ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

Projet Programmation Scientifique

TLE et traces des satellites dans le ciel

Réalisé par :

ABDUL KUTHOOS Hannick, LE Marie

26 Janvier 2022

Compréhension du sujet

Dans le cadre de ce projet de programmation scientifique, nous avons cherché à créer à l'aide du langage de programmation Python des cartes du ciel (skyplots).
Dans ce but, nous avons premièrement essayé de comprendre le fond physique afin de pouvoir s'approprier les formules présentées on a donc essayer d'interpréter les grandeurs successivement présentes dans les fonctions proposés dans les différentes étapes. Nous avions aussi besoin de renseignements sur les TLE et leur exploitation en tant que Dataframe à l'aide du module pandas ce qui a été possible à l'aide de la documentation de pandas ainsi que de l'annexe présente dans le sujet.
Le sujet repose ainsi sur deux parties, une ou l'on doit obtenir des traces pour les différentes constellations dans notre cas : ISS, GPS, GLONASS, GALILEO sur la semaine du lundi 08/11/2021 à 00 h 00 UTC au dimanche 14/11/2021 à 23 h 59 UTC. Et dans un second temps utiliser les fonctions codées dans la première partie et de nouvelles fonctions afin de créer des skyplots pour les différentes constellations, pour 15 stations différentes.(avec un skyplot bonus plus spécifique ou l'on considère une antenne sur l'ISS en tant que station..)

1 Programmation

1.1 Étape 1

Dans cette première étape nous avons premièrement réfléchis à la structure du code et à la répartition des tâches. Une personne s'occupait du dataframe tandis que l'autre s'occupait du plot et de la programmation des fonctions.
La structure de l'étape 1 sera identique pour les autres. Nous avons fait plusieurs scripts pour les différentes constellations ainsi que des scripts dédiés aux fonctions utilisés ainsi que l'ouverture des fichiers. Par exemple les fichiers 'iss.py', 'glo.py', 'galileo.py', 'gps.py' contiennent respectivement les traces des satellites correspondants. Le fichier 'etape_1.py' va montrer tout les plots en même temps. Le fichier 'fonctions.py' contient la programmation des équations physiques. Et le fichier 'lecture_de_tle.py' contient la méthode utilisée pour ouvrir les fichiers tle.

1.1.1 Lecture des fichiers

Après avoir examiné les 4 fichiers des constellations gps, glo, iss et galileo. Nous constatons que ces fichiers ont exactement la même structure (même emplacements pour les paramètres, mais différentes valeurs). Un fichier python lecture_de_TLE suffit donc pour traiter ces 4 fichiers.

On utilise Path().absolute().parent pour obtenir le chemin absolu du script et file pour avoir le répertoire parent du script (indépendamment du répertoire de travail actuel). Cette étape est importante pour éviter de réécrire tout le chemin d'accès du fichier à chaque fois.

```
working_directory = Path(__file__).absolute().parent
```

La fonction prepend insère une chaîne de caractère au début de tous les éléments d'une

liste. On aura besoin de cette fonction plus tard pour récupérer les valeurs corresponds à «l'heure» en ajoutant 0. devant chaque élément.

```
def prepend(list, str):
    str += '{0}'
    list = [str.format(i) for i in list]
    return(list)
```

La fonction elements_keplerien(fichier) prend en argument le fichier à traiter et renvoie un tableau pandas contenant les valeurs de 'Ascension', 'Argument du périgée', 'Anomalie moyenne', 'Heure', 'Jour' pour chaque satellite.

```
def elements_keplerien(fichier):

    omega=[]
    petit_omega =[]
    M= []
    nom_sat_beta = []
    date =[]

    with open(working_directory/fichier,"r") as f:
        lines = f.readlines()

        # Parcourir les lignes du fichier TLE
        for l in lines:
            # Parcourir les valeurs des lignes d'indice 2 du fichier TLE
            if l[0] == '2':
                omega.append(l.split()[3])
                petit_omega.append(l.split()[5])
                M.append(l.split()[6])

            # Parcourir les valeurs des lignes d'indice 1 du fichier TLE
            if l[0] == '1':
                date.append(l.split()[3])

            # Obtenir les noms des satellites; ils commencent soit par C,
                                                soit par G
            if l[0] == 'G' or l[0] == 'C' or l[0] == 'I':
                nom_sat_beta.append(l)

    # Transformer les valeurs du type string en float
    omega = [ float(x) for x in omega ]
    petit_omega = [ float(x) for x in petit_omega ]
    M = [ float(x) for x in M ]


    annee_jour = []
    heure = []

    for i in range(len(date)):
        annee_jour.append(date[i].split('.',1)[0])
        heure.append(date[i].split('.',1)[1])

    heure = prepend(heure, '0.')

    jour = [e[2:] for e in annee_jour]
```

```
    for i in range(len(M)):
        M[i] =(M[i]*math.pi)/180

    # Enlever les /n dans la liste nom_sat
    nom_sat = [s.replace("\n", "") for s in nom_sat_beta]

    # Cr er un tableau avec pandas
    mydataset = {
  'Ascension': omega, 'Argument du p rig e' : petit_omega,
  'Anomalie moyenne' : M, 'Heure' : heure, 'Jour' : jour
}
    myvar = pd.DataFrame(mydataset, index = nom_sat)

    # Afficher toutes les colonnes du tableau, car certaines sont cach es
    pd.set_option('display.max_columns', None)
    return myvar
```

La syntaxe open('fichier','r') est utilisée pour ouvrir le fichier TLE. Le deuxième paramètre est renseigné par un r, ce paramètre indique une ouverture de fichier en lecture.

```
with open(working_directory/fichier,"r") as f:
```

La méthode readlines() renvoie une liste contenant chaque ligne du fichier comme un élément de liste.

```
lines = f.readlines()
```

La méthode split() de Python est utilisée pour diviser la chaîne en morceaux. Nous précisons le délimiteur '.' quand pour séparer les éléments avant et après '.' (année-jour et heure), les espaces sont des délimiteurs par défaut sinon.

```
for l in lines:
    if l[0] == '2':
        omega.append(l.split()[3])
        petit_omega.append(l.split()[5])
        M.append(l.split()[6])
```

Pour convertir une liste de chaînes de caractères en une liste de nombres flottants, on crée une liste comme celle-ci floats = [float(x) for x in strings].

```
omega = [ float(x) for x in omega ]
petit_omega = [ float(x) for x in petit_omega ]
M = [ float(x) for x in M ]
```

Pour créer un DataFrame Pandas en Python, nous suivons le modèle générique

```
myvar = pd.DataFrame(mydataset, index = nom_sat)
```

La syntaxe pd.set_option('display.max_columns', None) étend l'affichage de la sortie pour voir plus de colonnes d'un dataframe Pandas. None en deuxième paramètre signifie nombre de colonne illimité.

```
pd.set_option('display.max_columns', None)
```

1.1.2 Fonctions

Dans la programmation des fonctions nous avons décider de donner des noms aux grandeurs physiques correspondant à leurs symboles la plupart du temps. Nous travaillons durant tout le projet avec des unités SI. Donc certaines valeurs doivent être rentrés en unité SI pour avoir des résultats cohérents (exemple: calcul_a ou on a besoin du moyen mouvement du satellite en rad/s). Les variables peuvent changer de noms selon les constellations traités dans les fichiers (exemple: a devient a_iss dans iss.py).
Le fichier test_unitaire.py permet de faire des tests unitaires sur les fonctions codés dans l'étape 1 et l'étape 2.

Le fichier fonctions.py commence par une fonction qui calcule la valeur de a à l'aide de la troisième loi de Kepler : $a^3n^2 = \mu$.

```
def calcul_a(n):

    GM = 3.986005*(10**14)
    a = (GM/(n**2))**(1/3)
    return a
```

Par la suite, nous avons deux fonctions pour définir notre échelle de temps. La fonction tle_to_utc renvois l'heure à partir de l'heure fractionnaire du TLE. dateJulienne qui prend en argument l'année, le mois, le jour et l'heure qui nous renvoie la date julienne UTC. Et la fonction t qui prend en argument la date julienne et renvois la date julienne modifié UTC en secondes.

```
def tle_to_utc(heure_tle):
    heure = heure_tle*24
    return np.floor(heure)

def dateJulienne(annee,mois,jour,heure):

    if mois<=2:
        a=annee-1
        m=mois+12
    else:
        a=annee
        m=mois
    b=int(a/400)-int(a/100)
    DJ = int(365.25*a)+int(30.6001*(m+1))+b+1720996.5+jour+heure*(1/24)
    return DJ


def t(DJ):

    return (DJ-2400000.5)*86400
```

Nous avons trois paramètres qui vont varier au cours du temps l'ascension, l'argument du périgée et l'anomalie moyenne donc trois fonctions qui vont implémenter ces variations : var_omega,var_OMEGA et var_anomalie_moyenne.

```
# Calcul de la variation  d anomalie  moyenne depuis  l instant  de
                                    r f rence du TLE
def var_anomalie_moyenne(M,n0,J2,a,e,i,t,t_tle):

```

```
    R = 6371000
    deltan = (3/4)*n0*J2*((R/a)**2)*(1/((1-e*e)**(3/2)))*(3*(np.cos(i)**2)
                                                -1)
    delta_M = (n0+deltan)*(t-t_tle)
    M = M + delta_M
    return M

# Calcul de la variation pour l'Ascension
def var_OMEGA(W0,n0,J2,a,e,i,t,t0):
    R = 6371000
    W_dot = (-3/2)*n0*J2*((R/a)**2)*(1/((1-e*e)**2))*np.cos(i)
    W = W0 + W_dot*(t-t0)
    return W

# Calcul de la variation pour l'argument

def var_omega(w0,n0,J2,a,e,i,t,t0):
    R = 6371000
    w_dot = (3/4)*n0*J2*((R/a)**2)*(1/((1-e*e)**2))*(5*(np.cos(i)**2-1))
    w = w0 + w_dot*(t-t0)
    return w
```

Les fonctions rayon_orbital et anomalie_vraie calculent respectivement le rayon orbital et l'anomalie vraie, or, on ne les utilisera pas par la suite. Nous allons plutôt utiliser la valeur de l'anomalie excentrique calculée par résolution de l'équation de Kepler.

```
# Resolution equation de Kepler

def kepler_resolve(M,e,epsilon):

    E = M
    seuil = 1e6
    while seuil>epsilon:
        nouveau_E = M + e*np.sin(E)
        seuil = abs(nouveau_E-E)
        E = nouveau_E
    return E

# Calcul du rayon orbital
def rayon_orbital(a,e,E):
    r = a*(1-e*np.cos(E))
    return r


# Calcul de l'anomalie vraie
def anomalie_vraie(e,a,E,r):
    v = np.arccos((a/r)*(np.cos(E)-e)) #E est en radian comme on a
                                          convertis M en radian
    return v
```

On passe ensuite aux fonctions qui calculent les coordonnées. La fonction coord_orbital va utiliser la méthode avec l'anomalie excentrique. Ensuite les fonctions coord_cart, coord_celeste et coord_geodesique vont calculer respectivement les coordonnées ECI, ECEF et géodésique.

```
# Coordonn es dans le plan orbital
def coord_orbital(a,e,E):
    x = a*(np.cos(E)-e)
```

```
    y = a*np.sqrt(1-e*e)*np.sin(E)
    return (x,y)


# Coordonn es dans le plan c leste
def coord_celeste(i,w,W,e,E,a):
    R_inclinaison = np.array([[1,0,0],[0,np.cos(i),-np.sin(i)],[0,np.sin(i
                                        ),np.cos(i)]])
    R_argument = np.array([[np.cos(w),-np.sin(w),0],[np.sin(w),np.cos(w),0
                                        ],[0,0,1]])
    R_ascension = np.array([[np.cos(W),-np.sin(W),0],[np.sin(W),np.cos(W),
                                        0],[0,0,1]])
    (x,y) = coord_orbital(a,e,E)
    coord_orb = np.array([[x],[y],[0]])
    coord = R_ascension.dot(R_inclinaison).dot(R_argument.dot(coord_orb))

    return coord

# Coordonn es cart siennes
def coord_cart(i,w,W,e,E,a,t,vitesse_angulaire):
    matrice_passage = np.array([[np.cos(vitesse_angulaire*t),-np.sin(
                                        vitesse_angulaire*t),0],[np.sin(
                                        vitesse_angulaire*t),np.cos(
                                        vitesse_angulaire*t),0],[0,0,1]])
    coord_car = np.dot(matrice_passage,coord_celeste(i,w,W,e,E,a))
    return coord_car

# Coordonn es g od siques ECEF
def coord_geodesique(i,w,W,E,t,vitesse_angulaire):
    a = 6378137.0
    f = 1/298.257222101
    b = a*(1-f)
    e = np.sqrt((a**2-b**2)/a**2)
    coord_car = coord_cart(i,w,W,e,E,a,t,vitesse_angulaire)
    X = coord_car[0][0]
    Y = coord_car[1][0]
    Z = coord_car[2][0]
    R = np.sqrt(X**2+Y**2+Z**2)
    mu = np.arctan((Z/np.sqrt(X**2+Y**2))*((1-f)+(((e**2)*a)/R)))
    lon = np.arctan2(Y,X)
    lat = np.arctan((Z*(1-f)+(e**2)*a*(np.sin(mu)**3))/((1-f)*(np.sqrt(X**
                                        2+Y**2)-(e**2)*a*(np.cos(mu)**3))
                                        ))
    return (lon,lat)
```

1.1.3 Plot

A l'aide des fonctions définies dans les parties précédentes, nous pouvons maintenant décrire les traces pour les différentes constellations. Nous définissons les paramètres constants du problème ainsi qu'un linspace sur la semaine sur lequel on va boucler pour créer le plot des longitudes, latitudes. Voici un exemple pour le satellite galileo GSAT0209 :

Définition des paramètres :

```
# D finition des param tres du satellite galileo GSAT0209
```

```
a_galileo = fonctions.calcul_a(1.45*(10**(-4)))

i_galileo = np.radians(55.0919)

e_galileo = 0.0002373

n0 = 1.45*(10**(-4))

J2 = 1.08262652304819194e-3

w0 = np.radians(lecture_de_TLE.elements_keplerien("donn es/galileo.txt")[
                                        'Argument du p rig e'][11])

W0 = np.radians(lecture_de_TLE.elements_keplerien("donn es/galileo.txt")[
                                        'Ascension'][11])

M_galileo = lecture_de_TLE.elements_keplerien("donn es/galileo.txt")['
                                        Anomalie moyenne'][11]

vitesse_angulaire = -7.2921151467e-5

heure_tle = fonctions.tle_to_utc(float(lecture_de_TLE.elements_keplerien("
                                        donn es/galileo.txt")['Heure'][11]))

date_tle = fonctions.t(fonctions.dateJulienne(2021,11,8,heure_tle))
```

Définition du linspace :

```
DATE = np.linspace(fonctions.t(2459526.5),fonctions.t(2459527.5),num=10000
                                        )
```

Boucle sur le linspace (et affichage):

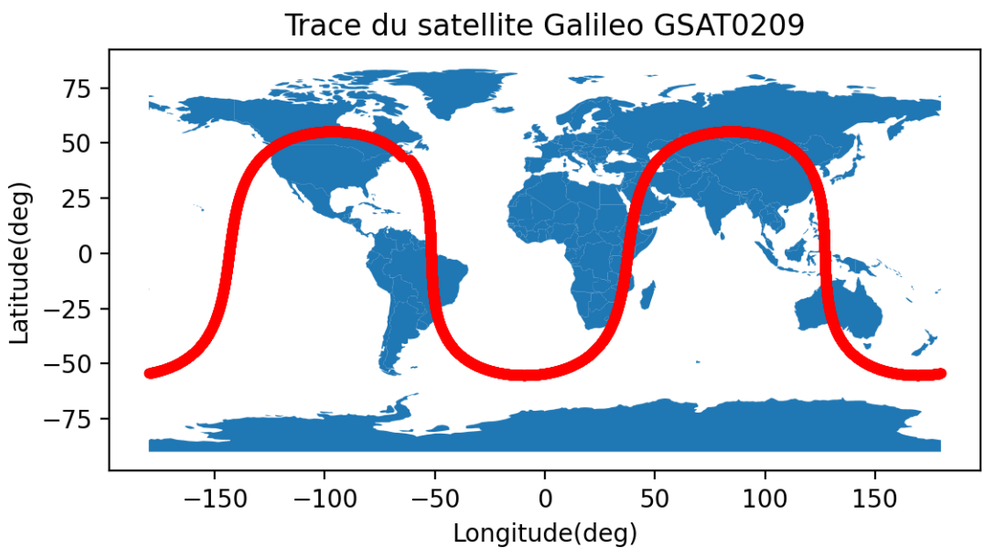
```
X = []

Y = []

for date in DATE:
    w_galileo = fonctions.var_omega(w0,n0,J2,a_galileo,e_galileo,i_galileo
                                        ,date,date_tle)

    W_galileo = fonctions.var_OMEGA(W0,n0,J2,a_galileo,e_galileo,i_galileo
                                        ,date,date_tle)

    M_corr_galileo = fonctions.var_anomalie_moyenne(M_galileo,n0,J2,
                                        a_galileo,e_galileo,i_galileo,
                                        date,date_tle)

    E_galileo = fonctions.kepler_resolve(M_corr_galileo,e_galileo,1e-6)

    coord_orbital_galileo = fonctions.coord_orbital(a_galileo,e_galileo,
                                        E_galileo)

    coord_celeste_galileo = fonctions.coord_celeste(i_galileo,w_galileo,
                                        W_galileo,e_galileo,E_galileo,
                                        a_galileo)

    coord_cart_galileo = fonctions.coord_cart(i_galileo,w_galileo,
```

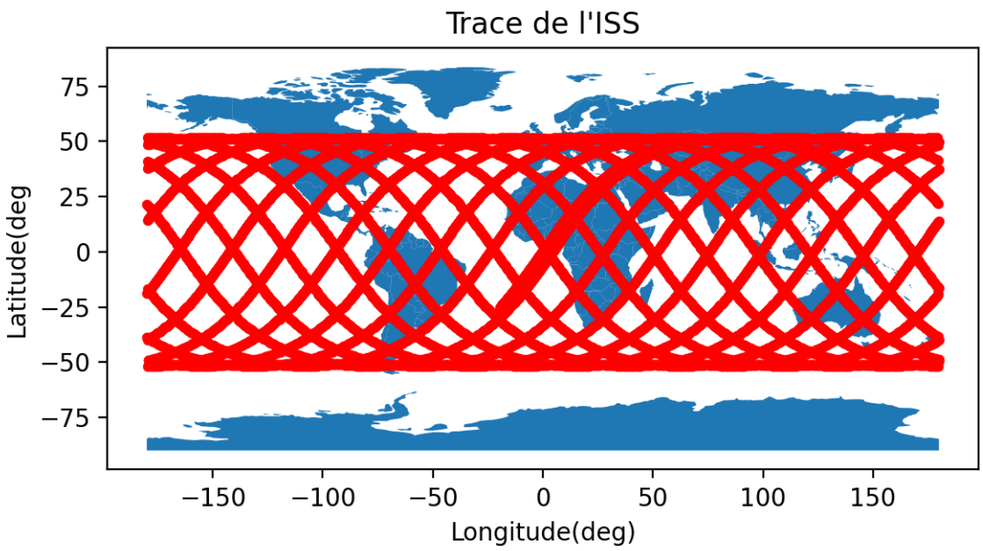
```
                                        W_galileo,e_galileo,E_galileo,
                                        a_galileo,date,vitesse_angulaire)

    coord_geodesique_galileo = fonctions.coord_geodesique(i_galileo,
                                        w_galileo,W_galileo,E_galileo,
                                        date,vitesse_angulaire)

    X.append(np.degrees(coord_geodesique_galileo[0]))
    Y.append(np.degrees(coord_geodesique_galileo[1]))
```

Nous avons aussi utilisé le module geopandas pour mettre un fond de carte en 2D :

```
world = geopandas.read_file(geopandas.datasets.get_path('
                                    naturalearth_lowres'))

fig, ax = plt.subplots()
world.plot(ax=ax)
plt.scatter(X,Y,s=10,c='red')
plt.title('Trace du satellite Galileo GSAT0209')
plt.xlabel('Longitude(deg)')
plt.ylabel('Latitude(deg)')
plt.show()
```

1.1.4 Quelques traces





1.2 Étape 2 & Étape 3

Dans cette partie, nous gardons la même structure pour les fichiers et la répartition des tâches reste la même. Nous devons utiliser les scripts précédents et de nouvelles fonctions pour avoir un skyplot.

1.2.1 Lecture du fichier DORIS

On écrit un nouveau code pour la lecture du fichier DORIS, car il ne contient pas les mêmes informations que les autres fichiers TLE. Ici, on a des coordonnées des stations et on souhaite créer 3 listes différentes pour les valeurs de X Y Z.

On utilise Path().absolute().parent pour obtenir le chemin absolu du script et file pour avoir le répertoire parent du script (indépendamment du répertoire de travail actuel).

```
working_directory = Path(__file__).absolute().parent
```

Ensuite, on ouvre le fichier avec la syntaxe open('fichier','r'). Le deuxième paramètre est renseigné par un r, ce paramètre indique une ouverture de fichier en lecture.

```
with open(working_directory/'donn es/DORIS_stations.txt','r') as f:
```

La méthode readlines() renvoie une liste contenant chaque ligne du fichier comme un élément de liste. En précisant [1:], on ne prend les lignes qu'à partir de la deuxième ligne du fichier.

```
lines = f.readlines()[1:]
```

La méthode split() de Python est utilisée pour diviser la chaîne en morceaux.

```
for l in lines:
    NAME.append(l.split()[0])
    X.append(l.split()[1])
    Y.append(l.split()[2])
    Z.append(l.split()[3])
```

Pour convertir une liste de chaînes de caractères en une liste de nombres flottants, on crée une liste comme celle-ci floats = [float(x) for x in strings].

```
X = [float(x) for x in X]
Y = [float(x) for x in Y]
Z = [float(x) for x in Z]
```

1.2.2 Fonctions

Comme dans la première étape, nous allons utiliser fonctions.py et créer un nouveau fichier avec les nouvelles fonctions fonctions_2.py.

Le fichier contient une fonction coord_sat_topo qui calcul les coordonnées des satellites dans le repère topocentrique.

```
def coord_sat_topo(X_sta,Y_sta,Z_sta,X_sat,Y_sat,Z_sat,lam,phi):
    matrice_de_passage = np.array([[-np.sin(lam),np.cos(lam),0],[-np.cos(
                                    lam)*np.sin(phi),-np.sin(lam)*np.
                                    sin(phi),np.cos(phi)],[np.cos(lam
                                    )*np.cos(phi),np.sin(lam)*np.cos(
                                    phi),np.sin(phi)]])
    A = np.array([[X_sat-X_sta],[Y_sat-Y_sta],[Z_sat-Z_sta]])
    coord = np.dot(matrice_de_passage,A)
    X_sat_loc = coord[0][0]
    Y_sat_loc = coord[1][0]
    Z_sat_loc = coord[2][0]
    return (X_sat_loc,Y_sat_loc,Z_sat_loc)
```

Une fonction azimuth_elevation qui va calculer l'élévation et l'azimut du satellite vis à vis d'une station en utilisant la fonction précédente.

```
def azimuth_elevation(X_sta,Y_sta,Z_sta,X_sat,Y_sat,Z_sat,lam,phi):
    X_sat_loc = coord_sat_topo(X_sta,Y_sta,Z_sta,X_sat,Y_sat,Z_sat,lam,phi
                                    )[0]
    Y_sat_loc = coord_sat_topo(X_sta,Y_sta,Z_sta,X_sat,Y_sat,Z_sat,lam,phi
                                    )[1]
    Z_sat_loc = coord_sat_topo(X_sta,Y_sta,Z_sta,X_sat,Y_sat,Z_sat,lam,phi
                                    )[2]

    d = np.sqrt(X_sat_loc**2+Y_sat_loc**2+Z_sat_loc**2)
    d_hori = np.sqrt(X_sat_loc**2+Y_sat_loc**2)
    azimuth = (2*np.arctan(X_sat_loc/(Y_sat_loc +d_hori)))%(2*np.pi)
    elevation = np.arcsin(Z_sat_loc/d)
    return (elevation,azimuth)
```

Une fonction coord_geodesique_bis qui calcul les coordonnées géodésiques à partir de coordonnées cartésiennes.

```
def coord_geodesique_bis(X,Y,Z):
    a = 6378137.0
    f = 1/298.257222101
    b = a*(1-f)
    e = np.sqrt((a**2-b**2)/a**2)
    R = np.sqrt(X**2+Y**2+Z**2)
    mu = np.arctan((Z/np.sqrt(X**2+Y**2))*((1-f)+(((e**2)*a)/R)))
    lon = np.arctan2(Y,X)
    lat = np.arctan((Z*(1-f)+(e**2)*a*(np.sin(mu)**3))/((1-f)*(np.sqrt(X**
                                        2+Y**2)-(e**2)*a*(np.cos(mu)**3))
                                        ))
    return (lon,lat)
```

1.2.3 Plot

Pour le plot, nous allons utiliser tous les fichiers de lecture et de fonctions vus précédemment.

Nous définissons les paramètres constants.

```
a_galileo = fonctions.calcul_a(1.45*(10**(-4)))

i_galileo = np.radians(55.0919)

e_galileo = 0.0002373

n0 = 1.45*(10**(-4))

J2 = 1.08262652304819194e-3

vitesse_angulaire = -7.2921151467e-5
```

On va par la suite boucler sur le nombre de stations puis le nombre de satellites et enfin sur le linspace en temps.

Pour chaque station, on vide la liste de valeurs à plot.

```
for j in range(15):
    azi = []
    ele = []
```

On définit les paramètres qui vont varier de satellites en satellites

```
for i in range(26):
    w0 = np.radians(lecture_de_TLE.elements_keplerien("donn es/galileo.
                                        txt")['Argument du p rig e'][i]
                                        )

    W0 = np.radians(lecture_de_TLE.elements_keplerien("donn es/galileo.
                                        txt")['Ascension'][i])

    M_galileo = lecture_de_TLE.elements_keplerien("donn es/galileo.txt")[
                                        'Anomalie moyenne'][i]

    heure_tle = fonctions.tle_to_utc(float(lecture_de_TLE.
                                        elements_keplerien("donn es/
                                        galileo.txt")['Heure'][i]))
```

```
    date_tle = fonctions.t(fonctions.dateJulienne(2021,11,8,heure_tle))

    name = lecture_DORIS.NAME[j]

    X_sta = lecture_DORIS.X[j]
    Y_sta = lecture_DORIS.Y[j]
    Z_sta = lecture_DORIS.Z[j]

    lam,phi = fonctions_2.coord_geodesique_bis(X_sta,Y_sta,Z_sta)
```

Puis on boucle sur le linspace en temps en précisant ce qui va varier dans le temps.

```
for t in date:
    w_galileo = fonctions.var_omega(w0,n0,J2,a_galileo,e_galileo,i_galileo
                                    ,t,date_tle)

    w_galileo = fonctions.var_omega(w0,n0,J2,a_galileo,e_galileo,i_galileo
                                    ,t,date_tle)

    W_galileo = fonctions.var_OMEGA(W0,n0,J2,a_galileo,e_galileo,i_galileo
                                    ,t,date_tle)

    M_corr_galileo = fonctions.var_anomalie_moyenne(M_galileo,n0,J2,
                                    a_galileo,e_galileo,i_galileo,t,
                                    date_tle)

    E_galileo = fonctions.kepler_resolve(M_corr_galileo,e_galileo,1e-6)

    coord_cart_galileo = fonctions.coord_cart(i_galileo,w_galileo,
                                    W_galileo,e_galileo,E_galileo,
                                    a_galileo,t,vitesse_angulaire)

    X_sat = coord_cart_galileo[0][0]
    Y_sat = coord_cart_galileo[1][0]
    Z_sat = coord_cart_galileo[2][0]
    if fonctions_2.azimuth_elevation(X_sta,Y_sta,Z_sta,X_sat,Y_sat,Z_sat,
                                    lam,phi)[0]>0:
        ele.append(90-np.degrees(fonctions_2.azimuth_elevation(X_sta,Y_sta
                                        ,Z_sta,X_sat,Y_sat,Z_sat,lam,
                                        phi)[0]))
        azi.append(fonctions_2.azimuth_elevation(X_sta,Y_sta,Z_sta,X_sat,
                                        Y_sat,Z_sat,lam,phi)[1])
```

Pour finir on fait un polar plot à chaque sortie de boucle sur les stations.

```
fig = plt.figure()
ax = fig.add_subplot(projection='polar')
c = ax.scatter(azi,ele,c='red',s=1)
plt.title(f'Skyplot pour la constellation galileo depuis la station {name}
                                    ')
plt.show(block=False)
plt.pause(3)
plt.close()
```

Pour l'étape 3 nous avons juste fait quelques modifications sur le script précédent, on va considérer que la station est l'antenne sur l'ISS. Il nous faut donc ses coordonnées cartésiennes

et on peut appliquer la même méthode que dans l'étape 2.

Il n'y a plus de boucle sur les stations et on définit les paramètres constants à nouveaux :

```
# Param tres ISS

a_iss = fonctions.calcul_a(0.0011264398169714)

i_iss = np.radians(51.6456)

e_iss = 0.0003349

n0_iss = 0.0011264398169714

J2 = 1.08262652304819194e-03

vitesse_angulaire = -7.2921151467e-5

date = np.linspace(fonctions.t(2459526.5),fonctions.t(2459527.5),num=10000
                                        )

w0_iss = np.radians(lecture_de_TLE.elements_keplerien("donn es/iss.txt")[
                                        'Argument du p rig e'][0])

W0_iss = np.radians(lecture_de_TLE.elements_keplerien("donn es/iss.txt")[
                                        'Ascension'][0])



M_iss = M_iss = lecture_de_TLE.elements_keplerien("donn es/iss.txt")['
                                        Anomalie moyenne'][0]

heure_tle_iss = fonctions.tle_to_utc(float(lecture_de_TLE.
                                        elements_keplerien("donn es/iss.txt"
                                        )['Heure'][0]))
date_tle_iss = fonctions.t(fonctions.dateJulienne(2021,11,7,heure_tle_iss)
                                        )

# Param tres GPS

a_gps = fonctions.calcul_a(1.45*(10**(-4)))

i_gps = np.radians(53.8532)

e_gps = 0.0063356

n0 = 1.45*(10**(-4))

NAME = lecture_de_TLE.elements_keplerien("donn es/gps-ops.txt").index.
                                        tolist()
```

On va ensuite boucler sur les satellites de la constellation GPS:

```
for i in range(30):
    azi = []
    ele = []

    w0 = np.radians(lecture_de_TLE.elements_keplerien("donn es/gps-ops.
```

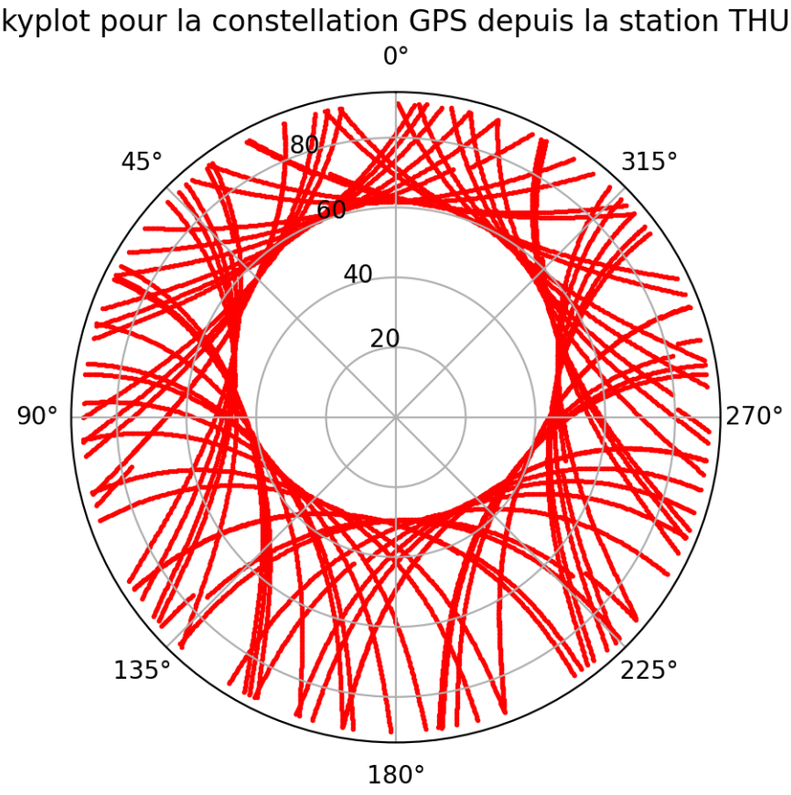
```
                                        txt")['Argument du p rig e'][i]
                                        )

    W0 = np.radians(lecture_de_TLE.elements_keplerien("donn es/gps-ops.
                                        txt")['Ascension'][i])

    M_gps = lecture_de_TLE.elements_keplerien("donn es/gps-ops.txt")['
                                        Anomalie moyenne'][i]

    heure_tle = fonctions.tle_to_utc(float(lecture_de_TLE.
                                        elements_keplerien("donn es/gps-
                                        ops.txt")['Heure'][i]))

    date_tle = fonctions.t(fonctions.dateJulienne(2021,11,8,heure_tle))
```

Puis sur le temps : (les coordonnées de la station varient aussi au cours du temps)

```
for t in date:

    w_gps = fonctions.var_omega(w0,n0,J2,a_gps,e_gps,i_gps,t,date_tle)

    W_gps = fonctions.var_OMEGA(W0,n0,J2,a_gps,e_gps,i_gps,t,date_tle)

    M_corr_gps = fonctions.var_anomalie_moyenne(M_gps,n0,J2,a_gps,e_gps,
                                        i_gps,t,date_tle)

    E_gps = fonctions.kepler_resolve(M_corr_gps,e_gps,1e-6)

    coord_cart_gps = fonctions.coord_cart(i_gps,w_gps,W_gps,e_gps,E_gps,
                                        a_gps,t,vitesse_angulaire)

    X_sat = coord_cart_gps[0][0]
    Y_sat = coord_cart_gps[1][0]
    Z_sat = coord_cart_gps[2][0]


    w_iss = fonctions.var_omega(w0_iss,n0_iss,J2,a_iss,e_iss,i_iss,t,
                                        date_tle_iss)

    w_iss = fonctions.var_omega(w0_iss,n0_iss,J2,a_iss,e_iss,i_iss,t,
                                        date_tle_iss)

    W_iss = fonctions.var_OMEGA(W0_iss,n0_iss,J2,a_iss,e_iss,i_iss,t,
                                        date_tle_iss)

    M_corr_iss = fonctions.var_anomalie_moyenne(M_iss,n0_iss,J2,a_iss,
                                        e_iss,i_iss,t,date_tle_iss)

    E_iss = fonctions.kepler_resolve(M_corr_iss,e_iss,1e-6)

    coord_cart_iss = fonctions.coord_cart(i_iss,w_iss,W_iss,e_iss,E_iss,
                                        a_iss,t,vitesse_angulaire)

    X_antenne = coord_cart_iss[0][0]
    Y_antenne = coord_cart_iss[1][0]
    Z_antenne = coord_cart_iss[2][0]

    lam,phi = fonctions_2.coord_geodesique_bis(X_antenne,Y_antenne,
```

```
                                        Z_antenne)

    if fonctions_2.azimuth_elevation(X_antenne,Y_antenne,Z_antenne,X_sat,
                                        Y_sat,Z_sat,lam,phi)[0]>0:
            ele.append(90-np.degrees(fonctions_2.azimuth_elevation(
                                                X_antenne,Y_antenne,
                                                Z_antenne,X_sat,Y_sat,
                                                Z_sat,lam,phi)[0]))
            azi.append(fonctions_2.azimuth_elevation(X_antenne,Y_antenne,
                                                Z_antenne,X_sat,Y_sat,
                                                Z_sat,lam,phi)[1])
```

1.2.4 Quelques skyplots



Skyplot pour GPSBIIR-11(PRN19) depuis l'antenne sur l'ISS