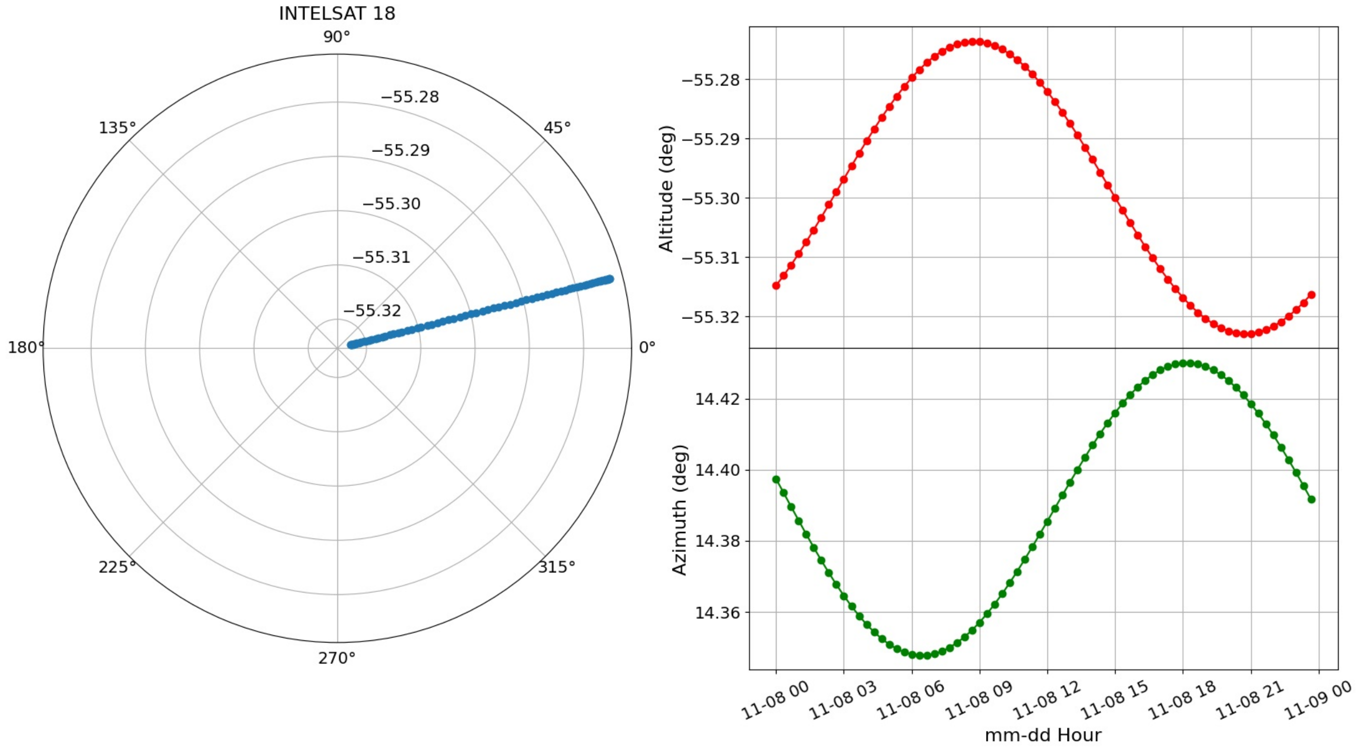
Make sure the plots are easily seen together (no scrolling!)

In [ ]:
```
## Check azimuth vs altitude
plt.plot(az,alt,'-o')
plt.xlabel('Azimuth (deg)')
plt.ylabel('Altitude (deg)')
plt.show()
```



In [ ]:
```
fig = plt.figure(figsize=(20,10))
plt.subplots_adjust(wspace=0.2, hspace=0)

# Altitude vs time
ax1 = fig.add_subplot(222)
ax1.plot(times, alt, '-o', color='r')
ax1.set_ylabel('Altitude (deg)', fontsize=16)
plt.xticks(color='w')
plt.yticks(fontsize=14)
plt.grid()
#ax1.set_title('Altitude')

# Azimute vs time
ax2 = fig.add_subplot(224)
ax2.plot(times, az, '-o', color='g')
ax2.set_xlabel('mm-dd Hour', fontsize=16)
```

```
ax2.set_ylabel('Azimuth (deg)', fontsize=16)
plt.xticks(rotation=25,fontsize=14)
plt.yticks(fontsize=14)
plt.grid()
#ax2.set_title('Azimuth')

## Position in polar coordinates (az, alt)
ax3 = fig.add_subplot(121, projection = 'polar')
ax3.plot(np.deg2rad(az), alt, '-o')
ax3.grid(True)
ax3.set_title("INTELSAT 18", va='bottom', fontsize=16)
ax3.set_rlabel_position(80)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



PART 2 - Processing TLEs on the large scale

Here is where you process the whole TLE file, propagate orbits, compute 3D cartesian positions of the satellites and write the output to a file for opening in other analsys software.

Below, I provide a utility function for reading the TLE file into a list of *EarthSatellite* objects. It is for you to understand how the function works or use another method.

In [ ]:
```
# Auxiliary function for reading 3LE files

def loadTLE(filename):
    """ Loads a TLE file and creates a list of satellites."""
    f = open(filename)
    satlist = []
    l1 = f.readline()
    while l1:
        l2 = f.readline()
        l3 = f.readline()
        sat = ephem.readtle(l1,l2,l3)
        satlist.append(sat)
        l1 = f.readline()

    f.close()
    print("%i satellites loaded into list"%len(satlist))
    return satlist
```

Read a TLE file

Read the TLEs from the file provided for this activity into a list of *EarthSatellite* objects

In [ ]:
```
## Set-up objects' list
```

```
objlist = loadTLE('SpaceTrack_3le_07112023.txt')
```

```
25875 satellites loaded into list
```

Compute orbits and create arrays of X,Y,Z cartesian positions relative to the centre of the Earth of the objects read above

The computations are for one fixed date of your choice. You may use the observer and a date defined in the beginning of the report, or set a new location and date (clearly indicated).

Sometimes, *ephem* will not compute the positions and exits with an error. This is often due to the old date of some TLEs. To overcome this unwanted termination, you will have to capture those exceptions and eliminate the problematic satellites from the list. Print on the screen the names of the satellites that have been discarded.

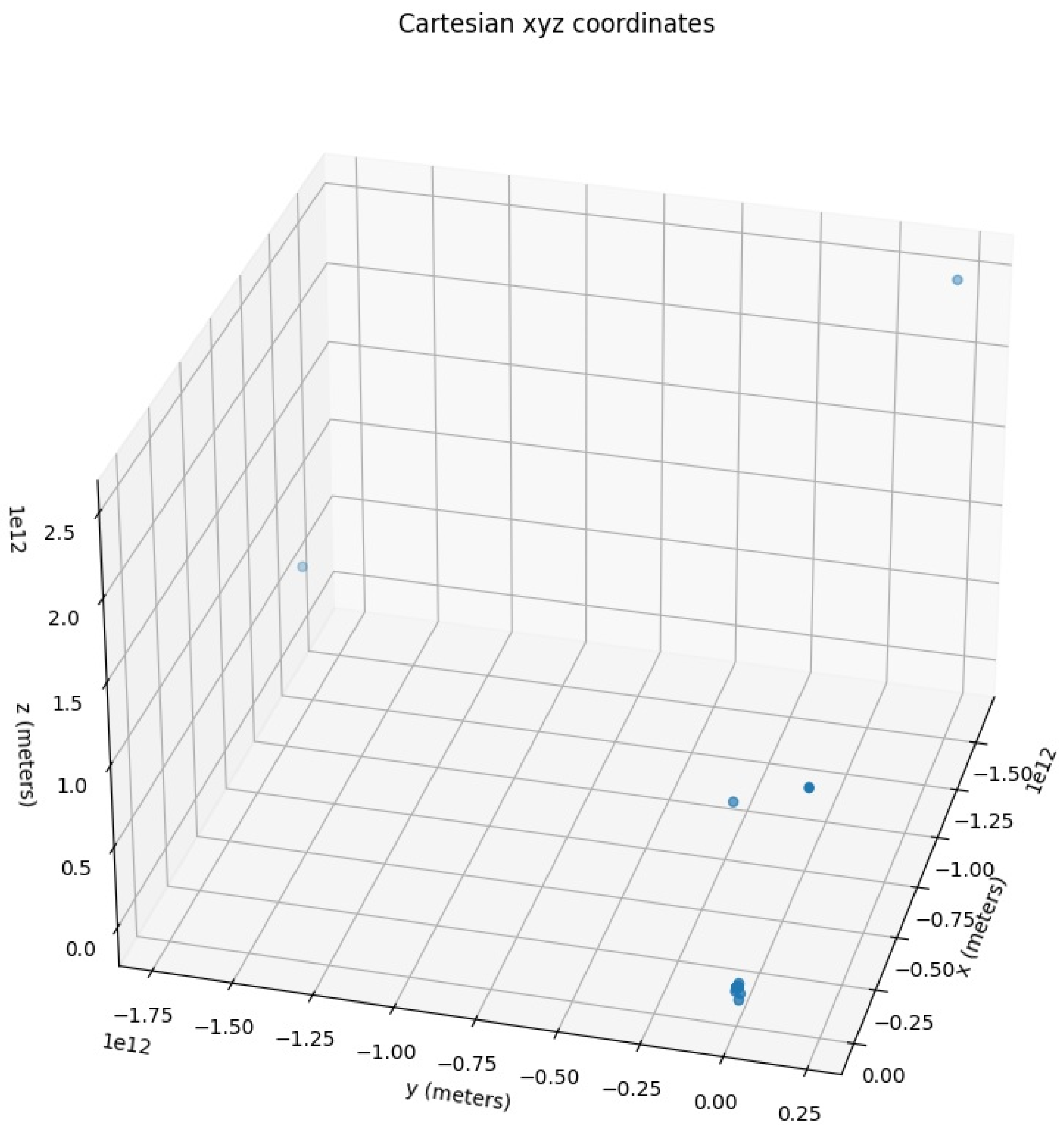
In [ ]:
```
## Compute orbits for the objects "objlist" relative to the observer "obs"
## and determine their X,Y,Z cartesian positions

XX, YY, ZZ = [], [], []

for i in range(0, len(objlist)):

    try:
        objlist[i].compute(obs)
    except ValueError:
        print("%i deleted %s: cannot compute the body's position at %s" % (i,objlist[i].name,obs.date))

    try:
        radius = ephem.earth_radius + objlist[i].elevation
        X = radius * np.cos(objlist[i].sublong) * np.cos(objlist[i].sublat)
        Y = radius * np.sin(objlist[i].sublong) * np.cos(objlist[i].sublat)
        Z = radius * np.sin(objlist[i].sublat)
    except RuntimeError:
        print("%i deleted %s: cannot compute the body's position at %s" % (i,objlist[i].name,obs.date))
        X=Y=Z = float('nan')

    XX.append(X)
    YY.append(Y)
    ZZ.append(Z)

XX = np.array(XX)
YY = np.array(YY)
ZZ = np.array(ZZ)
```

Plot cartesian XYZ positions in 3D

There are some outliers at great distances. Get rid of them to provide a good view of the LEO and GEO environments.

Note: You might get some warnings which can be safely muted with: np.warnings.filterwarnings('ignore')

In [ ]:
```
## 3D plot all data in the object list
fig = plt.figure(figsize=(14,10))
ax = plt.axes(projection='3d')
ax.scatter3D(XX,YY,ZZ)
ax.set_title('Cartesian xyz coordinates')
ax.set_xlabel('x (meters)')
ax.set_ylabel('y (meters)')
ax.set_zlabel('z (meters)')
ax.view_init(30,15)
```

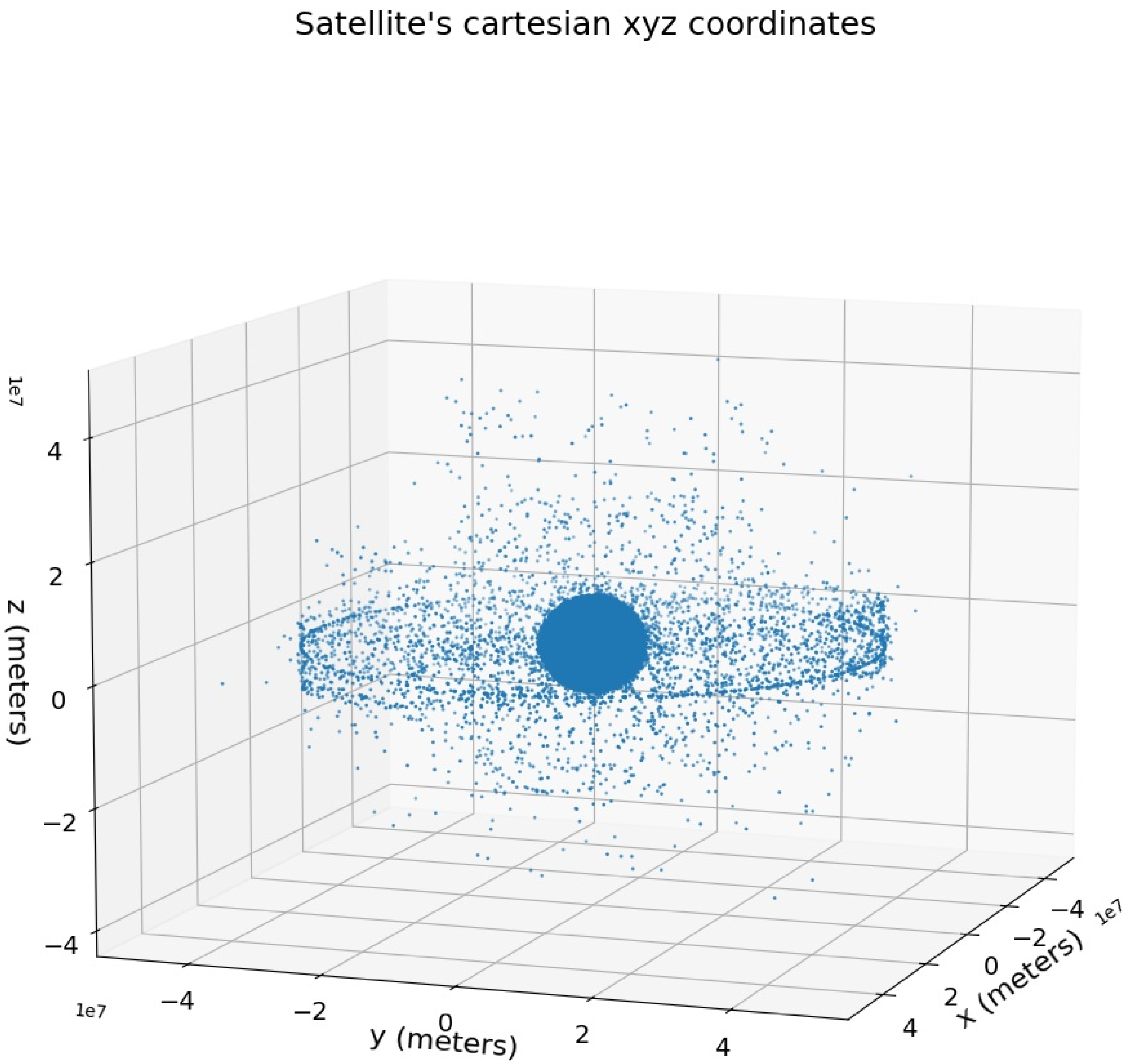


No gráfico anterior, das coordenadas cartesianas dos satélites, temos um porblema de escala (como vemos está em 10^{12} metro), devido a satélites que se afastam muito da superfície terrestre e que podemos considerar como *outliars* para o estudo que se segue.

Vamos então filtrar a lista de satélites para vermos apenas aqueles que estão até perto da órbita geoestacionária.

Para tal vamos aplicar uma "máscara" que seleciona os satélites com um raio máximo de distância ao observador de 5×10^7 metro

In [ ]:
```
## Set a maximum distance and create a mask for filtering out distance > maxrad
maxrad = 5.0e7 #meter
mask = (abs(XX) < maxrad) & (abs(YY) < maxrad) & (abs(ZZ) < maxrad)

fig = plt.figure(figsize=(18,12))
ax = plt.axes(projection='3d')
ax.scatter3D(XX[mask],YY[mask],ZZ[mask], s = 1)
ax.set_title("Satellite's cartesian xyz coordinates", fontsize=18)
ax.set_xlabel('x (meters)', fontsize=16)
ax.set_ylabel('y (meters)', fontsize=16)
ax.set_zlabel('z (meters)', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
ax.zaxis.set_tick_params(labelsize=14)
ax.view_init(10,20)
```



Na figura anterior identificamos 3 grandes regimes de órbitas dos satélites em torno da terra.

No centro, com uma distribuição aproximadamente esférica, relativamente próximos da superfície terrestre, encontramos os satélites em órbitas LEO, típicamente abaixo de ~ 2000 km.

Na zona mais externa e alinhados equatorialmente numa banda de baixa excentricidade temos as trajetórias GEO (geoestacionárias), com, como vimos anteriormente, períodos de revolução de aproximadamente 1 dia, localizados a cerca de 36000 km da superfície.

E entre estas duas zonas encontramos as trajétórias MEO, itermédias.

Export results to a csv file

You were able to do a 3D plot showing the Earth's orbital environments. That was nice. But now we want to export the data to a file to continue exploring them with more specialised interactive visualisation software.

For that, export the data from the orbit computation, as well as X,Y,Z to a CSV file. Include a header identifying the columns listed.

Note: You may also make the header more human friendy by changing the names in some fields (e.g. _n -> mean motion). Remember that the "help" instruction provides a description of the variables.

In [ ]:
```
## Print attributes and methods available for our satellite
attrlist=dir(objlist[12085])
print(attrlist)
```

```
['M', '_M', '__class__', '__copy__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__g
etattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_ap', '_
decay', '_drag', '_e', '_epoch', '_inc', '_n', '_orbit', '_raan', 'a_dec', 'a_epoch', 'a_ra', 'alt', 'ap', 'az',
'catalog_number', 'circumpolar', 'compute', 'copy', 'dec', 'decay', 'drag', 'e', 'eclipsed', 'elevation', 'elong
', 'epoch', 'g_dec', 'g_ra', 'ha', 'inc', 'mag', 'n', 'name', 'neverup', 'orbit', 'parallactic_angle', 'ra', 'ra
an', 'radius', 'range', 'range_velocity', 'rise_az', 'rise_time', 'set_az', 'set_time', 'size', 'sublat', 'sublo
ng', 'transit_alt', 'transit_time', 'writedb']
```

In [ ]:

```python
## Print attributes and methods' description and values
nobj = 12085
print("** Attributes for %s ** \n (name, value)"%objlist[nobj].name)
for attr in attrlist:
    a = getattr(objlist[nobj], attr)
    print("%s\t %s" % (attr,a))
```

```
** Attributes for 0 INTELSAT 18 **
 (name, value)
M        194.02340698242188
_M       194:01:24.3
__class__        <class 'ephem.EarthSatellite'>
__copy__         <built-in method __copy__ of ephem.EarthSatellite object at 0x7f29ba8191a0>
__delattr__      <method-wrapper '__delattr__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__dir__  <built-in method __dir__ of ephem.EarthSatellite object at 0x7f29ba8191a0>
__doc__  A satellite in orbit around the Earth, usually built by passing the text of a TLE entry to the `ephem.r
eadtle()` routine. You can read and write its orbital parameters through the following attributes:

_ap -- argument of perigee at epoch (degrees)
_decay -- orbit decay rate (revolutions per day-squared)
_drag -- object drag coefficient (per earth radius)
_e -- eccentricity
_epoch -- reference epoch (mjd)
_inc -- inclination (degrees)
_M -- mean anomaly (degrees from perigee at epoch)
_n -- mean motion (revolutions per day)
_orbit -- integer orbit number of epoch
_raan -- right ascension of ascending node (degrees)

__eq__   <method-wrapper '__eq__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__format__       <built-in method __format__ of ephem.EarthSatellite object at 0x7f29ba8191a0>
__ge__   <method-wrapper '__ge__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__getattribute__         <method-wrapper '__getattribute__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__gt__   <method-wrapper '__gt__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__hash__         <method-wrapper '__hash__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__init__         <method-wrapper '__init__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__init_subclass__        <built-in method __init_subclass__ of type object at 0x7f29bd53dc60>
__le__   <method-wrapper '__le__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__lt__   <method-wrapper '__lt__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__ne__   <method-wrapper '__ne__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__new__  <built-in method __new__ of type object at 0x7f29bd5405c0>
__reduce__       <built-in method __reduce__ of ephem.EarthSatellite object at 0x7f29ba8191a0>
__reduce_ex__    <built-in method __reduce_ex__ of ephem.EarthSatellite object at 0x7f29ba8191a0>
__repr__         <method-wrapper '__repr__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__setattr__      <method-wrapper '__setattr__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__sizeof__       <built-in method __sizeof__ of ephem.EarthSatellite object at 0x7f29ba8191a0>
__str__  <method-wrapper '__str__' of ephem.EarthSatellite object at 0x7f29ba8191a0>
__subclasshook__         <built-in method __subclasshook__ of type object at 0x7f29bd53dc60>
_ap      311:37:05.5
_decay   5.900000132896821e-07
_drag    0.0
_e       0.00020029999723192304
_epoch   2023/11/6 12:52:43
_inc     0:00:59.0
_n       1.00273532
_orbit   4424
_raan    273:02:54.6
a_dec    -4:44:16.6
a_epoch  2000/1/1 12:00:00
a_ra     14:54:13.70
alt      -55:18:58.5
ap       311.6181945800781
az       14:23:29.8
catalog_number   37834
circumpolar      False
compute  <built-in method compute of ephem.EarthSatellite object at 0x7f29ba8191a0>
copy     <built-in method copy of ephem.EarthSatellite object at 0x7f29ba8191a0>
dec      -4:50:02.2
decay    5.900000132896821e-07
drag     0.0
e        0.00020029999723192304
eclipsed         False
elevation        35777064.0
elong    0:00:00.0
epoch    2023/11/6 12:52:43
```

```
g_dec    -0:01:08.7
g_ra     14:51:38.57
ha       -11:27:21.57
inc      0.01640000008046627
mag      2.0
n        1.00273532
name     0 INTELSAT 18
neverup  True
orbit    4424
parallactic_angle        <built-in method parallactic_angle of ephem.EarthSatellite object at 0x7f29ba8191a0>
ra       14:55:28.90
raan     273.0484924316406
radius   0:00:00.0
range    47249952.0
range_velocity   -0.1031457781791687
rise_az  None
rise_time        None
set_az   None
set_time         None
size     0.0
sublat   -0:01:08.7
sublong  -179:59:18.7
transit_alt      None
transit_time     None
writedb  <built-in method writedb of ephem.EarthSatellite object at 0x7f29ba8191a0>
```

attrlist = ['perigee@epoch(deg)', 'DecayRate(rev/day^2)', '_drag', 'Eccentricity', '_epoch', 'Inclination(deg)', 'MeanMotion(rev/day)', '_orbit', '_raan', 'a_dec', 'a_epoch', 'a_ra', 'alt', 'az', 'catalog_number', 'circumpolar', 'dec', 'eclipsed', 'elevation', 'elong', 'g_dec', 'g_ra', 'mag', 'name', 'neverup', 'ra', 'radius', 'range', 'range_velocity', 'rise_az', 'rise_time', 'set_az', 'set_time', 'size', 'sublat', 'sublong', 'transit_alt', 'transit_time']

In [ ]:
```
## The list of actual data attributes - excludes object methods and docstrings
attrlist = ['_ap', '_decay', '_drag', '_e', '_epoch', '_inc', '_n', '_orbit', '_raan', 'a_dec',
               'a_epoch', 'a_ra', 'alt', 'az', 'catalog_number', 'circumpolar', 'dec', 'eclipsed',
               'elevation', 'elong', 'g_dec', 'g_ra', 'mag', 'name', 'neverup', 'ra', 'radius', 'range',
               'range_velocity', 'rise_az', 'rise_time', 'set_az', 'set_time', 'size', 'sublat', 'sublong',
               'transit_alt', 'transit_time']
```

In [ ]:
```
## Create the "header" array from the list of attributes
header = attrlist.copy()

## Include the previously calculated x, y and z coordinates' rows in the header
header.append('X')
header.append('Y')
header.append('Z')

## Update attributes' names to make them more human readable
header[0:14] = ['Perigee@Epoch(deg)',                                                          # _ap
                'DecayRate(rev/day^2)',                                                        # _decay
                'DragCoeff',                                                                   # _drag
                'Eccentricity',                                                                # _e
                'ReferenceEpoch',                                                              # _epoch
                'Inclination(deg)',                                                            # _inc
                'MeanMotion(rev/day)',                                                         # _n
                'OrbitNumber',                                                                 # _orbit
                'RightAscensionOfAscendingNode(deg)',                                          # _raan
                'AstrometricGeocentricDeclination',                                            # a_dec
                'DateGivingTheEquinoxOfTheBodyAstrometricRightAscensionAndDeclination',        # a_epoch
                'AstrometricGeocentricRightAscension',                                         # a_ra
                'Altitude',                                                                    # alt
                'Azimute']                                                                     # az
header[16] = 'Declination'                                                                     # dec
header[19:28] = ['ApparentGeocentricDeclination',                                              # g_dec
                 'ApparentGeocentricRightAscension',                                           # g_re
                 'Magnitude',                                                                  # mag
                 'ObjectName',                                                                 # name
                 'NeverUp',                                                                    # neverup
                 'RightAscension',                                                             # ra
                 'VisualRadius',                                                               # radius
                 'DistanceFromObserver2Satellite(m)']                                          # range

print(header)
```

```
['Perigee@Epoch(deg)', 'DecayRate(rev/day^2)', 'DragCoeff', 'Eccentricity', 'ReferenceEpoch', 'Inclination(deg)'
, 'MeanMotion(rev/day)', 'OrbitNumber', 'RightAscensionOfAscendingNode(deg)', 'AstrometricGeocentricDeclination'
, 'DateGivingTheEquinoxOfTheBodyAstrometricRightAscensionAndDeclination', 'AstrometricGeocentricRightAscension',
'Altitude', 'Azimute', 'catalog_number', 'circumpolar', 'Declination', 'eclipsed', 'elevation', 'ApparentGeocent
ricDeclination', 'ApparentGeocentricRightAscension', 'Magnitude', 'ObjectName', 'NeverUp', 'RightAscension', 'Vi
sualRadius', 'DistanceFromObserver2Satellite(m)', 'range_velocity', 'rise_az', 'rise_time', 'set_az', 'set_time'
, 'size', 'sublat', 'sublong', 'transit_alt', 'transit_time', 'X', 'Y', 'Z']
```

In [ ]:
```
## Open the file for writing
```

```python
resultFile = open("XYZtle_07112023.csv",'w')
wr = csv.writer(resultFile, delimiter=',')

## Write the header
wr.writerow(header)
```

Out[ ]: 630

In [ ]:
```python
## Write the data

# iterate over the objects
for i in range(0, len(objlist)):
    outline = []
    # fill the output line by iterating over the object's attributes
    for attr in attrlist:
        try:
            a = getattr(objlist[i], attr) # try to get the attribube from the object list
        except RuntimeError:
            a = (float('nan'))            # if it stinks, make it nan
        try:
            outline.append(float(a))      # try to append it as float
        except ValueError:
            outline.append(a)             # if it dosen't like it, append as it is
        except TypeError:
            outline.append(float('nan'))  # if still dosen't like it, make it nan

    # append the previously computed  x, y and z coordinates
    outline.extend((XX[i], YY[i], ZZ[i]))
    # write the line for this object
    wr.writerow(outline)

# close the output file
resultFile.close()
```

Check all went fine: Print the header and first lines to the screen

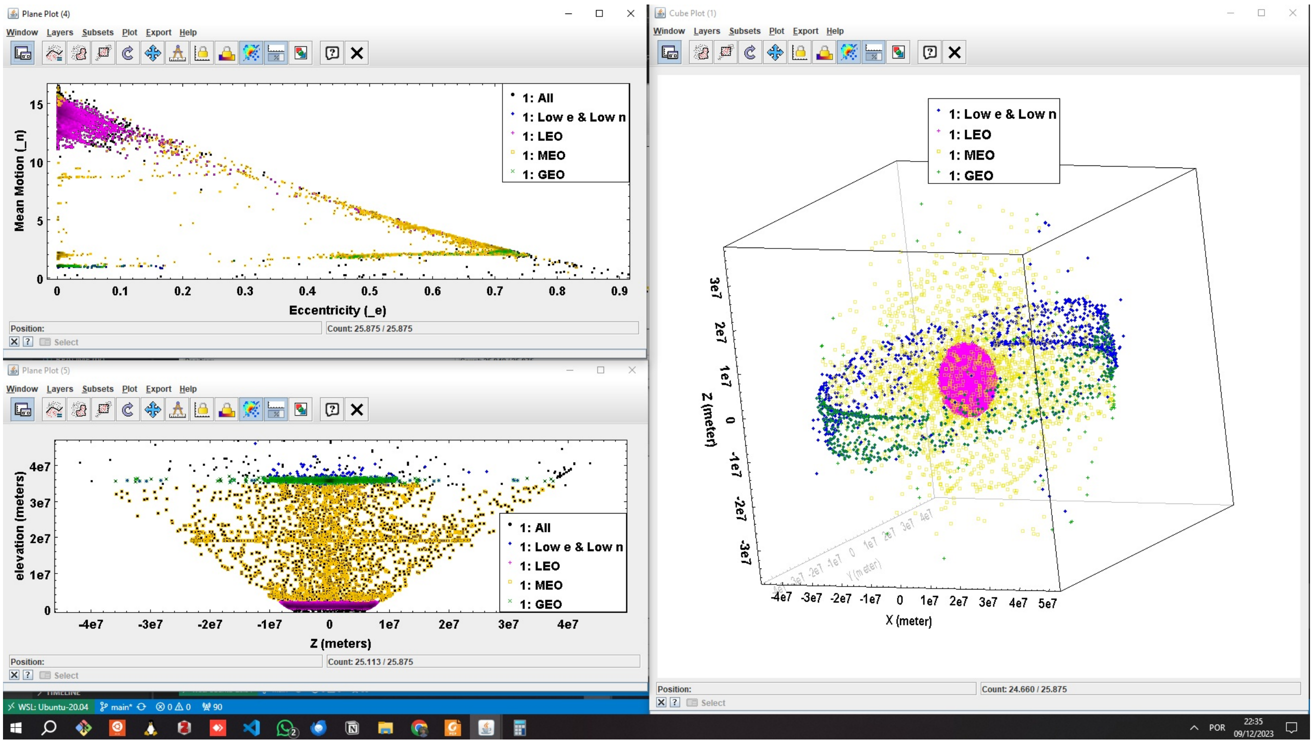
In [ ]:
```python
## Open result file
resultFile = open("XYZtle_07112023.csv",'r')

## Print header and first 3 data lines
for _ in range(4):
    print(resultFile.readline())
```

```
Perigee@Epoch(deg),DecayRate(rev/day^2),DragCoeff,Eccentricity,ReferenceEpoch,Inclination(deg),MeanMotion(rev/da
y),OrbitNumber,RightAscensionOfAscendingNode(deg),AstrometricGeocentricDeclination,DateGivingTheEquinoxOfTheBody
AstrometricRightAscensionAndDeclination,AstrometricGeocentricRightAscension,Altitude,Azimute,catalog_number,circ
umpolar,Declination,eclipsed,elevation,ApparentGeocentricDeclination,ApparentGeocentricRightAscension,Magnitude,
ObjectName,NeverUp,RightAscension,VisualRadius,DistanceFromObserver2Satellite(m),range_velocity,rise_az,rise_tim
e,set_az,set_time,size,sublat,sublong,transit_alt,transit_time,X,Y,Z

2.7769148202486647,4.850000095757423e-06,0.0006167599931359291,0.18433499336242676,45234.74765951,0.597752565359
765,10.85107214,33969.0,0.3041323639862901,-0.6991744107911209,36525.0,4.920503805020251,-0.9008600115776062,4.3
91244411468506,5.0,0.0,-0.6986872927293324,0.0,2956621.75,0.0,-0.5247940463653372,5.4226416551139405,2.0,0 VANGU
ARD 1,0.0,4.927743119961611,0.0,13436016.0,-1393.5147705078125,4.220431804656982,45237.51161829195,1.47927069664
00146,45237.53812050738,0.0,-0.5247940421104431,-1.6092764139175415,1.0828027725219727,45237.529037760745,-31078
7.50405727315,-8072593.281772855,-4677050.262134317

0.7274515065028502,1.0890000339713879e-05,0.000581329979468137,0.1460617035627365,45235.13808888,0.5736164029794
041,11.87373792,42192.0,1.5095858328263518,-1.1878481411143476,36525.0,0.5952346681951094,-0.30555179715156555,3
.262261390686035,11.0,0.0,-1.1859302618155598,1.0,1595031.625,0.0,-0.32778646871679895,0.8006830414762269,2.0,0
VANGUARD 2,0.0,0.5973470279031183,0.0,7049855.5,-3576.640380859375,3.696591377258301,45237.66774543216,1.7022973
29902649,45237.67529046734,0.0,-0.32778647541999817,0.05195024609565735,0.2097758948802948,45237.67155673794,753
8493.394649458,391979.2787310565,-2566954.279460798

1.8506546099458734,9.839999620453455e-06,0.0005819500074721873,0.16594550013542175,45235.29487632,0.574234256352
3024,11.45858574,42179.0,5.910264457632445,-0.1843165030970517,36525.0,5.810534283400359,0.040530163794755936,4.
448213577270508,12.0,0.0,-0.18224982877035245,1.0,1159794.375,0.0,0.4504118088101142,0.33769239315773264,2.0,0 V
ANGUARD R/B,0.0,5.816063545021478,0.0,3796648.25,-6442.734375,4.6664252281188965,45237.579074666704,1.7505921125
411987,45237.496301944484,0.0,0.4504117965698242,-0.41104039549827576,0.7580578923225403,45237.49196945907,62209
27.8641981855,-2711507.711662664,3281545.1539812987
```

Quick analyis with TOPCAT

- Open the csv file in TOPCAT
- Make a 3D plot of X,Y,Z to show the LEO/MEO/GEO environments
- Make a (2D) scatter plot of eccentricity vs mean_motion
- Create a subset of the data by selecting a region you find interesting in the 2D plot
- Show how this subset appears distributed in the 3D plot

- Take a screenshot of your Desktop with the TOPCAT plots clearly seen and display it in the markdown cell below (double click on the example image from Gaia DR3)



Na figura anterior vemos, do lado direito, a distribuição 3D das coordenadas x, y e z dos satélites. Neste gráfico são apenas visíveis os satélites posteriormente identificados como LEO, MEO, GEO e "Low e &Low n", e não todos os satélites presentes no ficheiro "XYZtle_07112023.csv".

A seleção dos três diferentes regimes de órbitas foi feita através do gráfico da elevação em função da coordenada z (à esquerda e em baixo) - mas pode ser obtida usando outros critérios.

Para os satélites LEO (+ rosa) foi feita uma selação com a ferramenta "freehand" dos pontos com elevação entre aproximadamente zero e 2000 km, ao longo de todo o z (neste caso entre aprox. -1×10^7 m e 1×10^7 m).

Para os satélites GEO (x verdes) foi feita a seleção com a ferramenta "freehand" dos pontos com elevação entre aprox. 35000 km e 36000 km, ao longo de todo o z (neste caso aprox. -4×10^7 m e 4×10^7 m).

Para os satélites MEO (□ amarelos) foi feita uma selação com a ferramenta "freehand" dos pontos com elevação entre aprox. 2000 km e 34000 km, para z entre aprox -2×10^7 m e 2×10^7 m.

No caso dos GEO vemos alguns pontos selecionados que não pertencem a esta órbita geoestacionária mas que acabam por ser incluidos no contorno "freehand" dos pontos.

Vamos agora localizar as órbitas anteriores no gráfico do movimento médio em função da excentricidade (esquerda, cima). Seria de esperar que os satélites LEO se localizassem no canto superior esquerdo do gráfico uma vez que se trata de satélites muito próximos, ou seja com excentricidades tipicamente pequenas, e com mean motion elevado por terem períodos de revolução curtos.

No caso dos satélites GEO, seria de esperar que tivessem também excentricidades relativamente pequenas, uma vez que as suas órbitas devem ser aproximadamente muito circulares e com movimento médio constante e relativamente pequeno sendo que têm períodos de órbita de ~1 dia. De facto, olhando de novo melhor para a distribuição 3D das coordenadas podemos identificar algumas órbitas que se começão a deslocar do movimento equatorial.

Uma outra escolha para os satélites GEO podia então ser, a partir do gráfico do movimento médio em função da excentricidade, os pontos que têm baixo movimento e excentricidade também relativamente baixa e constante (ex. $\diamond$ azuis).